Performance of Virtual Machines Under Networked Denial of Service Attacks: Experiments and Analysis

Ryan Shea, *Student Member, IEEE,* and Jiangchuan Liu, *Senior Member, IEEE*

***Abstract*—The use of virtual machines (VMs) to provide computational infrastructure and services to organizations is increasingly prevalent in the modern IT industry. The growing use of this technology has been driven by a desire to increase utilization of resources through server consolidation. Virtualization has also made the dream of such utility computing platforms as cloud computing a reality. Today, virtualization technologies can be found in almost every data center. However, it remains unknown whether the VMs are more vulnerable on external malicious attacks. If so, to what extent their performance degrades, and which virtualization technique has the closest to native performance? To this end, we devised a representative set of experiments to examine the performance of most typical virtualization techniques under typical denial-of-service (DoS) attacks. We show that, on a DoS attack, the performance of a web server hosted in a VM can degrade by up to 23%, while that of a nonvirtualized server hosted on the same hardware degrades by only 8%. Even with relatively light attacks, the file system and memory access performance of hypervisor-based virtualization degrades at a much higher rate than their nonvirtualized counterparts. We further examine the root causes of such degradation and our results shed new lights in enhancing the robustness and security of modern virtualization systems.**

***Index Terms*—Cloud security, denial of service, virtualization.**

I. Introduction

IN THE PAST decade, virtualization of computers has gone from little more than an interesting research topic to a nearly ubiquitous technology. A recent survey showed that 90% of organizations use virtual machines (VMs) in some capacity in their IT infrastructures [1]. As of 2011, 34% of organizations use virtualization to meet the majority of their server needs [2]. This uptake has been happening at a staggering rate considering that, prior to 2006, less than 5% of organizations had deployed virtualization on their systems [2]. This rapid uptake owes itself to promises of increased utilization of resources, increased server consolidation, and decreased costs [3]. Virtualization is also one of the keystone technologies that make such utility computing platforms as cloud computing possible. A prominent example is Amazon's EC2, the current market leader in cloud computing, which makes heavy use of the open-source Xen Virtualization system.

As with any new technology that has experienced such a dramatic expansion, there are a number of new challenges to be addressed, among which security and privacy have long been a focus. There has been significant research in this field, particularly on data leakage between running VMs and, in the case of public utility computing platforms such as cloud computing, the data leakage between a guest and the host itself. Yet, the external attacks that attempt to directly target the VMs have seldom been examined. It remains unclear whether virtualization is resistant to such attacks, or is even more vulnerable than conventional physical machines.

In this paper, we present a study on the performance of modern virtualization solutions under networked denial of service (DoS) attacks. We devise a representative set of experiments to examine the performance of most typical virtualization techniques under standard TCP-based distributed denial of service (DDoS) attacks. We also compare them with the same DDoS on the same services running on nonvirtualized servers. Our experiments cover a full spectrum of virtualization solutions with state-of-the-art implementations, and we also examine a comprehensive set of benchmarks. With these, we attempt to answer the following two critical questions.

1) Are virtual machines more vulnerable under DoS attacks? If so, to what degree does their performance degrade?
2) Which virtualization technique has the closest to native performance and, under what condition?

The answers to these questions will have significant implication to the deployment of virtualization, as any virtual server will be facing persistent threats from DoS attacks. The answers from our experiments however are not very optimistic. We show that, upon a DoS attack, a web application server hosted in a VM can degrade by up to 23%, while a nonvirtualized server hosted on the same hardware degrades by only 8%. Even with relatively light attacks, the file system and memory access performance of hypervisor-based virtualization degrades at a much higher rate than their nonvirtualized counterparts. These observations suggest that the state-of-the-art virtualization solutions need to be substantially revisited

Manuscript received October 9, 2011; revised March 28, 2012; accepted June 6, 2012. Date of publication November 30, 2012; date of current version April 17, 2013. Some experiments were first presented at IWQoS'12 and have been extended and updated for this publication.

The authors are with Simon Fraser University, Burnaby, BC V5A 1S6, Canada (e-mail: rws1@cs.sfu.ca; jcliu@cs.sfu.ca).

Digital Object Identifier 10.1109/JSYST.2012.2221998

in this perspective. We further examine the root causes of the performance degradations, shedding lights in enhancing the robustness and security of modern virtualization systems.

II. Overview of Virtualization

To properly analyze and compare virtualization techniques under DoS attacks, we need to select representative samples of virtualization packages, so as to cover the typical and state-of-the-art solutions. Broadly speaking, all current virtualization solutions can be classified into three main categories, which we discuss as follows.

A. Paravirtualization (PVM)

Paravirtualization was one of the first adopted versions of virtualization and is still widely deployed today. PVM requires no special hardware to realize virtualization, instead relying on special kernels and drivers that are aware they are being virtualized. The kernel inside a guest machine running on a PVM host will send privileged system calls and hardware access directly to a hypervisor, which in turn decides what to do with the request. The use of special kernels and drivers means a loss of some flexibility in terms of choice of operating systems. In particular, a user of a PVM-based virtualization solution must use an operating system that can be modified to work with the hypervisor. Although this does not present a significant problem for open-source operating systems such as Linux, it does create a problem for those wishing to use a proprietary OS such as Microsoft Windows. Paravirtualization does offer some advantages, such as reduced overhead to virtualize privileged operating systems calls, as special hardware is not needed to intercept them. Typical paravirtualization solutions include Xen [4] and User Mode Linux [5].

B. Hardware Virtual Machine

Hardware virtual machine (HVM) is the lowest level of virtualization, which requires special hardware capabilities to trap privileged calls from guest domains. It allows a machine to be fully virtualized without the need for any special operating systems or drivers on the guest system. The guest simply interacts with hardware drivers unaware that it is running in a VM and actually communicating with an emulated interface. Most modern CPUs are built with HVM capabilities, often called virtualization extensions. AMD and Intel both support HVM, under the name of AMD-V and VT-X, respectively. They detect when a guest VM tries to make a privileged call to a system resource such as sending on the network interface card. The hardware intercepts this call and sends it to a hypervisor which decides how to handle the call. This creates great flexibility for the guest since practically any OS can be run in these VMs, as it is not required that the guest be aware it is a VM. It has been noticed however that HVMs can also have the highest Virtualization overhead and as such may not always be the best choice for a particular situation [6], [7]. It is important to note that work has been done on Paravirtualization drivers for input/output (I/O) devices like the Network Interface Card. These drivers are aware of the fact that they are running in a virtualized system and can greatly alleviate the overhead introduced by HVMs. One such example of a paravirtualization driver package is the open-source VirtIO [8]. Representative Virtualization solutions that are HVM include VMware Server [9], KVM [10], and VirtualBox [11].

