JDMS

Journal of Defense Modeling and Simulation: Applications, Methodology, Technology
2015, Vol. 12(4) 439–456
© The Author(s) 2015
DOI: 10.1177/1548512915591050
dms.sagepub.com
SAGE

Virtual machine-based simulation platform for mobile ad-hoc network-based cyber infrastructure

Srikanth B Yoginath[1], Kalyan S Perumalla and Brian J Henz

Abstract
In modeling and simulating complex systems, such as mobile ad-hoc networks (MANETs), in defense communications, it is a major challenge to reconcile multiple important considerations: the rapidity of unavoidable changes to the software (network layers and applications), the difficulty of modeling the critical, implementation-dependent behavioral effects, the need to sustain larger scale scenarios, and the desire for faster simulations. Here we present our approach in successfully reconciling them using a virtual time-synchronized virtual machine (VM)-based parallel execution framework that accurately lifts both the devices, as well as the network communications, to a virtual time plane while retaining full fidelity. At the core of our framework is a scheduling engine that operates at the level of a hypervisor scheduler, offering a unique ability to execute multi-core guest nodes over multi-core host nodes in an accurate, virtual time-synchronized manner. In contrast to other related approaches that suffer from either speed or accuracy issues, our framework provides MANET node-wise scalability, high fidelity of software behaviors, and time-ordering accuracy. The design and development of this framework is presented, and an actual implementation based on the widely used Xen hypervisor system is described. Benchmarks with synthetic and actual applications are used to identify the benefits of our approach. The time inaccuracy of traditional emulation methods is demonstrated, in comparison with the accurate execution of our framework verified by theoretically correct results expected from analytical models of the same scenarios. In the largest high-fidelity tests, we are able to perform virtual time-synchronized simulation of 64-node VM-based full-stack, actual software behaviors of MANETs containing a mix of static and mobile (unmanned airborne vehicle) nodes, hosted on a 32-core host, with full fidelity of unmodified ad-hoc routing protocols, unmodified application executables, and user-controllable physical layer effects, including inter-device wireless signal strength, reachability, and connectivity.

Keywords
Network simulation, discrete event simulation, virtual machine, hypervisor, ad-hoc networks, virtual time, real time, scheduling, synchronization

1. Introduction

While the bulk of wireless networking research being conducted in academia and industry is focused on technologies such as cellular that have fixed infrastructure, the military has a requirement for robust mobile ad-hoc networks (MANETs). Although some aspects of cellular networks carry through to MANETs, such as radio frequency (RF) propagation models, the commercial and military technologies are distinct in many ways, with MANETs posing unique challenges. In MANETs, access points are mobile and coverage may vary widely within a region and across

[1]Computational Sciences and Engineering Division, Oak Ridge National Laboratory, USA
Computational and Information Sciences Directorate, U.S. Army Research Laboratory, USA

Corresponding author:
Kalyan S Perumalla, Computational Sciences and Engineering Division, Oak Ridge National Laboratory, Oak Ridge, TN 37831-6085, USA.
Email: perumallaks@ornl.gov

time. At times, the access points will cluster together, leaving parts of a geographic region with sparse coverage and other parts with compromised service due to competition for the available spectrum as communication channels become saturated. On the battlefield, relocating network nodes to provide better coverage is difficult or nearly impossible. With the large investments the US Department of Defense (DoD) is making in MANET programs, such as the Joint Tactical Radio System (JTRS), the Advanced Tactical Data Link (ATDL), and so on, it is critical to understand the behavior of MANETs under various conditions. The high cost of prototype hardware manufacturing makes detailed simulation a valuable tool for the development and analysis of new MANET waveforms.

One distinction in predicting the performance of a MANET versus that of a wired network is the simulation of interference in the physical layer. With many devices possibly sharing a common communication channel, the calculation and accurate simulation of noise levels is very sensitive to correct timing. For instance, if a distant transmitter is generating a signal during the same time that another signal is being received, this noise must be taken into account during this period and must be incorporated in the computation of the bit error rate (BER) for the received data. If the hypervisor is unable to accurately provide timestamps to incoming packets, it is impossible to accurately duplicate this behavior. The sharing of a physical central processing unit (CPU) core by multiple virtual CPUs (VCPUs) under a traditional hypervisor will not provide the guest operating system (OS) with an accurate snapshot of the spectrum usage during packet reception and calculation of signal-to-noise ratios.

1.1 Simulation challenges and the conventional approach

The simulated infrastructures of interest are characterized by detailed effects arising at multiple layers, such as wireless signal propagation effects at the physical layer, channel control protocol effects at the media access control layer, and complex ad-hoc routing dynamics at the next network and transport layers. Equally, or more importantly, the actual *applications*, such as voice-over-internet protocol (VOIP), video streaming, command-and-control state updates, and so on, are themselves hard-to-model complex functionalities that need to be exercised and tested on top of all the aforementioned layers.

The conventional *simulation* approach would develop models of the relevant components of each layer, integrate the models in a simulation package, and execute them in a simulator. Thus, for example, voice streaming applications of interest need to be modeled, and models of ad-hoc routing protocols need to be developed, and so on. For each variant of the ad-hoc network routing algorithm that is already available in software (e.g., as actual implemented network packages in Linux, such as Adaptive Online Distance Vector routing, or Optimized Link State Routing), new simulator-specific models need to be developed before they can be used in experimentation or testing. Similarly, for every traffic source or destination, the application(s) need to be abstracted and modeled.

However, a critical hurdle in this approach is that many of these functionalities are hard to model because either they are in binary executable forms (e.g., malware codes, monitoring tools, and third-party vendor-supplied software packages), or they evolve too rapidly (e.g., new encoding algorithms, new attack vectors, or new version releases), or their coded functionalities are extremely difficult or cost-prohibitive to duplicate via another effort (e.g., voice quality control algorithms, hashing algorithms). For complex applications and protocols, this approach of developing models of actual implementations can thus be prohibitively expensive in some cases, and simply be practically infeasible in others. Moreover, key features, such as repeatability, determinism, and speed of simulation, are expected from simulation-based experiments despite the complex interaction of components, which place additional demands on the simulation approach.

Consequently, the challenges for the next generation of emulators and simulators for testing and evaluation of net-centric systems include (1) the need to capture important implementation-specific behavioral detail, (2) the nearly impossible, very expensive, and time-consuming task of developing merely abstracted models of complex, real-life software-dominant systems across the network stack from physical up to application layers and overlay/peer-to-peer networks, and (3) the need to retain determinism and repeatability of tested scenarios simulated using modern multi-core processing host platforms.

1.2 Benefits of our approach

To help address the challenges in simulating complex communication networks like MANET, here we present a new time-synchronized, virtual machine (VM)-based parallel simulation approach. The approach presented in the rest of this paper makes it possible to use exactly the same subject (net-centric) system(s) for the dual purposes of testing and evaluation as well as deployment, with the following benefits: (a) assurance from having tested the actual system implementation itself; (b) avoidance of unintentional or unknown disparities between tested and deployed systems; (c) significant cost savings from the elimination of a modeling phase; (d) increased rapidity of the testing and evaluation process, thereby reaching actual deployments faster; (e) nearly eliminating the need for the testing personnel to have deep understanding of the actual

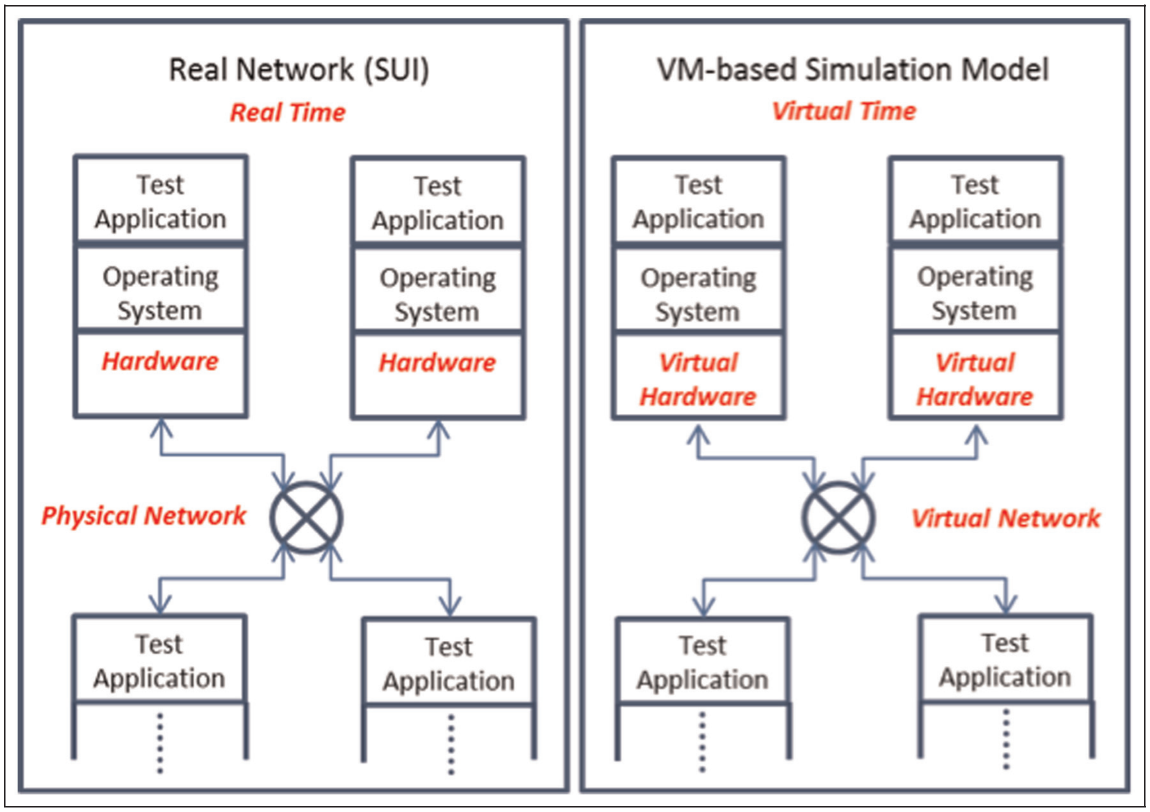

Figure 1. Differentiating the computational infrastructure between the system under investigation and its virtual machine (VM)-based simulation model.

systems being deployed in great detail, except to the extent needed for actual testing, thereby increasing productivity and agility of deployment; and (f) minimizing the loss of accuracy of testing and evaluation by avoiding the need for developing simplified abstractions of complex software components.