C. Container Virtualization

Container Virtualization, also know as OS-level virtualization, creates multiple secure containers to run different applications in. It is based on the intuition that a server administrator may wish to isolate different applications for security or performance reasons while maintaining the same OS across each container. Container virtualization allows a user to share a single kernel between multiple containers and have them securely use computer resources with minimal interference from others containers. It has been shown to have the lowest overhead among all the existing virtualization techniques [6]. This superiority however comes at the price of much less flexibility as compared to other solutions. In short, the user cannot mix different operating systems, e.g., a Debian Squeeze and an Open Solaris. Typical container virtualization implementations include OpenVZ [12], Linux-VServer [13], and Solaris Zones [14].

It is important to note that Hardware Virtualization and Paravirtualization both use a *Hypervisor* to interact with the underlying hardware whereas Container Virtualization does not. This distinction is crucial because the use of the hypervisor generally improves performance isolation between guests on a host by acting as a gatekeeper to the underlying hardware. However, it has been noted that the hypervisor can also introduce measurable overhead [15].

In our experiments, we chose Xen, KVM, and OpenVZ to be evaluated under DoS attacks. The choice was motivated by two key facts. First, they are all open source with publicly available documents and with cross-platform implementations. We can run their packages with the same platform without changing OS or computer hardware. This makes a fair comparison possible and the results reproducible. Second, all of them have been widely used in real-world production environments for server consolidation and cloud computing. As mentioned previously, Xen has been used heavily to provide cloud computing functionality, for example in Amazon EC2; KVM has been used by Ubuntu Enterprise Cloud [16] and Eucalyptus Cloud Service [17]; OpenVZ is a popular choice in offering virtual private server containers to the public.

III. Overview of Denial of Service

DoS attacks are attempts by a nonlegitimate user to degrade or deny resources to legitimate users. There are many different forms a DoS attack can take. In this paper, we will focus on networked DoS, the most common threat against modern IT infrastructure. In particular, we examine the TCP SYN flood attack against a target machine, which is one of the most common attacks on the Internet today and is notoriously hard to filter out before it reaches the end system. In this section, we first give a brief discussion on the TCP SYN flood attack and its potential threat to virtual machines.

A. *Transmission Control Protocol SYN Flood*

The transmission control protocol (TCP) is one of the foundations of the global Internet. It provides users with a powerful protocol to send data between two computers with many service guarantees. TCP provides reliable in-order delivery of whatever data the users wish to send [18]. When TCP was initially developed, the Internet remained a small private collection of computers and security issues inherent in the protocol were of little concern. As such some features of TCP can be exploited to perform DoS attacks. Specifically, many TCP-based DoS attacks take advantage of the TCP three-way handshake which we will describe next.

The TCP SYN flood is one of the simplest and most common attacks seen on the Internet. This attack takes advantage of the amount of resources that have to be allocated by a server in order to perform a three-way handshake. An attacker tries to overload a victim with so many connection requests that they will not be able to respond to legitimate requests. To perform this attack, the attacker initiates many connections to the victim's system using TCP SYN packets. The victim allocates buffers for the new TCP connection and transmits a SYN-ACK in response to the connection request. The attacker has no intention of opening a connection, so it does not respond to the SYN-ACK. If this attack uses up enough of the server resources, any connection requests from legitimate users will be rejected [19]. Flooding-based attacks can also exhaust other resources of the system such as CPU time.

B. *TCP DDoS Mitigation Strategy*

Many defenses have been proposed to combat the TCP SYN flood. The simplest is to use a firewall to limit the number of TCP SYN packets allowed from a single source. However, many attacks use multiple hosts or employ address spoofing. The case where multiple hosts are involved in an attack is often called a distributed denial of service attack (DDoS). More complex solutions have met with a better level of success and are usually deployed either on the end host or on the network. Network-based solutions include firewall proxies, which only forward the connection request after the client side ACK is received [20]. There has also been research done into filtering based on packet inspection. One effective solution is Hop Count filtering, which inspects the packets TTL field and drops suspected spoofed packets. It has been reported that this technique can achieve up to 90% detection rate [21], [22]. End point solutions include SYN cookies and SYN caches, both of which have been widely deployed. SYN caches work by allocating the minimum amount of data required when a SYN packet arrives, only allocating full state when the Client's ACK arrives [20]. SYN cookies allocate no state at all until the client's ACK arrives. To do this, the connection's states are encoded into the TCP SYN-ACK packet's sequence number. On receipt of the ACK, the state can be recreated based on the ACK's header information [23].

Since virtual machines interact with the network through their virtual interfaces in much the same way that physical machines interact with the network, many of the considerations and defenses for DDoS attacks mentioned above apply to virtualized systems. However, it is well known that current hypervisor-based virtualization can experience high overhead while using their I/O devices such as the network interface. Since DoS attacks attempt to exhaust resources on a targeted server, the stresses on the network interface would amplify the virtualization overhead and thus become even more effective at degrading the target. This will be demonstrated by our experimental results, even though such preventive strategies as SYN cookies and caches have been enabled in our experiments.

IV. Related Works

In the recent literature, there have been many performance analyses performed on different applications and scenarios in virtualized systems. In 2007, researchers from the University of Michigan and HP performed a performance evaluation comparing different virtualization techniques for use in server consolidation [6]. They compared Xen, a hypervisor-based paravirtualization technique, and OpenVZ, a container-based virtualization technique. The results showed that OpenVZ had better performance and lower overhead then Xen. Xen also suffered from much longer response to HTTP requests when under load in their benchmarks.

Soltesz *et al.* [15] performed a comparison between Xen and Linux VServer in terms of performance and architectural design. Researchers at Clarkson University tested HVM, PVM, and container virtualization for performance isolation [24]. They found that HVM has better performance isolation, followed closely by PVM, and that container-based solutions provide the least isolation.

Recently, Ostermann *et al.* [25] performed a performance analysis on Amazon EC2 to determine its suitability for high-performance scientific computing. They found that the use of virtualization can impose significant performance penalties on many scientific computing applications. The impact of virtualization on network performance in Amazon EC2 was evaluated in [26]. It showed that, due to virtualization, users often experience bandwidth and delay instability.

In a recent issue of IEEE Network we explored the overhead created by network interface virtualization [27]. We found that even with hardware pass-through devices such as single-root I/O virtualization (SR-IOV) the overhead of virtualized interfaces remains several times higher than their nonvirtualized counterparts.

There have also been recent works on virtual machines performance hosted in the cloud computing context [28]–[30].

Despite these pioneer works on quantifying the overhead and performance of virtualization under various environments, to our knowledge, the performance of virtualization under networked DoS attacks remains largely unexplored.

V. Experimental Architecture

To evaluate each virtualization technique, we created a small scale yet representative test network and system in our lab. We

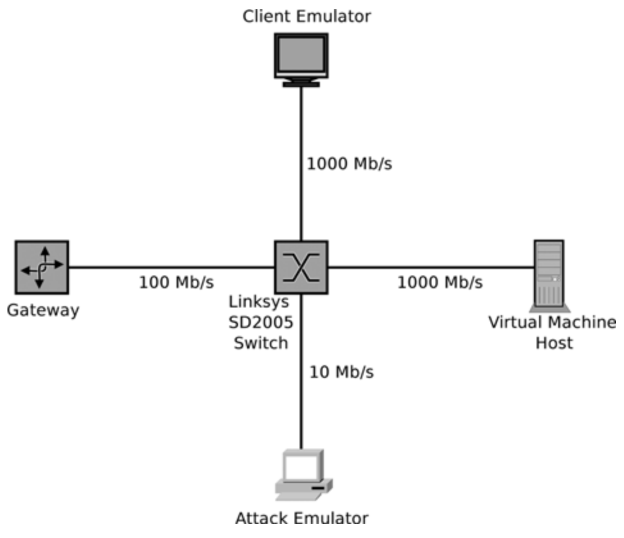


Fig. 1. Network setup.

now give a detailed description of the hardware and software used in our tests.

A. Physical Hardware and Operating System

We used a modern mid-range PC with an Intel Core 2 Q9500 quad core processor running at 2.83 GHz. We enabled Intel VT-X in the bios as it is required for HVMs support. The PC was equipped with 4 GB of 1333 MHz DDR-3 SDRAM and a 320 GB 7200 RPM hard drive with 16 MB cache. The network interface is a 1000 Mb/s Broadcom Ethernet adapter attached to the PCI-E bus.

The host and the guests all used Debian Squeeze as their operating system. The kernel version remained constant at 2.6.35-5-amd64 across all tests. Since Xen and OpenVZ require special kernel patches, we used 2.6.35-5-Xen and 2.6.35-5-OpenVZ for those tests. In all tests, we use the amd64 version of the kernel and packages.

B. Network Setup

To emulate a DDoS attack against our servers, we employed the network configuration as shown in Fig. 1. All machines on the network are attached directly to a Linksys 1000 Mb/s SOHO switch. The attack emulator has been configured to have a link rate of only 10 Mb/s. The client emulator used in our experiments was a dual core Pentium D PC, which created clients for our comprehensive benchmarking. The gateway is the default route for any host outside this directly connected subnet. When we simulate an attack, the gateway is configured to drop forwarded packets. Dropping these packets makes it appear as though they have been forwarded to an external network from the perspective of the virtual machine host. If the gateways were not present, many gratuitous ARP requests would have been created by the virtual machine host as it searched for a route to deliver the packets to the external network.

C. Emulating a Denial of Service Attack

The DDoS attack simulated for our experiments is the standard TCP SYN flood. Our choice of this attack was motivated by the fact that it is one of the most common DoS attacks seen on the Internet today. It is also notoriously hard to be filtered out from legitimate traffic. In our experiments, we assume that a 100 Mb/s distributed SYN flood is being performed on our network and we have successfully filtered out 90% of the attack. A total of 10 Mb/s of attack TCP SYN traffic has bypassed the detection and reached the end host. We believe this is a reasonable setting as no existing solutions have been shown to effectively filter out all attack traffic without greatly affecting legitimate clients [21]. On the end host, we have enabled the SYN cookies defense and, by default, the Linux Kernels use the SYN Cache defense.

To generate the actual attack, we used the open-source hping3 [31] tool. The tool allows us to create arbitrary packets with which to flood a target host. We set hping3 to create TCP SYN packets and randomly selected the source address. We targeted an open port on the target machine. In the case of our synthetic benchmark, it is the secure shell (SSH) port 22; in our comprehensive benchmarks, it is Apache running on port 80. The DDoS traffic originates from our attack emulator, which can be seen in Fig. 1.

D. Virtualization Setup

As explained earlier, we have chosen Xen, OpenVZ, and KVM in our experiments, for their open-source nature and their extensive deployment in the real world. In our experiments we create only a single VM on our host. Although the use of a single VM is not common in practice, the use of this setup allows us to investigate the best case scenario, where a VM has access to 100% of the systems resources. In a multiple VM environment the effect of a DoS attack is likely to be more severe, since the VM has access to less of the host's system resources.

1) *Xen System Setup:* We installed the Xen 4.0 Paravirtualization Hypervisor on our test system. To configure networking we created a bridged adapter and attached our primary interface and Xen's virtual interfaces to it. Xen virtual machines received an IP address from the DHCP running on our gateway. For disk interface, we used Xen's LVM features. We set the number of virtual CPUs (VCPU) to four and the amount of RAM to 2048 MB.

2) *OpenVZ System Setup:* We installed the OpenVZ container-based virtualization package from the Debian repository. We configured our container using the Debian Squeeze template. The container was given access to 2048 MB of main memory and complete access to the four CPU processing cores. Like the Xen setup, network access was provided to the container by bridging the container virtual Ethernet interface to our physical interface.

3) *KVM System Setup:* We used kernel-based virtual machine (KVM) version 0.12.5 from the Debian Squeeze repository. Once again the virtual machine was given access to all four processing cores as well 2048 MB of memory. The disk interface was configured as a flat file on the physical host's file system. Networking was configured once again as a bridge between the virtual machine's interface and the system's physical NIC. To enable the best network performance, we configured KVM to use the VirtIO network drivers [8].

4) *Nonvirtualized Vanilla System Setup:* Finally, as the baseline for comparison, we had a Vanilla setup with no virtualization running, i.e., all direct access to the hardware. The same drivers, packages, and kernel were used as in the previous setup. This configuration enabled us to obtain the minimal amount of performance degradation that our system can experience.

VI. Benchmark Setup

We have chosen a comprehensive set of synthetic benchmarks and a single comprehensive benchmark to evaluate the impact of DDoS attacks on different components of our test machines. Our goal is to test the CPU, network, memory, and file system performance under normal and attack conditions. We now describe the benchmarks chosen to test these system components.

A. CPU Benchmark

We chose the 7-Zip benchmark [32] and the SysBench [33] CPU test to measure the CPU performance because they are both well known and regularly used for gauging raw CPU performance.

7-Zip is an open-source LZMA-based compression application. Among its various features is a multithreaded benchmarking mode, which calculates a system's performance in millions of instructions per second (MIPS). The benchmark compresses and decompresses a set of data and measures the performance. The 7-zip benchmark is not configurable, it simply runs with as many threads as processing cores and outputs the number of MIPS. We recorded the number of MIPS achieved in the compression portion of the benchmark. The 7-Zip test, although CPU bound, is also a moderate user of memory for its benchmarking. For this reason we also selected SysBench since it is more isolated to the CPU.