1.3 VM-based simulation platform

VMs are useful for high-fidelity cyber-infrastructure simulations because of a wide range of significant advantages they offer. Any OS with its implementation of the entire communication protocol stack could be put under simulation test without abstraction and re-implementations of the protocols. New distributed applications or protocols could be developed and tested on any target OSs. Furthermore, the software used as a model in testing can be directly ported to real systems with little modification. This direct use of simulation models as replicas of the complex software systems offers greater fidelity in comparison with abstracted and modeler-implemented simulation systems. The actual time involved in developing a cyber-infrastructure simulation study is also reduced drastically as software is reutilized for dual purposes of simulation and deployment. Given these advantages, it is natural for the cyber-infrastructure simulation community to actively pursue research in VM-based cyber infrastructure simulation platforms. However, as demonstrated by Yoginath et al.,[1] most of the VM-based simulation/emulation platforms have ignored some of the key distinctions between the real and virtual systems, and thus have compromised the correctness of the simulation/emulation models.

1.3.1 Real versus virtual operation. Figure 1 shows the key distinctions between cyber-infrastructure and its simulation model developed using VM-based technologies. There are two key differences: (a) real host versus virtual host; and (b) real network versus virtual network. These core distinctions also define the concepts of real time and virtual time.

In contrast to an OS instance running directly on the hardware, an OS running on VM platforms does not have exclusive access to hardware. Instead, the physical hardware is shared among multiple hosted OS instances. Hence, an OS instance hosted on the VM platform only interacts with the software-based virtual components rather than the physical hardware. The VM platform or the hypervisor multiplexes multiple virtual components onto the physical hardware components based on some principle of resource sharing. While this distinction between the cyber-infrastructures operating on real hardware and the simulations using a VM-based platform is relatively simple, its implications are subtle on the notions of real time and virtual time when the CPU resources are shared. A mismatch between virtual time and real time arises even in the case of one-to-one mapping from physical to virtual resources, and gets accentuated on larger multiplexing ratios. Thus, the problem is fundamental in nature, impacting the correctness of the simulation.

Similarly, the distinction between the virtual network and the physical network (ad-hoc network) needs to be carefully considered. The VM platform only provides the essential software/hardware-assisted infrastructure support for the VMs hosted on the hypervisor to communicate and interact with OS-hosted applications of VMs. A simple setup to ensure interaction between VMs on a single physical node is through a software bridge. Every VM connected to this software bridge is visible to other connected VMs all the time, which is in complete contrast with the ad-hoc networks, where the nodes get connected and disconnected based on their position in space, which changes in time. Hence, it is necessary to accommodate ad-hoc behavior of the physical network into the all visible virtual network.

1.3.2 Real time versus virtual time. The real time corresponds to the actual wall clock time consumed by a set of the real-hardware and real-network operations. Since the physical resources are shared in the VM platforms, the virtual time corresponding to a VM is only the aggregate time periods through which the VM actually engages the physical compute resource. These distinctions are to be understood clearly in order to effectively utilize VMs for cyber-infrastructure simulations/emulations.

Most of the published works on the cyber-infrastructure simulation/emulation models are tightly coupled with the real time. The most common approach used in the development of VM-based network simulation/emulation has been to interface actual applications executing on VMs with network simulators, and ensuring that the network simulations execute in (or faster than) real time. This approach makes the fundamental assumption that the VMs operate in real time. However, this assumption cannot be supported even when there exists a 1:1 mapping between the physical compute resources and virtual compute resources.

To understand the incorrectness in this assumption, first the composition of the simulation time in the VM-based simulation model should be understood. The simulation time can be broadly divided into (a) computation time an (b) idle time. The computation time is the number of CPU cycles utilized by the VCPU, which is accounted by all the VMs. The idle time is the time the CPU spends without doing any useful work. The idle time is not captured by the VM platforms, since the physical resources are shared; hence, when no useful work is being performed by a VM, its VCPU is swapped out and replaced with another VCPU. Note that this functional behavior is unique to VMs and does not apply to actual physical devices being modeled. This functional sharing aspect of the VM platform make conventional real time-based simulation/emulation models unreliable for cyber-infrastructure simulation/emulation.

Further, by using real time, the ordering sequence of operations in distributed applications executing over VMs gets affected. This is because the CPU resource multiplexing entity in the hypervisor is agnostic to any notion of time and hence this results in time-order errors in real time-based cyber-infrastructure simulations.

Hence, to ensure the correctness of the cyber-infrastructure simulation it is essential to (a) move away from reliance on real time in VM-based simulations/emulations, (b) account for both compute time and idle time during simulation/emulation, (c) provide a simulation time awareness to the resource scheduler in the hypervisor, so that simulation time-order errors are controlled or eliminated, and (d) provide a simulation time-ordered communication of inter-VM packets among the VMs simulating the cyber-infrastructure.

Here, we describe our new approach to track, maintain, and evolve in virtual time the VMs in a virtual time-order manner. We refer to our approach as VM-based simulations, even though exact implementation of the software components is used. This is because we utilize virtual time of VMs as the simulation time, and ensure simulation time-ordered execution in our simulations. We also account for the idle time to derive the simulation time, as discussed in Section 2.2. Since virtual time of the VMs determines the simulation time, in this paper, the *virtual time* and the *simulation time* are synonymously used.

1.3.3 Real versus virtual network. To model the real ad-hoc network using the virtual network where all VMs are visible/connected to each other, we need methodologies to model (a) varying network connectivity between VMs, (b) varying network bandwidth and packet drops between VMs, AND (c) varying latency between VMs. Modeling these would be sufficient because the change in position over time can impact THE MANET only in terms of connectivity, bandwidth, packet drop, and latency. By varying the network connectivity among the VMs, we can model the movement of nodes in MANET, as shown in Figure 2, in which the network effects due to mobility are modeled in terms of dynamic changes to connectivity at the link (level 2 media access) layer. Once the connectivity is

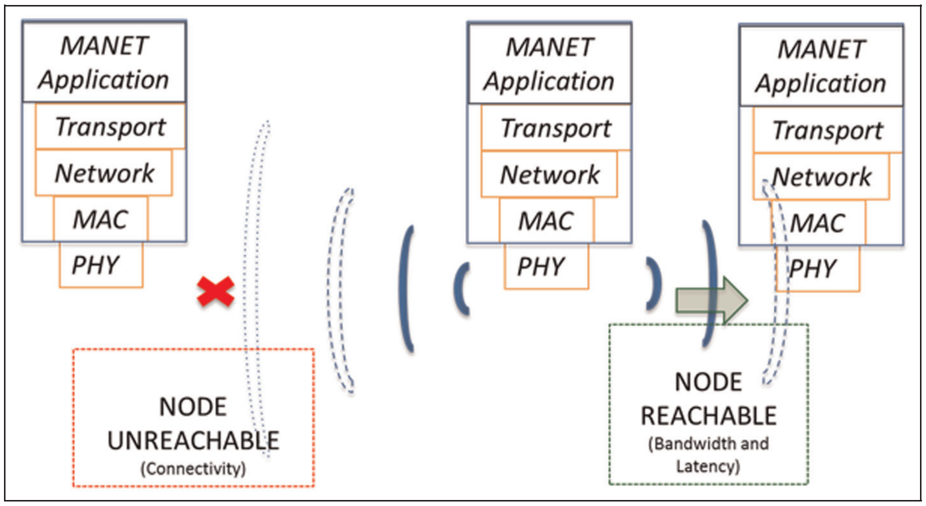

Figure 2. Modeling ad-hoc network of mobile ad-hoc network (MANET).

established, the quality of communication between any pair of nodes is determined by the physical media model and the distance between the mobile nodes. By varying the communication bandwidth, packet-drop rates, and latency, we can closely replicate effects of the physical media or the distance between the connected mobile nodes in the virtual network.

Hence, it is necessary for the MANET simulation platform to provide a means to alter the connectivity, bandwidth, packet-drop, and latency parameters on demand. We can expect higher accuracy of MANET simulation for lower resolution of this period. Further, all the periodic updating instrumentation is required to happen with minimal or no interference to the simulation execution, while ardently following the simulation time order during execution. In this paper, we describe our approach to model the ad-hoc network behavior in the VM virtual network using various networking sub-systems.

1.4 Xen hypervisor

All the methodology and techniques described in this article were implemented and experimented with our prototype network simulation system named NetWarp, which is a high-fidelity network simulator built over VMs and virtual networks. For virtualization support in NetWarp, we use the Xen hypervisor[2] (in para-virtualization mode). Xen refers to VMs as Guest Domains or DOMs. Each DOM is identified by its DOM-ID. The Control-VM referred to as "DOM-0" affords special hardware privileges and its service is utilized to provide networking and input/output (I/O) support among VMs. The Control-VM is also used to launch, manage, and monitor VMs. Each VM has its own set of virtual devices, including virtual multi-processors called VCPUs. The credit-based scheduler (CSX) is the default Xen scheduler, which schedules VCPUs onto physical CPU cores (PCPUs) based on the fair-sharing principle. CSX uses *credits* for every VM; these credits are expended as the VM's VCPUs are scheduled for execution.