The SysBench CPU test runs a prime number calculator. It continues to calculate primes until a threshold chosen by the user is reached and the results are presented as the total time to calculate the primes. We chose to calculate the primes in the first 50 000 integers and assigned four threads, so that each of our four cores would be involved in the benchmark. We then recorded the total amount of time taken to calculate the primes.

B. Memory Benchmark

For memory benchmarking, we chose the SysBench memory bandwidth test. The benchmark allocates a segment of either global or thread local memory and performs either read or write operations on it. It outputs the total time taken as well as the memory bandwidth. In our experiments, we assign a single thread to perform 10 GB of writes to main memory.

C. Network Benchmark

To test network performance, we use two tests, namely, iperf [34] and tbench [35]. Iperf is a simple application that attempts to find the maximum TCP or UDP bandwidth between two network hosts. To do this we specified iperf to run in server mode on one host and use it as a client on the other host. In our experiments, the client emulator in our network was chosen as the server and TCP was used as the protocol. Each test was run for 25 s, and the results are output in Mb/s.

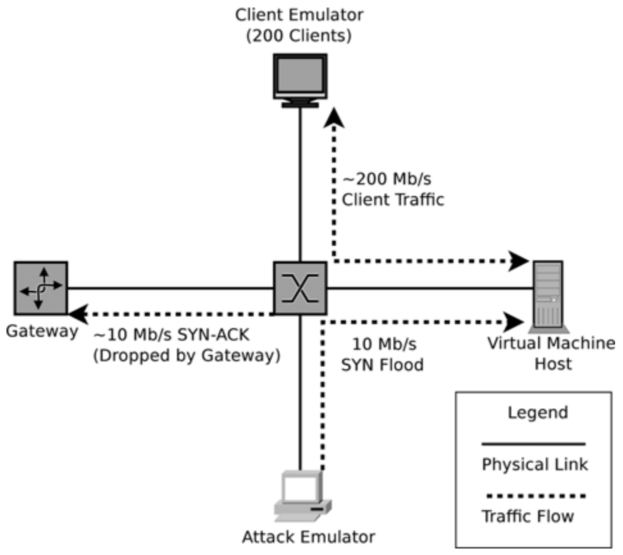


Fig. 2. Comprehensive benchmark setup.

Our second benchmark is tbench, which emulates the network portion of the standard Netbench performance test. Tbench sends TCP data based on a workload profile which simulates a network file system. We specified our client emulator as the tbench server and the virtual machine as a client, and then ran the work load for 50 s to find the throughput. The results are given in MB/s.

D. File System Benchmark

The file system performance was tested using the SysBench fileio test. This benchmark creates a specified set of files on the disk and performs read and write operations on them. For our experiments, we created 2 GB worth of files and a single thread performed random read and write operations on the files. The results are given as MB/s.

E. Comprehensive Benchmark—Web Application

To further understand the overall system performance, we have devised a comprehensive benchmark based on a simple 2-tier Web Server and Database. We used the Debian repositories to install the Apache 2.2 Web Server and the SQL database MySQL Server 5.1. To create a web application representative of a real-world service, we installed the RuBBoS [36] bulletin board benchmark. We chose the PHP version of the RuBBoS and installed the necessary Apache extensions for PHP. We stored the RuBBoS data into our MySQL database. RuBBoS was then installed in each of our test configurations.

Although RuBBoS comes with its own client simulator, we chose to use the Apache benchmark instead. This is because it has been commonly used for web server stress testing. Although the RuBBoS simulator can perform tests specific to RuBBoS, we only require maximum request rate and connection timing, which are much more straight forward to extract with the Apache Benchmark.

TABLE I
CPU Usage While Under Attack: System Idle

	TCP DDoS		UDP DoS	
System	10 Mb/s	100 Mb/s	10 Mb/s	100 Mb/s
KVM	˜102% (˜75%)[1]	˜205% (˜213%)[1]	˜64%	˜272%
Xen	˜98% (˜60%)[1]	˜187% (˜170%)[1]	˜46%	˜145%
OpenVZ	˜8% (˜6%)[1]	˜100% (˜30%)[1]	˜1%	˜1%
Vanilla	˜6% (˜4%)[1]	˜90% (˜25%)[1]	˜1%	˜1%

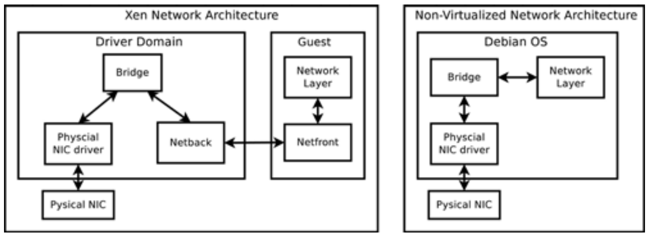


Fig. 3. Simplified network architecture.

We ran the Apache Benchmark against the RuBBoS website in each of the test setups. We simulated 200 clients requesting the latest forum topics page. By using this page, the web server must perform a single SQL query and render the PHP page for each user request. We then used the Apache benchmark to calculate how long it takes to service 500 000 requests. Fig. 2 shows the network configuration and the traffic flows during this experiment.

VII. CPU Usage During DoS

We first measure the impact of DoS traffic on CPU usage while the system idle. To this end we started the system and killed all nonoperating system processes. To provide a target for the TCP DDoS we ran a SSH server and configured it to listen to port 22. For KVM, OpenVZ, and Vanilla, the measurements were performed using the Linux top command. Since Xen is not compatible with the standard Linux top command, we used the xentop command to measure the CPU usage. We ran each system under both 10 and 100 Mb/s TCP SYN floods. To illustrate the effect enabling SYN cookies have on our systems, we provide CPU usage for our systems when SYN cookies are enabled as well as when they are disabled. We also included the CPU usage from a 10 and 100 Mb/s UDP flood. The results are given in Table I.

Under a 10 Mb/s TCP DDoS, the hypervisors in both KVM and Xen consume the CPU time of an entire core simply delivering SYN packets to the VM and returning SYN-ACKs to the network. OpenVZ and Vanilla, on the other hand, use only between 6% and 8% of CPU time on a core to service the same attack. If we increase the attack rate to 100 Mb/s, all systems increase their CPU usage; however, both Xen and KVM consume nearly half of the systems total CPU resources simply processing the attack traffic. As we increase the attack traffic rate, the corresponding increase in CPU usage indicates that the systems will continue to degrade as they are exposed to higher attack rates. As can be seen, enabling SYN cookies increases the CPU usage in each of our systems. This is an expected observation, since SYN cookies ensure each incoming SYN packet generates a SYN-ACK, which is returned to the network. This increase in outgoing packets manifests itself in the increased CPU utilization experienced by all systems. The deployment of SYN cookies ensures a server can still establish incoming TCP connections even while under attack by a TCP SYN flood. Without SYN cookies enabled our test systems would quickly exhaust their connection queues and connections to our SSH server would be impossible.