1.5 Related work

Simulations and emulations are useful in the analysis of networked systems of various kinds. Analytical models and network simulation tools, such as NS2 or NS3[3] and OPNET,[4] can be categorized as network-centric simulation models in which abstractions of the end-nodes and the network for modeling offers good scaling, speed, and accuracy. Very large-scale simulations are enabled via parallel computing in simulators such as GTNets,[5] SSFNet,[6] GloMoSim,[7] and PDNS.[8] In all these simulation tools, the simulation time is completely different from real time or wall clock time. On the other hand, emulation tools rely on real time (essentially equating virtual time and real time) to incorporate network behaviors at very high fidelity. Tools such as Dummy-net,[9] ENTRAPID,[10] ModelNet,[11] Emulab,[12] and ROSENET[13] fall into this category. To differentiate them from simulators, they have been largely referred to as network emulation tools or emulation testbeds in the literature.

The concept of time dilation[14] was introduced in emulations with goals similar to ours but for older uni-core processor platforms. In time dilation, any desired bandwidth or latency characteristics of a modeled (simulated) network could be emulated over a physical (simulator) network that has higher or lower bandwidth or latency. This is achieved by simply scaling up/down the host processor clock rate and manipulating the time query points of the end-nodes/OSs. This was also referred to as time virtualization in subsequent literature. Using *resource* virtualization from conventional hypervisors, augmented with *time* virtualization from time dilation, various network emulation systems have been proposed, such as V-eM,[15] DieCast,[16] VENICE,[17] dONE,[18] and Time-Jails,[19] allowing flexibility in configuring the emulation setup. Note that although time dilation alters the perception of time for the end-nodes, the simulation pacing is still real time-based. We categorize these and similar[20] types of VM-based models, where real time reliance is necessary, as VM-based emulations. There are other sets of VM-based models, where the model maintains its own simulation time and is not polluted by real time, such as NetWarp,[1,21,22] SliceTime[23] and OpenVZ-based virtual time systems,[24] we refer to these VM-based realizations as VM-based simulations.

MANET-specific emulation tools, such as the Extensible Mobile Ad-hoc Network Emulator (EMANE) from DRS Cengen,[25] have also been recently developed. EMANE contains a number of wireless *network emulation modules* (NEMs) that model the *physical* (PHY) and layer2 or MAC layers of the network stack. For high fidelity and ease of modeling, EMANE applications are hosted on VMs.[26] The EMANE application hijacks the communication packets from VM network interfaces and passes them through virtual physical and MAC layers to enforce the mobility characteristics. To enforce emulated network conditions, the path loss information is periodically communicated to the relevant EMANE modules. Since EMANE uses real time as the simulation time, the two notions of time pollute each other. Virtual time periods get added to real time for scheduling network activity into future (e.g., virtual wireless link latency added to current real time to determine the receipt time at destination). Moreover, in addition to EMANE objects, there are multiple independent components that are communicating in real time, which are artificial instrumentations that do not exist in a realistic scenario and their asynchronous computational effects incorrectly perturb the emulation timing. The virtual time-ordered framework described here avoids these pitfalls by clearly separating the virtual time and real time for computation and by separating the virtual network from real network.

There is also a relation of the scheduler presented here to the virtual time-aware scheduler for efficiently executing discrete event simulation (DES) on cloud computing and VM platforms.[27] Let the present work be denoted as ‘‘virtual machines over discrete event simulation’’ (VM-on-DES) and the other work as ‘‘discrete event simulation over virtual machines’’ (DES-on-VM). Then, VM-on-DES dynamically obtains virtual time from the DES-based hypervisor, while DES-on-VM supplies virtual time from DES to the hypervisor.

1.6 Contributions

This paper is a consolidation and a significant extension to work reported in our previous conference publications.[1,21,22] Our previous publications on VM-based network simulations introduced the basic concepts and sub-system prototypes necessary to realize time-synchronized execution of VM-based simulations. This paper presents a complete solution specific to MANET operation and needs. While the basic approach, concepts, and prototypes are borrowed from our previous conference articles, the following are new in this journal article: application to ad-hoc networking, integrated execution of the virtual scheduler and virtual network sub-system, a new, analytically tractable benchmark with analysis to derive a theoretical solution to verify correctness, a new variant of our previous Cyber-security benchmark customized for the MANET environment, an entirely new set of benchmark runs, experimental results, and their corresponding analyses.

1.7 Organization

The rest of the article is organized as follows. Section 2 describes the design and implementation of our simulation framework. In Section 3, we present the benchmark applications, and analyses of controlled benchmark configurations for theoretically correct results. In Section 4, a performance study is presented with results from execution of the benchmark scenarios demonstrating the low variability and significantly higher accuracy of our approach. We identify our future work and conclude in Section 5.

2. Design and implementation

To effectively model and simulate MANET operations using VMs, we require (a) virtual time tracking, maintenance, and evolution, (b) virtual time-ordered execution, (c) virtual time-ordered communication, and (d) ad-hoc networking and mobility feature support.

2.1 Virtual time tracking, accounting, and advancing

To address the challenge of tracking, accounting, and advancing virtual time of VMs, we designed a scheme of maintaining a virtual clock for each VCPU core that advances based on the VCPU core utilization. This essentially results in multiple virtual timelines corresponding to the number of VCPU cores supported by the VM. For the purpose of network simulation, these multiple VCPU core timelines must be consolidated to a single VM timeline of a single MANET node. Similarly, a single global simulation timeline must be consolidated from the multiple VM timelines on the simulator host node.

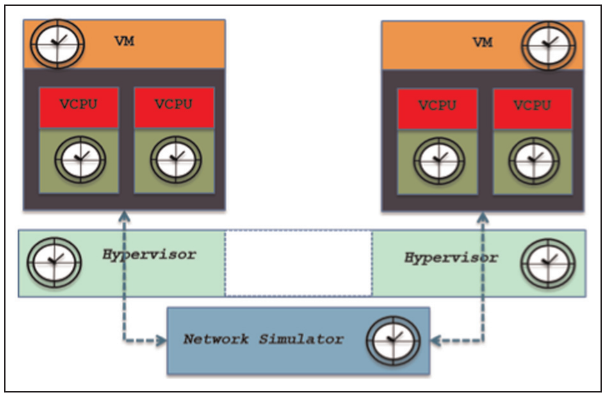

Figure 3. Virtual time accounting: a virtual clock per virtual central processing unit (VCPU), per DOM, and per node.

In Figure 3, the three conceptual timelines are shown as clocks in VCPUs, VMs, and the hypervisor. Each clock represents a distinct virtual timeline; the hypervisor scheduler performs synchronization across the timelines.

1. Local Virtual Time (LVT): timeline corresponding to each VCPU that advances in discrete time instants as the VCPU consumes PCPU cycles (measured in the units of ticks).
2. VM LVT (VM_LVT): timeline corresponding to each VM that is calculated using the LVTs corresponding to its VCPUs and advanced in discrete time as its corresponding VCPU LVTs advance.
3. System or Node LVT (SYS_LVT): global timeline corresponding to the entire simulation, with the VMs as the end-nodes connected through the controlled virtual network of the hypervisor environment.

2.2 Virtual time-ordered execution

As previously mentioned, the strategy in hypervisor scheduling used for compute resource sharing among the VMs is fair share-based and is performed without any notion of THE time-ordering principle. However, the simulations need to progress in the order of their simulation time, violating which leads to time-order errors (events of the future in one VM incorrectly affecting the events in the past of other VMs). To simulate a network behavior without time-order errors, it is essential that the events (such as packet arrival and departure events) be processed in simulation-time order, using the previously defined timelines. Simulation time-ordered scheduling of the VCPUs onto the PCPUS of the physical host can achieve time-ordered execution of all the VMs. The multiplexing time granularity of the VMs can be made as fine as necessary to achieve any desired degree of accuracy (e.g., in practice, even a scheduling time unit in the microsecond range can dramatically reduce or eliminate errors). Hence, we replaced the default fair-share-based hypervisor scheduler with THE Least-LVT-First (LLF) scheduler. Figure 4 shows the Symmetric Multi-Processing (SMP) schedulers using the run-queues for each PCPU to schedule the VCPUs on to PCPUs. The SMP schedulers pick the lowest VCPU_LVT value from the local run-queue and check for any VCPUs lower THAN VCPU_LVT in the peer run-queues before picking a VCPU for scheduling.

Another significant aspect of this design is that it accounts for the idle cycles of the VCPUs in the simulation time without losing actual CPU cycles in the process. Note that, when the VCPUs are idle, their VCPU_LVT values remain unchanged and their VCPU_LVT values increase as they utilize PCPU cycles. This results in a staggering of LVT values for different VCPUs within the same VM, as shown in the Figure 5. The hypervisor scheduler's LLF policy ensures that the VCPU with the least VCPU_LVT value is always picked. If the VCPU_LVTs lag with respect to others, it is just because they HAVE BEEN idle for a significant period. By pulling the VCPU_LVTs periodically to the maximum of VCPU-LVTs, we can account for the idle time in our simulation time. This synchronization in our MANET simulation is performed by the NetWarp Network Control (NNC), discussed in the next section. Further, details on the implementations of the NetWarp hypervisor scheduler for Xen (NSX) can be found in Yoginath and Perumalla.[21]

2.3 Virtual time-ordered communication

The hypervisors provide a variety of means to set up a virtual network to support the interaction across the hosted VMs. We use a private bridge that isolates the VMs involved in the simulation from the privileged VM (Control-VM). By controlling the network, we would have the capability to introduce any simulated *delay* on any transiting packet based on its source and destination addresses as required, and we dictate when a packet should be delivered to its destination. This enables us to enforce virtual time order. The virtual network control is intended to provide the following:

a) introduce a virtual delay without making a (byte-)copy of the packet buffer, or moving the transiting packets from kernel to user space;
b) introduce virtual time delays that may be varied on a per-packet basis; the delay specification may be static (e.g., for wireline networks) or dynamic (e.g., for MANETs);
c) minimal overhead while processing the trapped transiting packets.