TABLE II
Xen: CPU Usage of Different Networking Modes While Under Attack

	Bridged	Routed	NAT
Xen	˜98%	˜105%	˜125%

Though our focus is on TCP SYN attack, we have also devised a UDP DoS to determine if TCP was the culprit for the massive CPU usage experienced by the virtual machines. To create the UDP flood, we once again used hping3 with 10 Mb/s and 100 Mb/s of UDP traffic target the system. We did not randomize the source address and the packets contained no data. As can be seen, the high CPU usage is present for both KVM and Xen in the UDP case as well. This tells us that high CPU phenomena is experienced whenever the hypervisor experiences a data stream that contains small packets at a high rate. We also tested ICMP and plain IP packets and found that any small packet sent at a high rate reaching the end system leads to this phenomena.

To ensure the bridge interface used in our Xen and KVM systems was not the cause of the CPU usage, we configured our Xen system to use each of its networking modes and ran each configuration under a 10 Mb/s random source SYN Flood. The Xen networking modes include Bridging, Routing, and network address translation (NAT). Table II shows that our choice of a bridged interface has the lowest CPU usage at 98%, followed by routed at 105% and NAT at 125%. It is unsurprising that routing and bridging are quite similar in terms of performance, since they work on a similar principle, essentially forwarding the packet unmodified to the virtual machine. NAT on the other hand is in a strikingly different situation, where each packet must have a network layer header replacement before it is sent or received by the VM.

Our data show that the CPU must expend much more resources processing small packets in hypervisor-based virtual machines, such as KVM and Xen, than in Vanilla or OpenVZ. Much of this extra CPU usage comes from the many extra operations that must be performed to deliver a packet to a virtualized system. Fig. 3 shows a simplified drawing of the steps involved in delivering packets to a Xen VM, as well as to our Vanilla host. On receipt of a packet in Xen, the physical NIC driver delivers the packet to the bridge; Once the packet arrives at the bridge it is transferred to the Netback module, which allocates buffers for the packet and notifies Netfront, in the Guest VM, of the incoming packet; Finally, Netfront receives the packet and passes it the guests' network layer.

[1]TCP SYN cookies disabled.

Our Vanilla host on the other hand collects the packet from the physical NIC and delivers it to the bridge; the bridge then passes the packet directly to the network layer. It is clear that the many extra steps involved in delivering packets to a VM must contribute to the CPU usage we have discovered.

In the next section, we will perform benchmarks on our systems under a 10 Mb/s TCP DDoS to quantify the performance degradation experienced by the systems.

VIII. Benchmark Results under DoS

For each test, we ran each benchmark four times and calculated the mean. To ensure our results are statistically significant, we also calculated the standard deviation for each measurement and express it as percentage of the mean. We calculate the performance degradation experienced by the system while SYN cookies are both enabled and disabled. Performance degradation is quantified as the percentage change from the base line benchmark performance to the attack performance.

A. Result CPU Benchmark

The 7-Zip benchmark results are given in Table III, where higher number of MIPS implies higher performance. The SysBench Prime benchmark is given in Table IV and lower completion time is better in this case. We can see that in the baseline case all virtualization techniques perform within 5% of each other on both tests. It is interesting to note that our OpenVZ and Vanilla systems appear to give near identical performance, as in both tests they are within a standard deviation of each other.

However, we can see that even a relatively small 10 Mb/s DDoS has a significant effect on the CPU performance of both KVM and Xen. When SYN cookies are disabled the 7Zip results show both KVM and Xen degrade by 16%. When SYN cookies are enabled the degradation increases dramatically to over 24% for both KVM and Xen. The Sysbench prime number benchmark show similar results for KVM and Xen. With KVM suffering from 16% less performance while SYN cookies are disabled and 31% while enabled. Xen loses over 15% performance when SYN cookies are disabled and 30% while enabled. This loss is due to the amount of time the Xen/KVM hypervisor spends on the CPU servicing the attack packets. OpenVZ and nonvirtualized Vanilla host fared much better, both with a smaller but still measurable amount of performance degradation. It is worth noting that all systems consume more CPU resources when SYN cookies are enabled, which is unsurprising as each incoming SYN packet is generating a corresponding SYN-ACK packet. The added CPU consumption comes both from generating the SYN-ACK packet using the SYN-Cookie algorithm, as well as sending the packet on the network.

B. Result Memory Benchmark

The memory benchmark results shown in Table V provided some intriguing results as there is a wide variation in the base line performance. In this particular benchmark, Xen fared by far the worst, being almost 10× slower than our Vanilla system setup. KVM fared much better then Xen but still managed only about half of the memory bandwidth of Vanilla or OpenVZ.

Under the TCP DDoS, all setups showed a measurable slowdown in performance, with Vanilla, OpenVZ, and Xen having all below 8% performance degradation with both SYN cookies enabled and disabled. KVM on the other hand experienced a slowdown of 13% when SYN cookies were disabled and over 17% when SYN cookies are enabled. In KVM the, hypervisor must map a memory access request from the guests memory address to the corresponding physical address on the host machine. We conjecture that the KVM hypervisor is busy servicing I/O request created by the DDoS packets. With this, the memory requests must wait longer to be mapped to the correct physical address. This delay manifests itself in the large performance degradation experienced by KVM in this test.

C. Result Network Benchmark

The Iperf results are given in Table VI and the tbench results are given in Table VII. The Iperf benchmark measures maximum transfer rate between two hosts. The maximum transfer rate base line performance is comparable for the Xen, OpenVZ, and Vanilla systems. KVM however has over 17% lower performance in this bandwidth test. The tbench tested a more complex set of TCP connections between client and server. This benchmark also provided some interesting results as OpenVZ and Vanilla performed with near identical performance. Xen was noticeably slower suffering from 13% lower performance than Vanilla and OpenVZ. KVM however has abysmal performance in this test being over 60% slower then OpenVZ and Vanilla.

Under TCP DDoS attack, the IPerf results showed only a slight drop in the throughput for our OpenVZ and Vanilla systems. With SYN cookies enabled Xen lost nearly 5% of its throughput. KVM however lost nearly 16% of its throughput with SYN cookies disabled but interestingly less than 11% when the SYN cookies are enabled. We conjecture the reason for this surprising result is due to an optimization found in KVM VirtIO drivers. The VirtIO drivers can reduce the virtualization overhead of sending packets by bundling several packets into a single delivery to the hypervisor. The packet bundle is either delivered to the hypervisor when a timer expires or a threshold based on the number of packets is reached. Since SYN cookies creates a SYN-ACK packet for every SYN packet generated by our DoS, it is likely the number of packets threshold is reached sooner, creating a slight increase in performance when SYN cookies are enabled.

Tbench also provided some interesting results. OpenVZ and Vanilla experienced only a small amount of performance degradation when under attack. Xen however lost a measurable 7% with SYN cookies disabled and 14% when enabled, likely due to the CPU time consumed by the hypervisor while servicing the DoS packets. KVM once again lost the most performance, with nearly 31% with SYN cookies disabled and 33% when enabled. For applications that have a similar profile to tbench, it is clear that KVM should be avoided, as it is an extremely poor performer.

TABLE III
7-ZIP BENCHMARK RESULT

	Base		Attack			Attack SYN cookies		
System	MIPS	ST-DEV (%)	MIPS	ST-DEV (%)	Performance Degradation (%)	MIPS	ST-DEV (%)	Performance Degradation (%)
KVM	8360	1.60	7004	1.72	16.22	6346	0.71	24.09
Xen	8445	3.70	7090	3.44	16.05	6417	2.29	24.02
OpenVZ	8690	2.46	8007	1.55	7.87	7525	1.74	13.40
Vanilla	8607	0.99	8018	0.59	6.85	7807	0.54	9.29

TABLE IV
SYSBENCH CPU BENCHMARK RESULT

	Base		Attack			Attack SYN cookies		
System	Sec.	ST-DEV (%)	Sec.	ST-DEV (%)	Performance Degradation (%)	Sec.	ST-DEV (%)	Performance Degradation (%)
KVM	19.91	0.03	23.17	0.28	16.36	26.11	0.11	31.12
Xen	19.92	0.21	22.93	0.40	15.06	26.00	0.31	30.47
OpenVZ	19.84	0.04	20.55	0.04	3.59	21.34	0.07	7.57
Vanilla	19.82	0.01	20.35	0.09	2.63	20.83	0.15	5.07

TABLE V
SYSBENCH MEMORY BENCHMARK RESULT

	Base		Attack			Attack SYN cookies		
System	MB/s	ST-DEV (%)	MB/s	ST-DEV (%)	Performance Degradation (%)	MB/s	ST-DEV (%)	Performance Degradation (%)
KVM	1056	0.45	914	1.94	13.48	870	0.73	17.66
Xen	234	0.68	226	2.17	3.20	220	1.79	6.11
OpenVZ	2326	0.27	2233	0.34	3.99	2143	0.19	7.85
Vanilla	2325	0.13	2265	0.48	2.60	2190	0.40	5.82

TABLE VI
IPERF NETWORK BENCHMARK RESULT

	Base		Attack			Attack SYN cookies		
System	Mb/s	ST-DEV (%)	Mb/s	ST-DEV (%)	Performance Degradation (%)	Mb/s	ST-DEV (%)	Performance Degradation (%)
KVM	535	1.62	450	3.68	15.93	478	1.60	10.63
Xen	656	0.12	652	1.31	0.65	625	2.41	4.80
OpenVZ	649	1.05	635	2.51	2.12	633	3.48	2.50
Vanilla	656	0.19	637	2.31	2.82	636	1.88	2.94

D. Results I/O Benchmark

Our final synthetic benchmark result is for the SysBench I/O test, given in Table VIII. Although there is a significant difference in base line performances in this test, it is hard to make a direct comparison, due to the nature of disk benchmarking. For example, Xen showed significantly faster performance than the others tested, however its LVM volume was allocated near the outside track of the physical disc. With current hard drive design, being near the outside of the disc can be significantly faster than being near the inside. For these reasons, we will refrain from comparing system's baseline performance on this benchmark and instead focus on the performance loss during a DoS.

Under the DoS conditions, OpenVZ and Vanilla systems suffered no significant performance loss. On the other hand, both KVM and Xen lose considerable performance. KVM loses the most performance, with SYN cookies disabled 26%, and a staggering 29% when SYN cookies are enabled. Xen also loses considerable performance with nearly 22% lower-random access to its file system while SYN cookies are disabled, and 25% when SYN cookies are enabled. We believe that Xen and KVM's performance loss is likely due to the hypervisor delaying disk access and instead favoring to deliver attack packets to the virtual machine. We also tested Xen performance when using a flat file for its file system instead of LVM. With SYN cookies enabled this system suffered from nearly 19% lower random access to its file-system.

IX. COMPREHENSIVE BENCHMARK RESULTS: WEB APPLICATION

From our synthetic benchmarks, it is clear that hypervisor-based virtualization techniques are more susceptible to performance degradation under DoS attacks. Although synthetic benchmarks are excellent at pinpointing performance bottle necks, we further examine a more complex scenario to gauge the effect these attacks have on a real-world web system. In this experiment, SYN cookies are enabled on all systems. Without SYN cookies the throughput of our web server drops to zero on all systems. This is because without SYN cookies, the TCP connection queue is quickly filled by the TCP SYN DoS packets, leaving no room for legitimate connections from our client simulator.

A. Requests Per Second

We start our evaluation from the maximum throughput of each system. We express the maximum throughput as

TABLE VII
TBENCH NETWORK BENCHMARK RESULT

	Base		Attack			Attack SYN cookies		
System	MB/s	ST-DEV (%)	MB/s	ST-DEV (%)	Performance Degradation (%)	MB/s	ST-DEV (%)	Performance Degradation (%)
KVM	16.1	0.88	11.1	1.21	30.91	10.7	0.49	33.34
Xen	41.8	0.37	38.7	0.21	7.44	36.0	0.96	14.02
OpenVZ	47.9	0.09	47.3	0.75	1.23	46.2	0.40	3.54
Vanilla	48.2	0.15	47.8	0.14	0.86	47.2	0.08	2.09

TABLE VIII
SYSBENCH IO BENCHMARK RESULT

	Base		Attack			Attack SYN Cookies		
System	MB/s	ST-DEV (%)	MB/s	ST-DEV (%)	Performance Degradation (%)	MB/s	ST-DEV (%)	Performance Degradation (%)
KVM	4.79	5.83	3.52	1.83	26.59	3.40	6.70	29.09
Xen	7.89	1.70	6.17	2.75	21.80	5.89	1.00	25.28
OpenVZ	5.68	4.09	5.55	6.70	2.32	5.52	5.49	2.73
Vanilla	4.72	0.53	4.71	0.80	0.32	4.69	1.05	0.80

TABLE IX
REQUEST SERVICED PER SECOND

	Base		Attack		
System	Requests/s	ST-DEV (%)	Requests/s	ST-DEV (%)	Performance Degradation (%)
KVM	1654.8	0.21	1267.7	0.96	23.39
Xen	1850.7	0.12	1481.1	0.28	19.97
OpenVZ	2886.5	0.02	2556.0	0.16	11.44
Vanilla	3179.8	0.11	2949.4	0.31	7.24

maximum number of requests serviced per second. As can be seen in Table IX, KVM and Xen have significantly lower performance in the base line test than OpenVZ and Vanilla. Both KVM and Xen service 35% less requests per second that OpenVZ or Vanilla. As expected, under DDoS conditions all systems experienced measurable performance degradation. Vanilla was the least susceptible falling by less then 8% followed by OpenVZ at less than 12%. Xen suffered a 20% performance degradation and KVM lost over 23%. When the systems are under attack, KVM and Xen provided less than 50% of the throughput that the Vanilla host can provide while using the same amount of system resources.