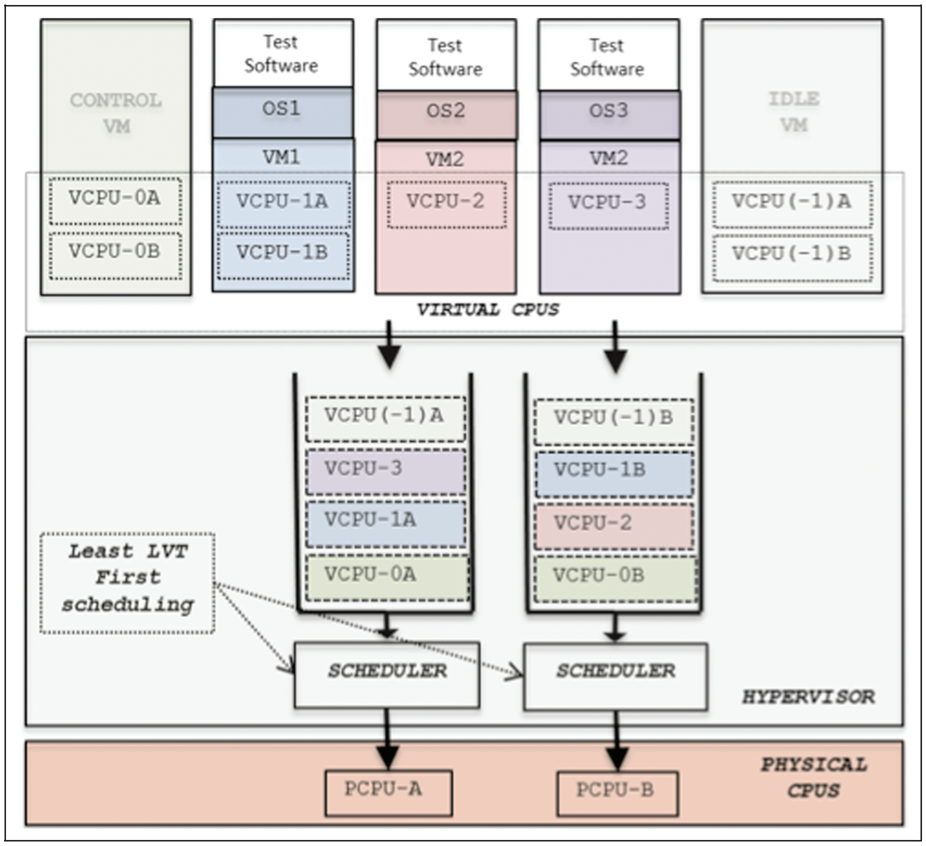

Figure 4. Virtual time-ordered scheduling.

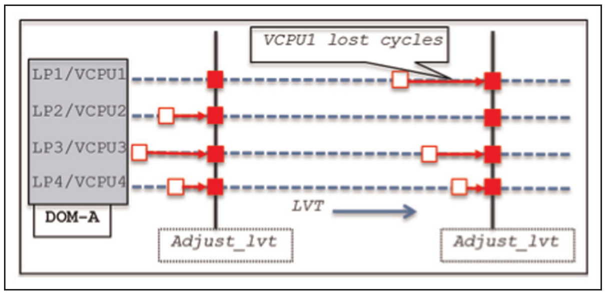

Figure 5. Accounting for idle time.

2.3.1 ***NetWarp Network Control.*** The NNC is a multi-threaded module built using THE Netfilter[28] library. Multiple Netfilter queues (NFQS), along with their corresponding service-threads, equaling the number of VMs (specific to the simulation scenario) are used in NNC. The iptables[29] rules in the CONTROL-VM are set such that all the transiting packets are routed to a single NFQ, with 0 queue identifier (qid). In Figure 6, this functionality is schematically presented using directional pointers suggesting the path of packet movement from a VM application to the front-end network device and then to its back-end counterpart before being enqueued in the NFQ with qid=0. The corresponding thread (service-thread-0) determines and marks the packet with the emit time before setting a NF_QUEUE verdict on the packet that results IN the enqueuing of the packet into a NFQ, with qid=1 (the next higher queue identifier). The first service-thread performs only this operation and, hence, it continuously processes the arriving packets without introducing any additional delay other than the processing itself.

Apart from service-thread-0, all the other service-threads (corresponding to the other NFQs) on receiving the packet query for the simulation time and check if it is greater than THE (emit time – INT_DELAY) value. If it is equal or greater then the packet is released from the NNC sub-system by issuing NF_ACCEPT verdict, as shown by the third service-thread. If the destination VM's virtual time has not advanced to the emit time yet, then the corresponding service-thread generates an event with an event time of (simulation time + INT_DELAY), inserts the event to the eventlist, and then blocks itself waiting on a signal from the scheduler-thread. In Figure 6 the solid line represents the service-thread receiving and

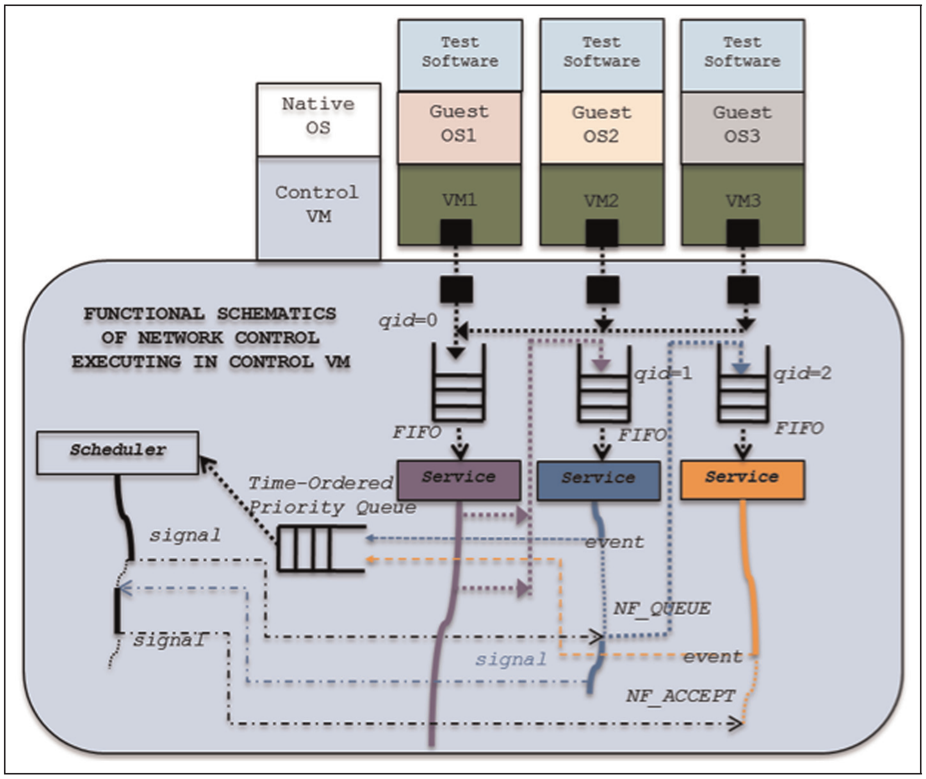

Figure 6. Virtual time-ordered network control.

subsequently processing of the transiting packet, while the thin dotted line represents THE service-thread waiting for a signal from its peer. The INT_DELAY mentioned previously refers to intermediate delay and is computed as

$$\text{INT\_DELAY} = \text{MIN\_DELAY} + \text{ceil}\left(\frac{\text{MAX\_DELAY}}{\text{NUM\_DOMs} - 1}\right)$$

where MIN_DELAY is a constant and MAX_DELAY is the maximum of the range of delays to be enforced by NNC on a transiting packet. With segmented intermediate delays we ensure that a transiting packet is released from NNC before it reaches the last NFQ, and it also ensures that the specified delay is introduced in its transit. Also, note that all delays are enforced in terms of *simulation time*.

The *scheduler-thread* continuously processes the events in *event time-orde*r and signals the respective thread when the *simulation time* advances to the *event time* and waits for the signal from the signaled *service-thread* before processing with next event. On receiving the signal from the scheduler-thread the *service-thread* enqueues the packet into a NFQ with next higher *qid*, using the *NF_QUEUE* verdict. Thus, at every NFQ, the *service-thread* either releases the packet or introduces a virtual time delay of INT_DELAY. Further, NNC also performs the *simulation time* synchronization across all VMs to the latest packet *emit time* and this results in a leap in *simulation time*. More details on the design and implementation of NNC can be found in Yoginath et al.[22]

2.4 Ad-hoc networking and mobility support

In a MANET, the mobile nodes establish or break communication links with their peers as they move, without relying on any infrastructural support to perform such actions. To model the movement of nodes in MANETs, a mechanism is necessary to introduce mobility behavior into the VM-based simulation. Further, the inter-node distance effects are reflected in terms of varying the bandwidth, latency and packet-drop parameters of the connected nodes. Since the connectivity and link parameters change over the course of the simulation, this inter-VM virtual network information needs to be periodically updated in simulation time across all the VMs. On obtaining such information, each VM can locally enforce the communication model via iptables rules. To enforce the ad-hoc networking support, the optimized link state routing Optimized Link State Routing Protocol's (OLSR's)[30] optimized implementation (OLSRD[31]) was used. Since the

```
#iptables rule to ensure no connectivity with mac address XXX
iptables -A INPUT -m mac --mac-source XXX  -j DROP

#iptables rule for bandwidth controlled connectivity with mac address XXX
iptables -I INPUT 1 -m limit --limit PKT_PER_SEC/sec -m mac --mac-source XXX -j ACCEPT
iptables -I INPUT 2 -m mac --mac-source  XXX -j DROP
```

Figure 7. *Iptables* rules enforcing connectivity and bandwidth.