B. Increase in Connection time

Next, we measure the average time taken to establish the HTTP connection to the web server. The Apache benchmark collects the time taken to create each connection to the server and calculates the average from all connections. The results are given in Table X. We can see from the table that all connection times are relatively close. Interestingly KVM and Xen are both measurably faster than OpenVZ and Vanilla in the baseline tests.

The attack scenario showed some very interesting results. Note that the connections to the web server in KVM and Xen both took a considerably longer time under our 10 Mb/s DoS. Specifically, KVM increased from 6 ms to almost 33 ms, an increase of nearly 4.5 times; Xen increased from an average connection time of 6 ms to a staggering 84 ms, which is over 13 times longer. On closer inspection we found that, in both KVM and Xen, the median connection time remained the same. The massive change in the average connection time is because of an increase in connections that take an abnormally long amount of time to establish. We observed that, when Xen and KVM were under attack, up to 5% of connections take over 250 ms to establish. Xen suffered the worst abnormally long connection times, with up to 2% of its connections taking over 2500 ms to establish. Overall, we can see that the connection processing time is greatly affected in our hypervisor-based virtualized systems.

TABLE X
INCREASE IN AVERAGE CONNECTION TIME

	Base	Attack	
System	Connection Time (ms)	Connection Time (ms)	Increase (%)
KVM	6	32.75	445.83
Xen	6	84.25	1304.17
OpenVZ	8	9	12.50
Vanilla	8	8	0

X. FURTHER DISCUSSIONS AND CONCLUSION

The use of end-point defenses such as SYN cookies has mitigated much of the effect that TCP SYN-Floods have on nonvirtualized systems. However, our experiments have shown that they do not provide adequate protection for hypervisor-based virtualization systems such as KVM and Xen. One possible solution is to implement SYN-Proxies in the hypervisor. Since a SYN-Proxy works by only forwarding a connection once the final ACK is received, this modification could prevent the attack traffic from reaching the VM. In Section VII, we show that any small packet sent at a high rate can cause degradation to a virtualized system; a SYN-proxy will not solve the general problem of small packets sent at a high rate. A more general solution could involve modifying the network virtualization drivers to make them more efficient at processing small packets. For example, instead of having the hypervisor deliver every packet as it arrives, a more efficient strategy could be to wait for a small time out, and deliver all packets that arrived in that time period. In a related work, we modified the VirtIO drivers in KVM and verified that this buffering strategy does indeed alleviate some of the overhead [37].

The performance implications of larger DoS attacks have yet to be quantified. Based on our CPU usage tests in Section VII, high data rates will clearly lead to larger performance degradation. Our preliminary experiments with high data rates also show that any small packets sent at high rate can severely degrade both KVM and Xen systems.

Recent advances in hardware design have led to products that help alleviate the inherent overhead of network virtualization. In particular, SR-IOV replicates certain hardware on the network card to provide each VM with more direct access to the device. This technology has been shown to offer a increase in throughput and a reduction in overhead when run against paravirtualization drivers such as VirtIO [38]. The same research however, indicates that the CPU overhead of processing small packets remains much higher in virtualized system.

We have also preformed preliminary experiments on two other Virtualization systems, namely VMware Server and Oracle's Virtual-Box. The experiments indicate that a similar degradation is experienced by these systems while under a DoS attack. This is a strong indication that this problem is indeed general to hypervisor based virtualization and not isolated to KVM and Xen.

Further, though the TCP SYN DoS attack targets mainly the network infrastructure of a system, it has serious impact on the performance of other system components in Xen and KVM. In particular, they lost a considerable amount of throughput to their respective file systems. Much of this degradation owes itself to the large amount of CPU overhead we found when the virtualized systems are under attack. In the global Internet, a 10 Mb/s SYN flood is considered rather small; however both the KVM and Xen hypervisors used the CPU time of an entire core, simply servicing that level of attack. When compared to OpenVZ and Vanilla, which used only 6% of a single core to serve the same attack, it becomes clear that it is significantly more expensive to use hypervisor-based virtualization on systems exposed to DoS traffic.

In this paper, through extensive experiments, we showed that even a light DoS attack on a virtualized system can have serious performance impacts. Our experiments suggested that all virtualization techniques suffer from greater performance degradation compared with its nonvirtualized counter parts. This is, particularly, severe for PVM and HVM due to their inherent virtualization structure. The container-based virtualization thus becomes a potentially better solution for a system exposed to DoS attacks.

References

[1] CDW. (2010, Jan.). *CDW's Server Virtualization Life Cycle Report* [Online]. Available: http://www.cdwnewsroom.com/

[2] Commvault Newsroom. (2011, Aug.). *Commvault Releases Results of Annual End-User Virtualization Survey Emphasizing the Need for Modern Approach to Protecting Managing Virtual Server Environments* [Online]. Available: http://news.commvault.com/

[3] W. Vogels, "Beyond server consolidation," *Queue*, vol. 6, pp. 20–26, Jan. 2008.

[4] [Online]. Available: http://www.xen.org/

[5] [Online]. Available: http://user-mode-linux.sourceforge.net/

[6] P. Padala, X. Zhu, Z. Wang, S. Singhal, and K. Shin, "Performance evaluation of virtualization technologies for server consolidation," HP Labs Tec. Rep. HPL-2007-59R1, 2007.

[7] K. Ye, X. Jiang, S. Chen, D. Huang, and B. Wang, "Analyzing and modeling the performance in Xen-based virtual cluster environment," in *Proc. 12th IEEE Int. Conf. High Performance Comput. Commun.*, Sep. 2010, pp. 273–280.

[8] R. Russell, "Virtio: Toward a de-facto standard for virtual I/O devices," *SIGOPS Oper. Syst. Rev.*, vol. 42, pp. 95–103, Jul. 2008.