VM can host an entire OS, the ad-hoc network support daemon (OLSRD) was installed on the OSs hosted on each of the VMs. When the new updates to network parameters such as connectivity, bandwidth, packet loss, or latency are enforced using the iptables rules, the MANET adapts to these changes dynamically with the help of OLSRD executing on each VM. In this way, an actual implementation of the OLSR was used to simulate the dynamic route adapting behavior amongst the VM-based MANET nodes.

2.4.1 Network connectivity model. In order to model MANET scenario-specific connectivity, an input file was used to provide the RF signal bandwidth between every pair of MANET nodes. An empirically specified low bandwidth was used as a cut-off value to demarcate the existence or absence of connectivity between any pair of MANET nodes. The multi-hop network was achieved by (a) communicating the connectivity information to VMs and (b) enforcing connectivity at the VM network interfaces.

To communicate the connectivity information to the VMs, a single bit (0 or 1) per pair representing the existence of connectivity between all pairs of MANET nodes was written in *Xenstore*. This data in the *Xenstore* is visible to all hosted VMs and, hence, can be read by all the VMs that represent the MANET nodes. To enforce the user-specified connectivity among the virtual network interfaces, every VM at boot time reads the *Xenstore*, and executes the necessary *iptables* rules specifying either to *accept* or *drop* the packets originating from peer network interfaces.

To support simulation of mobility, we developed a mobility daemon executing in the Control VM (DOM0) that refreshed connectivity (based on physical positions) information of MANET nodes at periodic virtual time intervals. Hosting this in DOM0 helps eliminate virtual time perturbation on the DOMs hosting the MANET nodes. The mobility daemon updates the *Xenstore* periodically with the connectivity information (at every t_1 s), and the VMs representing the simulated MANET nodes read the *Xenstore* periodically (at every t_2 s) and update their *iptables* rules to reflect the MANET connectivity. For correct operation, we ensure $t_1 > t_2$.

2.4.2 Network bandwidth model. Between any pair of connected MANET nodes, inter-node bandwidth is controlled by dropping packets at the receiver from its peer when a user-specified arrival rate threshold is about to be exceeded. This threshold was enforced by the VMs using iptables rules using the limit module. Similarly, nodes simply drop packets from peers with whom there is no connectivity in the simulated MANET. Thus, the VMs running as MANET nodes enforced these iptables rules based on the connectivity information read from the Xenstore. Figure 7 lists the necessary iptables rules that were used for this purpose. Note that this drop-based model does not perturb the traffic in the modeled scenario because the VMs are on a private software bridge whose broadcasting burden is excluded from the virtual time charged to the VMs.

2.4.3 Network latency model. The NNC previously discussed in Section 2.3 is used to enforce delay on packets transiting from one MANET node to another. The NNC executing in DOM0 utilizes iptables rules to trap the inter-VM packets and redirect them to NNC. IN Addition, with MANETs, the dynamic connectivity of the mobile nodes IS enabled via periodic exchange of broadcast packets (hello messages) between the nodes as per THE OLSR protocol. For performance reasons we do not redirect the broadcast packets to NNC and, hence, only the APPLICATION-SPECIFIC packets passed through NNC. Figure 8 illustrates the set of iptables rules enforced in DOM0 for this purpose.

3. Benchmark applications

3.1 Experimental MANET scenario

For experimentation purposes, we simulate a network connectivity based on a star topology centered on coordinates of 39.70°N, 111.22°W, located in Utah. The RF propagation path loss was computed using the LONGLEY–RICE algorithm with a cut-off value of –94 dB, the resulting network is used in our performance study. The MANET layout shown in Figure 9 is very similar to our test MANET on which we had earlier reported performance results from VOIP experiments in Yoginath ET AL.[1]

```
#iptables rule deleting the default rule of Xen for DOM0
iptables -D FORWARD -m physdev --physdev-in vifXXX -j ACCEPT

#iptables adding rule to accept the broadcast packets
iptables -A FORWARD -m physdev --physdev-in vifXXX -j -d 192.168.X.255 -j ACCEPT

#iptables rule to redirect the other non-broadcast packets to NNC
iptables -t mangle -A FORWARD -m physdev --physdev-in vifXXX -j -d !192.168.X.255 -j
NFQUEUE --queue-num 0
```

Figure 8. *Iptables* rules for NetWarp Network Control.

The MANET configuration with node 61 static at a position shown by the blue-shaded circle (around 61) is referred to as the ''static MANET'' setup in the rest of the paper. In the mobile MANET setup, node 61 represents a circulating unmanned airborne vehicle (UAV) that revolves around other static MANET nodes anti-clockwise, as shown in Figure 9. This UAV mobility results in the formation and tearing down of connections dynamically with the peripheral nodes in regular virtual time intervals.

Two benchmarks, namely (a) Constant Network Delay (CND) and (b) cyber-security, are used to evaluate the functional correctness of the MANET simulations. Being theoretically tractable to derive the receive order and relative message receive time or receive pattern, the CND benchmark is used to measure and compare the simulation time-order errors with theoretical expectations and with default VM platforms results (lacking virtual time-order execution and network control). The cyber-security benchmark is used to verify the mobility feature of the MANET, and to compare the infection spread rate and pattern in the static and mobile MANET scenarios.

3.2 Constant Network Delay benchmark

In CND, the MANET nodes are numbered by ranks from 0 to m–1, where m is the number of MANET nodes. Node rank m–1 initiates m–2 commands (C_1 to C_{m-2}) one after the other, destined to node rank 0. Each of these commands is routed via a single intermediate node before node hosting rank 0 is reached. Command C_i goes from node rank m–1 via node rank i to node rank 0. Thus, the first command would hop on node rank 1, the second command would hop on the node rank 2 process and so on up to node rank m–2. This simple benchmark verifies the correctness and measured the errors in *eunits* using the command actual receive order and the expected receive order as

$$E = \frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{m}\left|X_{ij} - O_{ij}\right|$$

where n is the number of replicated runs, m is the number of nodes, X_{ij} is the expected identifier of the jth command in the ith run, and O_{ij} is the observed identifier of the jth command in the ith run.

Note that cnd is described under the notion that every command makes exactly one logical hop before reaching the target. For manet experimentation with the cnd benchmark, we use the static manet setup, as shown in figure 9. Hence, a single logical hop of a command at the application level is not always equal to THE number of physical hops taken by the command in a MANET. The number of physical hops is actually dependent on the shortest distance between the source and destination nodes. For example from Figure 9, even though the first message passing from node 64 (rank 63) to node 1 (rank 0) logically makes a hop at node 2 (rank 1), the message actually makes THREE physical hops to reach node 2 from node 64, and a single hop from node 2 to node 1. In this particular example the message actually takes FOUR physical hops to fulfill a logical hop. Due to this mismatch between logical and physical hops, the message receive order might be different than the sent order. Hence to evaluate the correctness, we should first determine the correct receive order of messages arriving at rank 0.

The CND application benchmark uses blocking Message Passing Interface (MPI)[32] calls for implementing the logical messaging pattern. Hence, the MPI_Send routine would not return until the message sent is buffered at the destination host, as shown in Figure 10. Recall that a single logical hop in a MANET can correspond to multiple physical hops across VMs over the wireless links. Hence, the actual delay for every logical hop is proportional to the shortest path distance between the source-target pair. Using the shortest-path distances between the nodes, the correct receive order for the deterministic CND benchmark can be estimated. Further, the relative time of packet arrivals at the lowest rank can also be determined.

Let τ_i^f correspond to the first logical hop taken by the ith message to reach the ith ranked process and τ_i^s correspond to the second logical hop taken from the ith ranked process to the lowest rank (rank0). Then the reception time of the packet γ_i for any ith packet can be determined as

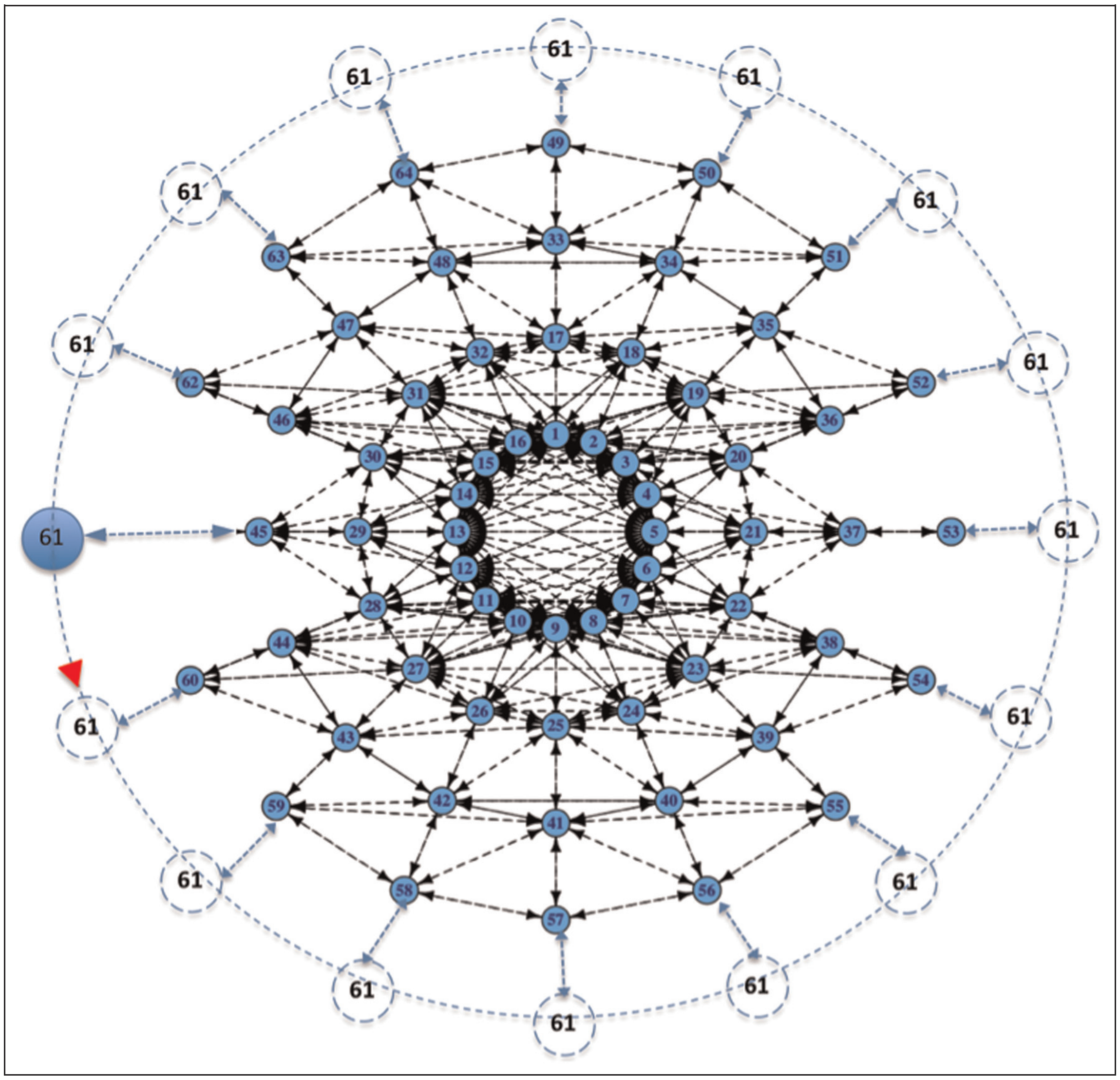

Figure 9. Mobile ad-hoc network setup. (Color online only.)

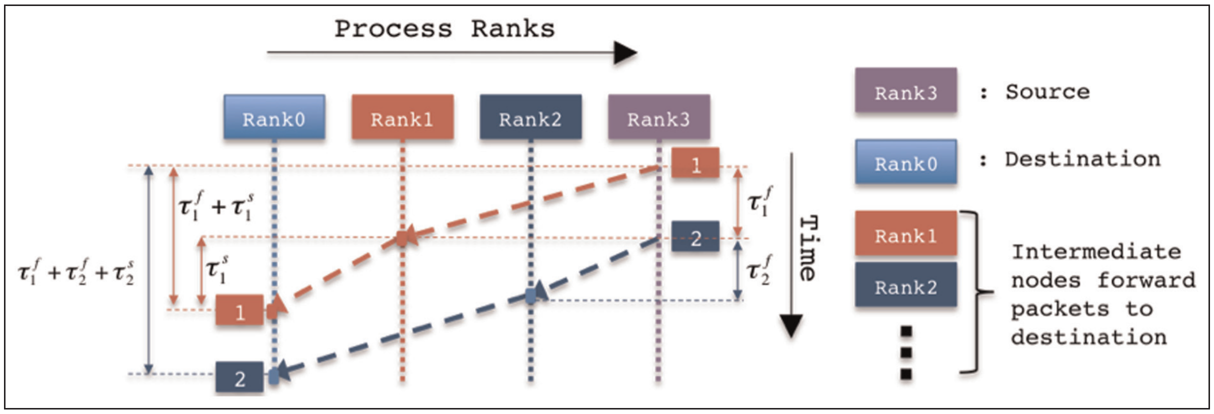

Figure 10. Constant Network Delay functional diagram.

$$\gamma_i = \tau_i^f + \tau_i^s, \tau_i^f = \sum_{k=1}^{i} \tau_k^f, \gamma_i = \sum_{k=1}^{i} \tau_k^f + \tau_i^s$$

where τ_i^f α is shortestPathDistance(highestRank, i) and τ_i^s α is shortestPathDistance(i, lowestRank).

Note here that τ_i^f is the summation of all previous first logical hop times. This is because we used the blocking MPI routine to send out the message and hence every generation of consecutive message suffers from this delay. Note τ_i^s does not incur any additional overheads. The number of physical hops in the shortest-path distance required to perform one logical hop is used to determine the receive order and to calculate the relative receive time of the arriving packets.

In addition to the receive order, an estimation of the receive time and hence the receive pattern can be obtained from this benchmark. These metrics of the CND benchmark were utilized for verification of correctness in the static MANET setup scenario.

3.3 Cyber-security benchmark

This benchmark emulates the behavior of a worm infection and its subsequent propagation across the hosts in an interacting multi-server and multi-client scenario. The worm propagation in the system proceeds as a simple instance of the well-known Susceptible-Infected "SI" model. The experiment involves a Vulnerable Service (VS), which listens to the requests from legitimate or non-malicious clients, referred to as Legitimate Clients (LCs). Upon receiving a request from any LC, a VS responds by spawning a service-thread that would subsequently transfer data of uniform-randomly distributed size ranging from 1 to 10 kB. Every LC generates requests to randomly selected service hosts, with a random inter-request interval ranging from 10 to 100 ms.

Each VM node involved in the experiment hosts a single instance of A VS and LC. The experiment starts with a single infected VS. This infected VS spawns a Shooting Agent (SA) executable. This SA process is exactly similar to LC in its operation. It picks a next-hop VS to infect and makes a request similar to a LC. In addition to the LC's normal data-transfer behavior, the malicious request from THE SA initiates a process that opens a backdoor-port for worm payload transfer, as shown in Figure 11. The payload file (4 KB) transfer makes the VS host infected. The newly spawned SA starts infecting its peers, similar to the infected (seed) VS. In this way, the worm infects all OF the VSs.

Note that the infection spread in this benchmark happens in two phases. In the initial phase, the worm infects the VS hosted on its *next-hop* nodes and this results in opening of the backdoor socket on the infected node. In the final phase the worm transfers payload to the already

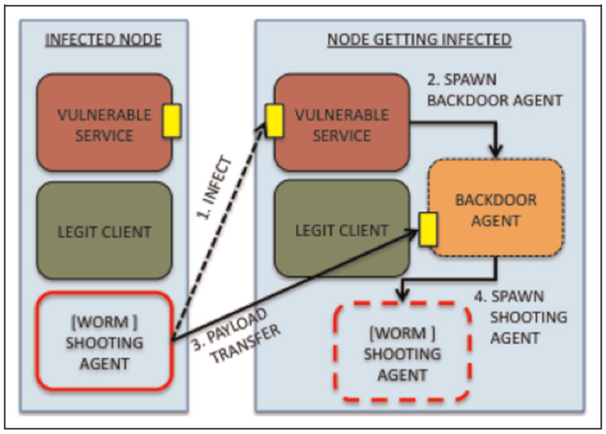

Figure 11. Worm infection and propagation.

infected service. Note that the same infected node might not be responsible for both initial and final phase infections. After the initial infection the SA tries multiple times to connect to the backdoor and transfer the payload; in the event of failure to connect, it moves on to infect the next node. However, if the backdoor is already open, some other SA from a different node might complete the final phase of infection with the payload transfer.

We conducted the experiments on 64 VMs and each VM starts up an instance of VS and LC. All LCs and the very first SA spawned by the seed VS are delayed by a 5-second sleep at their startup, to ensure all VSs are ready to accept requests. The mobility is triggered at the start of simulation by the benchmark application. This is accomplished by the initially infected service node, which uses the *Xenstore* to inform the mobility daemon about the simulator readiness, where the readiness is a state, when all the services reach a barrier after their corresponding initial setups.

4. Performance evaluation

4.1 Hardware and software

The experiments were performed on a Mac-Pro with two hex-core Intel® Xeon processors at 2.66 GHz, 6.4 GT/s processor interconnect speed with 32 G of memory. With hyper-threading enabled, Xen perceives 24 cores. With Xen creating 24 PCPUs to handle this, all our experiments view this system as a 24-core machine. OpenSUSE 11.1 executing over Xen-3.4.2 (built from source) was used on this hardware. The guest OSs hosted by VM were also OpenSUSE-11.1 distribution of Linux. We used the OpenMPI v1.4.3 distribution of the MPI to implement our test programs, in order to realize controlled point-to-point communications, for ease of experiment initiation, termination, statistics gathering, and to facilitate reusability.

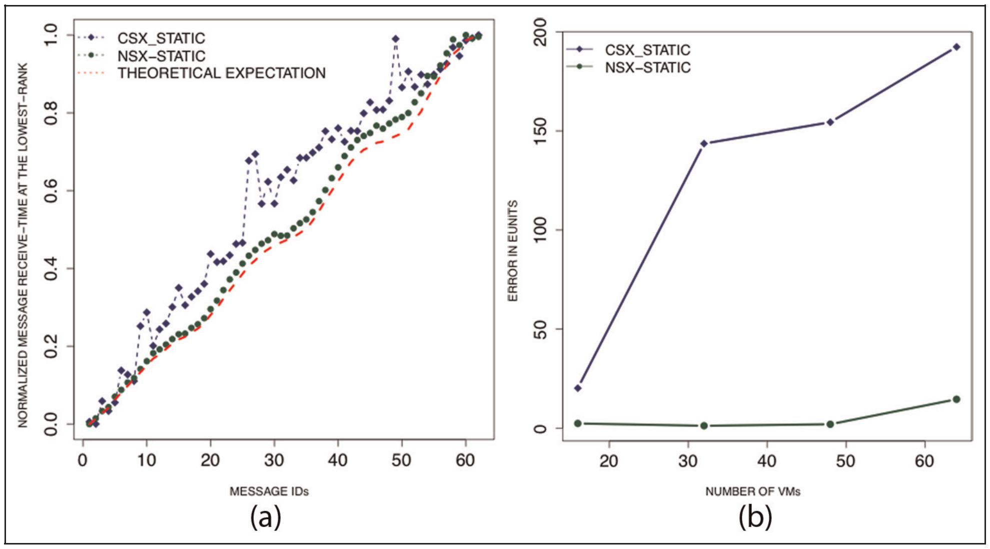

Figure 12. CSX and NSX receive-pattern comparison with theoretical expectations (a). CSX and NSX errors in *eunits* (b).