[9] [Online]. Available: http://www.vmware.com/

[10] [Online]. Available: http://www.linux-kvm.org/

[11] [Online]. Available: http://www.virtualbox.org/

[12] [Online]. Available: http://www.openvz.org

[13] [Online]. Available: http://linux-vserver.org/

[14] [Online]. Available: http://www.oracle.com/technetwork/systems/containers/index.html

[15] S. Soltesz, H. Pötzl, M. E. Fiuczynski, A. Bavier, and L. Peterson, "Container-based operating system virtualization: A scalable, high-performance alternative to hypervisors," *SIGOPS Oper. Syst. Rev.*, vol. 41, pp. 275–287, Mar. 2007.

[16] S. Wardley, E. Goyer, and N. Barcet. (2009, Aug.). Ubuntu enterprise cloud architecture. *Technical White Paper* [Online]. Available: http://www.canonical.com

[17] D. Nurmi, R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff, and D. Zagorodnov, "The eucalyptus open-source cloud-computing system," in *Proc. 9th IEEE/ACM CCGRID*, May 2009, pp. 124–131.

[18] J. Postel, "RFC 793: Transmission control protocol," 1981.

[19] W. Eddy, "RFC 4987: TCP SYN flooding attacks and common mitigations," 2007, p. 19.

[20] W. Eddy, "Defenses against tcp syn flooding attacks," *Cisco Internet Protocol J.*, vol. 8, no. 4, pp. 2–16, 2006.

[21] H. Wang, C. Jin, and K. G. Shin, "Defense against spoofed ip traffic using hop-count filtering," *IEEE/ACM Trans. Netw.*, vol. 15, no. 1, pp. 40–53, Feb. 2007.

[22] C. Jin, H. Wang, and K. Shin, "Hop-count filtering: An effective defense against spoofed ddos traffic," in *Proc. 10th ACM Conf. Comput. Commun. Security*, 2003, pp. 30–41.

[23] D. Bernstein. (1996). *Syn Cookies* [Online]. Available: http://cr.yp.to/syncookies.html

[24] J. Matthews, W. Hu, M. Hapuarachchi, T. Deshane, D. Dimatos, G. Hamilton, M. McCabe, and J. Owens, "Quantifying the performance isolation properties of virtualization systems," in *Proc. Workshop Exp. Comput. Sci.*, 2007, article 6.

[25] S. Ostermann, A. Iosup, N. Yigitbasi, R. Prodan, T. Fahringer, and D. Epema, "A performance analysis of EC2 cloud computing services for scientific computing," *Cloud Comput.*, vol. 34, pp. 115–131, Nov. 2010.

[26] G. Wang and T. Ng, "The impact of virtualization on network performance of amazon EC2 data center," in *Proc. IEEE INFOCOM*, Mar. 2010, pp. 1–9.

[27] R. Shea and J. Liu, "Network interface virtualization: Challenges and solutions," *Network, Special Issue: Wired Wireless Network Virtualization*, vol. 26, no. 5, pp. 28–34, Sep.–Oct. 2012.

[28] A. Iosup, S. Ostermann, N. Yigitbasi, R. Prodan, T. Fahringer, and D. Epema, "Performance analysis of cloud computing services for many-tasks scientific computing," *IEEE Trans. Parallel Distributed Syst.*, vol. 22, no. 6, pp. 931–945, Jun. 2011.

[29] J. Ekanayake and G. Fox, "High performance parallel computing with clouds and cloud technologies," *Cloud Computing*, vol. 34, pp. 20–38, Nov. 2010.

[30] K. Jackson, L. Ramakrishnan, K. Muriki, S. Canon, S. Cholia, J. Shalf, H. Wasserman, and N. Wright, "Performance analysis of high performance computing applications on the amazon web services cloud," in *Proc. 2nd IEEE Int. Conf. Cloud Computing Technol. Sci.*, Nov.–Dec. 2010, pp. 159–168.

[31] [Online]. Available: http://www.hping.org/

[32] [Online]. Available: http://www.7-cpu.com/

[33] [Online]. Available: http://sysbench.sourceforge.net/

[34] [Online]. Available: http://iperf.sourceforge.net/

[35] [Online]. Available: http://linux.die.net/man/1/tbench

[36] [Online]. Available: http://jmob.ow2.org/rubbos.html

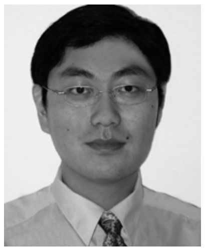
[37] R. Shea and J. Liu, "Understanding the impact of denial of service attacks on virtual machines," in *Proc. IEEE 20th Int. Workshop Quality Service*, Jun. 2012, p. 27.

[38] J. Liu, "Evaluating standard-based self-virtualizing devices: A performance study on 10 GbE NICS with SR-IOV support," in *Proc. IPDPS*, Apr. 2010, pp. 1–12.

Ryan Shea (S'08) received the B.Sc. degree in computer science from Simon Fraser University, Burnaby, BC, Canada, in 2010, where he is currently pursuing the Ph.D. degree with the Network Modeling Laboratory, and received the Certificate in University Teaching and Learning from the same university.

He has been a Network Administrator with various nonprofit groups. His current research interests include computer and network virtualization, and performance issues in cloud computing.

Mr. Shea received the Best Student Paper Award at IEEE/ACM IWQoS'2012.



Jiangchuan Liu (S'01–M'03–SM'08) received the B.Eng. degree (*Cum Laude*) from Tsinghua University, Beijing, China, in 1999, and the Ph.D. degree from the Hong Kong University of Science and Technology, Clear Water Bay, Hong Kong, in 2003, both in computer science.

He is currently an Associate Professor with the School of Computing Science, Simon Fraser University, Burnaby, BC, Canada. He was an Assistant Professor with the Department of Computer Science and Engineering, Chinese University of Hong Kong, Shatin, Hong Kong, from 2003 to 2004. His current research interests include multimedia systems and networks, wireless *ad hoc* and sensor networks, and peer-to-peer and overlay networks.

Dr. Liu is a member of Sigma Xi. He is an Associate Editor of the IEEE TRANSACTIONS ON MULTIMEDIA, and an editor of IEEE COMMUNICATIONS SURVEYS AND TUTORIALS. He is the TPC Vice Chair for Information Systems of IEEE INFOCOM'2011. He was a recipient of the Microsoft Research Fellowship in 2000, the Hong Kong Young Scientist Award in 2003, and the Canada NSERC DAS Award in 2009. He was a co-recipient of the IEEE ComSoc Best Paper Award on Multimedia Communications in 2009, the Best Paper Award of IEEE Globecom'2011, and the Best Student Paper Award of IEEE/ACM IWQoS'2008 and IWQoS'2012.