The CSX setup was configured to use four VCPUs in Control-VM and each weight of the Control-VM processors was 10 times larger than the weights of other VCPUs. During scheduling each VCPU was assigned a time-slice of 100 μs. While the bandwidth restriction of 300 packets/second per peer network device was enforced on all CSX runs, no latency was introduced (which is in favor of CSX). For the mobile scenarios a wall clock delay of 1 second was used for the mobile node (node 61) to hop from one peripheral node to another, as shown in Figure 9. Scaled wall clock time is used as the simulation time for the CSX runs and the scaling factor was determined by the ratio of $\frac{\text{number of PCPUs}}{\text{number of VCPUs}} = \frac{24}{64}$.

Similarly, the NSX setup was also configured to use four VCPUs in DOM0. To significantly favor the accuracy of CSX, the time slices for DOM0 VCPUs were increased by 10 fold in comparison with other VCPUs that used a 100 μs time slice. The bandwidth restriction, the same as CSX, was enforced in the NSX setup and the NNC was used to ensure latency control. A 10 ms delay was enforced on every (non-broadcast) transiting packet. The virtual time was collected through the *Xenstore* interface using *libxcutil* library. A virtual time equal to 1 second was enforced for the mobile node (node 61) to hop from one peripheral node to another.

4.2 CND benchmark results

Figure 12(a) compares the message receive pattern (relative message receive time) at the VM hosting the lowest-rank MPI process. To obtain the normalized message receive time, the receive time of each of the message is negated by the receive time of the first received message and divided by the receive time of the last received message. To obtain the theoretical curve, the weight for each in the MANET was set to unity and the shortest paths taken by the messages were used to estimate the receive time of the message arrivals. As observed, the NSX-STATIC, that is, the NSX runs for the static MANET scenario have a very close correspondence to the theoretically derived result, and in contrast the CSX runs show irregular pattern highly differing from expectations.

The errors measured from NSX and CSX runs in terms of eunits are presented in Figure 12(b). Increase in time-order errors (in eunits) with increase in the number of VMs in the experiment was observed with the CSX setup, and for the same experiment, almost no errors were observed in the NSX setup. In Figure 13(a), the variability of the CSX and NSX packet-receive simulation times with 95% confidence intervals are presented. A high variability in CSX and almost none in NSX runs can be observed.

For the static MANET scenario, we could theoretically derive correct receive order and receive pattern. However, similar theoretical derivation for determining the receive order and receive pattern is extremely difficult for the mobile MANET scenario. This is because of the physical path uncertainty between the source–destination pair, which makes it relatively harder to predict the exact number of physical hops a message could take to reach its destination at any given instance of simulation time. Hence, for the mobile scenario, only the CSX and NSX packet-receive simulation time with 95% confidence intervals are presented. Figure 13(b) shows very low variability in the packet-receive simulation time with the NSX setup. Further, in NSX we observe low variance for internal MANET nodes, while slightly higher variance is observed

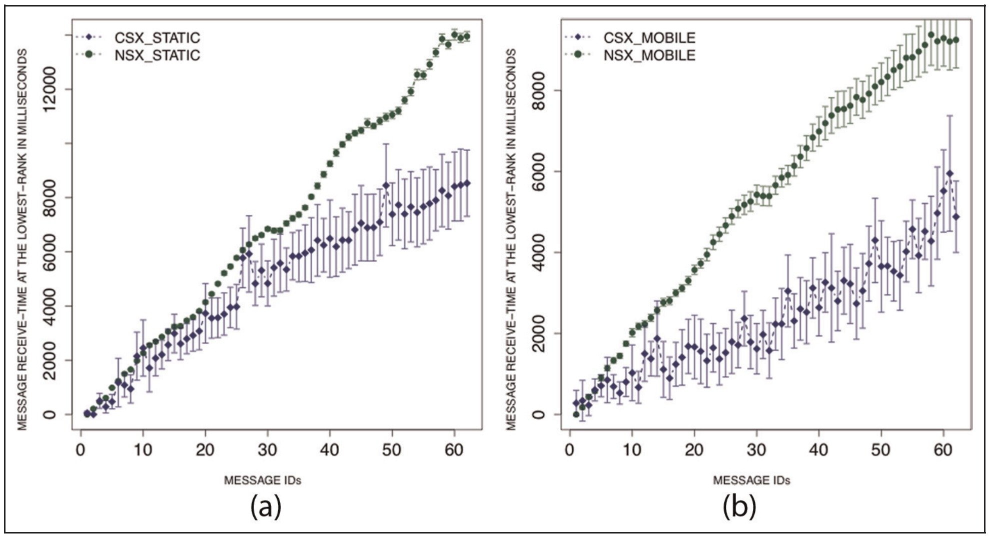

Figure 13. CSX and NSX simulation time in static mobile ad-hoc network (MANET) (a) and mobile MANET (b) scenarios with 95% CI (a).

at the peripheral nodes, as they are affected by the mobility of node 61. However with CSX setup, the highly irregular and variable packet-receive simulation times are observed.

4.3 Cyber-security benchmark results

With the cyber-security benchmark results from the runs using the NSX setup, we verify the correctness of the mobility feature of the MANET scenario. As mentioned in Section 3.3, the worm in the cyber-security benchmark infects only the next-hop nodes. Hence, in the static MANET scenario, it is expected that the worm from the VS (node 61) infects its only neighbor (node 45), from which all other nodes get infected. This exactly is observed in the worm propagation graph, as shown in Figure 14(a).

Similarly, in the mobile MANET scenario, where the mobile node 61 revolves around the periphery of the MANET, it is expected that node 61 infects other peripheral nodes other than node 45. This exactly is observed in Figure 14(b), where node 61 not only infects node 45 but also infects node 59, node 56, node 53, node 52, node 50, and node 62. This verifies the correctness of the mobility feature in the mobile MANET scenario.

In Figure 15, we plot the virtual time of infection across a number of VMs for both static and mobile MANET scenarios, with 95% confidence intervals. This graph also shows the worm propagation trend in static and mobile scenarios. In analyzing the plots, it should be noted that the infection propagates only through one-hop neighbors and the infection propagation itself is in two phases. Further, only after the second phase (after payload transfer) the infected node actively infects others. Hence, if a MANET node has fewer neighbors, the parent node aggressively tries and quickly converts its neighbor into an infected, worm-propagating node. However, if the MANET node is mobile, it might not be able to completely infect its neighbor as it moves along.

In Figure 14(a), we see that the infection starts from node 61, and infects all. The lack of mobility in this scenario helps for a quick rate of initial infection propagation, as seen in Figure 15. In contrast, Figure 14(b) shows that node 61is not able to infect all of its visited neighbors successfully. However, even with lower number of infected nodes (seven out of 16 visited neighbors), the infection rate in the mobile scenario picks up at later stages of the simulation, as shown in Figure 15. Hence, the infection propagation rate is actually dependent on the ability of the mobile node to successfully convert its neighbors to worm propagators.

5. Conclusion and future work

An important benefit of virtual time-synchronized execution of VMs is in its ability to increase the level of determinism and repeatability across multiple test runs of even the most complex distributed algorithms and their actual implementations. We presented a virtual time-synchronized approach to employing VMs for modeling and simulating complex software-controlled net-centric systems, such as MANETs in defense communications, reconciling the challenging constraints of modeling effort, fidelity, scale, and speed. Our prototype realized in the Xen hypervisor is observed to provide excellent repeatability and accuracy, verified via multiple benchmarks. Based on this feasibility, a useful alternative arises in

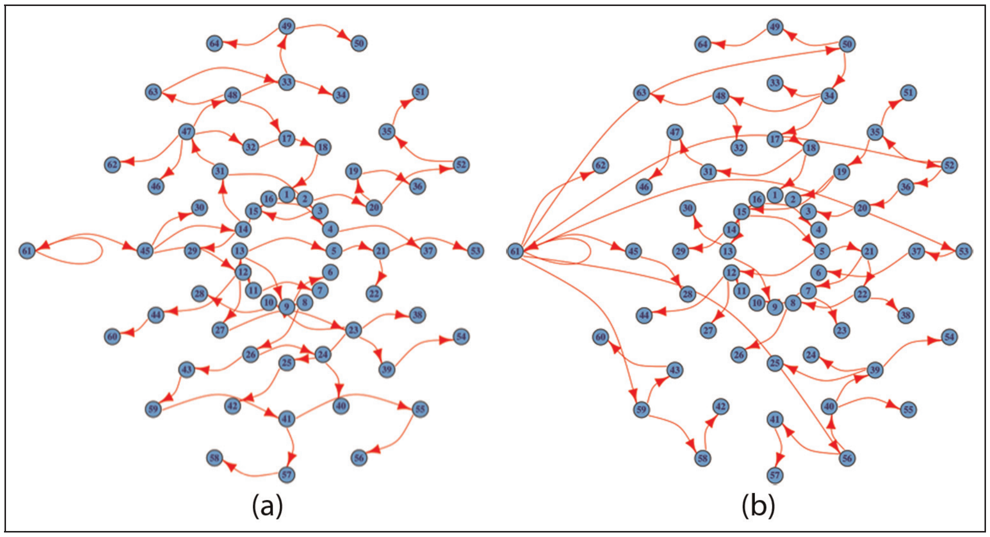

Figure 14. Worm spread from node 61 in static (a) and mobile (b) mobile ad-hoc network (MANET) scenarios.

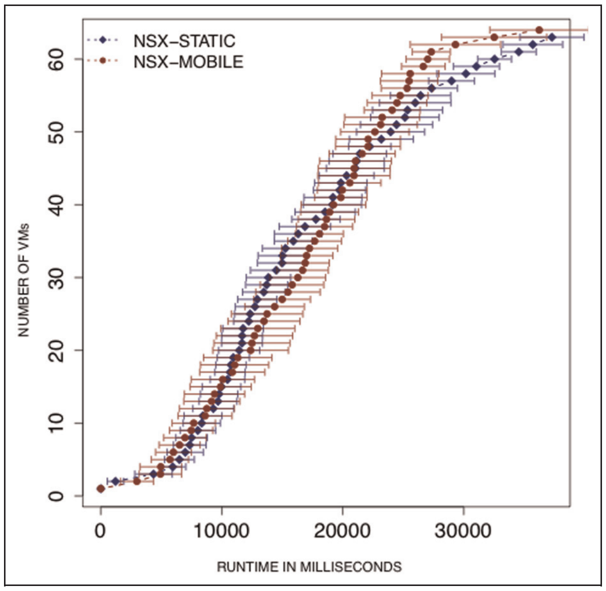

Figure 15. Worm spread dynamics in static and mobile ad-hoc network scenarios.

defense network modeling, namely the use of actual, unmodified software implementations for the dual purposes of simulation-based experiments as well as actual deployment.

While most traditional approaches can be classified as either a simulation or an emulation, the approach presented here is neither alone but in fact is a combination. It retains the desirable features of high fidelity and time accuracy while overcoming limitations in scale, bias, and non-repeatability of free-running emulations based on VMs. Because the virtualized network under NetWarp is synchronized by virtual time rather than real time, NetWarp's components can be time-synchronously executed with external tools, such as signal strength simulators, for accuracy at the physical layer, or for satellite link functionality with complex frame/bit processing state machines. Further, our simple synthetic benchmark, with a theoretically derived analytic solution, and our method of measuring time-order errors serve as a *correctness-gauge* for any VM-based MANET emulator and/or simulator.

Additional work, however, is needed to test the highest limits of temporal resolution at which the scheduling can be realized, although, theoretically, any amount of instrumentation may be inserted with minimal perturbation to the MANET systems being tested. Future work also includes investigation of additional schemes for realizing inter-node connectivity, multi-node synchronization, wireline network integration, and mobile OS support.

Regarding inter-node connectivity, the approach used here is best for scenarios with rapidly changing inter-VM signal reachability. Our scheme based on internet protocol (IP) tables at the receivers provides an efficient, low-overhead mechanism for implementing reachability dynamically. However, for slow-changing or mostly static interconnectivity, it may be possible to improve speed by allocating pair-wise software bridges that minimize broadcast traffic in the virtual network devices of all the VMs.

To further increase the scale of scenarios, multiple nodes would be needed to host a larger number of VMs. In this case, the hypervisor scheduler functionality needs to be augmented with inter-node time synchronization. Additional work is needed to design a new inter-node synchronization algorithm that provides globally virtual time-ordered VM execution by interfacing with multiple instances of the intra-node scheduler (one on each node) described here. Finally, it is conceivable to apply this approach to civilian applications by supporting newer mobile OSs, such as Android, in simulating additional device types and networked behaviors.

Declaration of conflicting interest

The authors declare that there is no conflict of interest.

Funding

This paper has been authored by UT-Battelle, LLC, under contract DE-AC05-00OR22725 with the US Department of Energy. Accordingly, the United States Government retains and the publisher, by accepting the article for publication, acknowledges that the United States Government retains a non-exclusive, paid-up, irrevocable, worldwide license to publish or reproduce the published form of this manuscript, or allow others to do so, for United States Government purposes. This work was partly funded by the US Army Research Laboratory (2133-V510-11).

6. References

1. Yoginath SB, Perumalla KS and Henz BJ. Taming wild horses: the need for virtual time-based scheduling of VMs in network simulations. In: *international symposium on modeling, analysis & simulation of computer and telecommunication systems*, 2012.
2. Chisnall D. *The definitive guide to the Xen hypervisor*. Pearson Education, Westford, Massachusetts, 2007.
3. Fall K. Network emulation in the VINT/NS simulator. In: *IEEE international symposium on computers and communications*, 1999.
4. Xinjie C. Network simulations with OPNET. In *Proceedings of the 31st conference on Winter simulation*, 1999.
5. Riley GF. Large-scale network simulations with GTNetS. In: *winter simulation conference*, 2003.
6. Cowie J, Liu H, Liu J, Nicol D, and Ogielski A. Towards realistic million-node internet simulations. In International Conference on Parallel and Distributed Processing Techniques and Applications. 1999.
7. Zeng X, Bagrodia R and Gerla M. GloMoSim: a library for parallel simulation of large-scale wireless networks. In: *workshop on parallel and distributed simulation*, 1998.
8. Fujimoto RM, Perumalla K, Park A, et al. Large-scale network simulation: how big? How fast? In: *IEEE/ACM international symposium on modeling, analysis and simulation of computer telecommunications systems*, 2003.
9. Rizzo L. Dummynet: a simple approach to the evaluation of network protocols. *ACM SIGCOMM Comput Commun Rev* 1997; 27: 31–41.
10. Huang XW, Sharma R and Keshav S. The ENTRAPID protocol development environment. In: *annual joint conference of the IEEE computer and communications societies*, 1999.
11. Vahdat A, Yokum K, Walsh K, et al. Scalability and accuracy in a large-scale network emulator. *ACM SIGOPS Oper Syst Rev* 2002; 36: 271–284.
12. White B, Lepreau J, Stoller L, et al. An integrated experimental environment for distributed systems and networks. *ACM SIGOPS Oper Syst Rev* 2002; 36: 255–270.
13. Gu Y. *ROSENET: a remote server-based network emulation system*. PhD Dissertation, Georgia Institute of Technology, 2008.
14. Gupta D, Yocum K, McNett M, et al. To infinity and beyond: time-warped network emulation. In: *3rd conference on networked systems design & implementation - volume 3*, USENIX Association, 2006.
15. Apostolopoulos G and Hassapis C. V-eM: a cluster of virtual machines for robust, detailed, and high-performance network emulation. In: *IEEE international symposium on modeling, analysis, and simulation*, 2006.
16. Gupta D, Vishwanath KV and Vahdat A. DieCast: testing distributed systems with an accurate scale model. In: *5th USENIX symposium on networked systems design and implementation*, USENIX Association, 2008.
17. Liu J, Rangaswami R and Zhao M. Model-driven network emulation with virtual time machine. In: *winter simulation conference*, 2010.
18. Bergstrom C, Varadarajan S and Back G. The distributed open network emulator: using relativistic time for distributed scalable simulation. In: *workshop on principles of advanced and distributed simulation*, 2006.
19. Grau A, Maier S, Hermann K, et al. Time jails: a hybrid approach to scalable network emulation. In: *workshop on principles of advanced and distributed simulation*, 2008.
20. Erazo MA and Liu J. Leveraging symbiotic relationship between simulation and emulation for scalable network experimentation. In: *ACM SIGSIM conference on principles of advanced discrete simulation*, New York, 2013, pp.79–90.
21. Yoginath SB and Perumalla KS. Efficiently scheduling multi-core guest virtual machines on multi-core hosts in network simulation. In: *workshop on principles of advanced and distributed simulation*, 2011.
22. Yoginath SB, Perumalla KS and Henz BJ. Runtime performance and virtual network control alternatives in VM-based high-fidelity network simulations. In: *winter simulation conference*, 2012.
23. Weingartner E, Schmidt F, Vom Lehn H, et al. SliceTime: a platform for scalable and accurate network emulation. In: *8th USENIX conference on networked systems design and implementation*, 2011.
24. Zheng Y and Nicol DM. A virtual time system for OpenVZ-based network emulations. In: *IEEE workshop on principles of advanced and distributed simulation (PADS)*, 2011.
25. EMANE User Manual 0.7.3, DRS Cengen, Bridgewater, NJ, 2012.
26. Henz BJ, Parker T, Riche D, et al. Large scale MANET emulations using U.S. Army waveforms with application: VoIP. In: *military communications conference*, 2011.
27. Yoginath SB and Perumalla KS. Efficient parallel discrete event simulation on cloud/virtual machine platforms. *ACM Trans Model Comput Simul* (in Press); 26.

28. Netfilter: Netfilter - Firewalling, NAT and Packet Mangling for LINUX, http://www.netfilter.org (April, 2012).
29. Hoffman D, Prabhakar D and Strooper P. Testing iptables. In: *conference of the centre for advanced studies on collaborative research*, IBM Press, 2003.
30. Clausen T and Jacquet P. Optimized Link State Routing Protocol (OLSR). RFC 3626.
31. OLSRD: Optimized Link State Routing Daemon, www.olsr.org (January, 2012).
32. Gropp W, Lusk E and Skjellum A. *Using MPI: portable parallel programming with the message-passing interface - Vol. 1*. MIT Press, Cambridge, MA, 1999.

Author biographies

Srikanth B Yoginath is a Research Staff Member at the Computational Sciences and Engineering Division, Oak Ridge National Laboratory (ORNL) in Oak Ridge, Tennessee, USA. He holds a PhD in Computational Science and Engineering (2014, Georgia Institute of Technology).

Kalyan S Perumalla is a Distinguished Research and Development Staff Member and Manager at the ORNL in Oak Ridge, Tennessee, USA. He is the founding group leader of the Discrete Computing Systems Group in the Computational Sciences and Engineering Division at ORNL. He also serves as an adjunct processor in the School of Computational Science and Engineering at the Georgia Institute of Technology, Atlanta, Georgia, USA, and was a Fellow of the Institute of Advanced Study, Durham University, UK.

Brian J Henz is a Research Team Lead in the Computational and Informational Sciences Directorate at the US Army Research Laboratory, Aberdeen Proving Ground, Maryland, USA. He holds a PhD in Mechanical Engineering (2009, University of Maryland, College Park).