

Understanding the Impact of Denial of Service Attacks on Virtual Machines

Ryan Shea
Simon Fraser University
Burnaby, Canada
Email: ryan_shea@sfu.ca

Jiangchuan Liu
Simon Fraser University
Burnaby, Canada
Email: jcliu@cs.sfu.ca

Abstract—Virtualization, which allows multiple Virtual Machines (VMs) to reside on a single physical machine, has become an indispensable technology for today’s IT infrastructure. It is known that the overhead for virtualization affects system performance; yet it remains largely unknown whether VMs are more vulnerable to networked Denial of Service (DoS) attacks than conventional physical machines. A clear understanding here is obviously critical to such networked virtualization system as cloud computing platforms.

In this paper, we present an initial study on the performance of modern virtualization solutions under DoS attacks. We experiment with the full spectrum of modern virtualization techniques, from paravirtualization, hardware virtualization, to container virtualization, with a comprehensive set of benchmarks. Our results reveal severe vulnerability of modern virtualization: even with relatively light attacks, the file system and memory access performance of VMs degrades at a much higher rate than their non-virtualized counterparts, and this is particularly true for hypervisor-based solutions. We further examine the root causes, with the goal of enhancing the robustness and security of these virtualization systems. Inspired by the findings, we implement a practical modification to the VirtIO drivers in the Linux KVM package, which effectively mitigates the overhead of a DoS attack by up to 40%.

I. INTRODUCTION

Virtualization has recently become an indispensable technology in terms of both academic and industrial applications. At its core, computer virtualization allows multiple Virtual Machines (VMs) to reside on a single physical machine. The applications of this exciting technology are both interesting and far reaching. A recent survey showed that 90% of organizations use Virtual Machines in some capacity in their IT infrastructures [1]. A further 34% of organizations use virtualization to meet the majority of their server needs [2]. Organizations were quick to uptake this technology due to promises of increased utilization of resources, increased server consolidation, and decreased costs [3]. Virtualization is also a prominent technology allowing the long held dream of utility computing platforms such as cloud computing to become a reality. One prominent example is the current market leader in Cloud Computing Amazon EC2, which utilizes the Xen virtualization systems to provide customers with computational resources.

With any new technology there are downsides and tradeoffs; virtualization is no exception to this rule. With the ability to run multiple different VMs and even different operating systems at the same time comes the cost of considerable overhead. This overhead effects all components of the system; however it has been noted to be especially high in terms of network performance [4]. Given the inherent overhead of networking on virtualized systems, it is a surprise to find that

very little research has been done to quantify the effect of networked Denial of Service (DoS) attacks on these systems.

In this paper, we explore the performance of modern computer virtualization systems under a TCP based Denial Of Service (DoS) attacks. We devise a representative set of experiments to examine the performance of some typical virtualization techniques under TCP-based DoS attacks. By comparing the performance of virtualized and non-virtualized systems under a TCP based DoS attack we show that virtualized systems are much more susceptible to networked DoS attacks than their non-virtualized counterparts. Even with relatively light attacks, the file system and memory access performance of hypervisor-based virtualization degrades at a much higher rate than their non-virtualized counterparts. Further, while under a DoS attack we show a web server hosted in a virtualized system can serve 23% fewer request per second, while our bare-metal server hosted in the same hardware degrades by only 8%. These observations suggest that the state-of-the-art virtualization solutions need to be substantially revisited in this perspective. We further examine the root causes, with the goal of enhancing the robustness and security of these virtualization systems. With the data gained from our experiments, we show that with a few clever modifications hypervisor-based virtualization systems can be made much more resistant to networked DoS attacks. We implement our modifications into a real-world virtualization system and show we can mitigate the overhead of a DoS attack by up to 40%.

The rest of the paper is organized as follows. In Section II, we present an overview of modern virtualization solutions. In Section III, we overview the DoS attack and its implications regarding virtual machines. A series of related works are discussed in Section IV, followed by experimental architecture and benchmark settings in Sections V and VI, respectively. We then present and discuss the results in Section VII. In Section VIII, we use modern performance profiling techniques to discover the underlying causes of performance degradation. In Section IX we use the information gained from our profiling to harden KVM against DoS attacks. Finally in Section X we further discuss our findings and conclude the paper.

II. OVERVIEW OF VIRTUALIZATION

To thoroughly 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. The kernel 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 flexibility in terms of choosing the operating systems. In particular, PVM must use an OS that can be modified to work with the hypervisor. Although this does not present a significant problem for such open-source OSes as Linux, it does create problems with such proprietary OSes as Microsoft Windows. Yet PVM reduces the overhead to virtualize privileged operating systems calls, as special hardware is not needed to intercept them. Typical PVM solutions include Xen and User Mode Linux.

B. Hardware Virtual Machine (HVM)

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*. Examples include AMD's AMD-V and Intel's VT-X. They detect if 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. 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 [4][5]. Yet paravirtualization I/O drivers can alleviate such overhead; one example of a paravirtualization driver package is the open source VirtIO [6]. Representative Virtualization solutions that are HVM include VMware Server, KVM, and Virtual-Box.

C. Container Virtualization

Container Virtualization, also known 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 other containers. It has been shown to have the lowest overhead among all the existing virtualization techniques [4]. 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, Linux-VServer and Solaris Zones.

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 hypervisor acting as a gatekeeper to the underlying hardware generally improves performance isolation between guests on a host. However it has been noted that the hypervisor can also introduce measurable overhead [7].

In our experiments, we chose Xen, KVM and OpenVZ to be evaluated under DoS attacks. We believe this choice is representative for the following two reasons. First, they are all open-source with publicly available documents and with cross-platform implementations. We can run their packages on 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 and Eucalyptus Cloud Service [8]; OpenVZ is a popular choice in offering Virtual Private Server(VPS) containers to the public.

III. OVERVIEW OF DENIAL OF SERVICE (DOS)

Denial of Service (DoS) attacks are attempts by a non-legitimate user to degrade or deny resources to legitimate users. There are many different forms of DoS attacks. 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 difficult to filter out before it reaches the end system. As we will show later, however, that our findings are indeed general and not simply confined to TCP SYN flood. In this section, we first give a brief discussion on the TCP SYN flood attack and its potential threat to virtual machines.

A. TCP SYN Flood

The Transmission Control Protocol (TCP) is one of the foundations of the global Internet. TCP provides reliable in-order delivery of whatever data the users wish to send. 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.

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 3-way handshake. An attacker tries to overload a victim with so many connection requests that it will not be able to respond to legitimate requests. This is done through sending many TCP SYN packets to the victim. The victim allocates buffers for each 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 [9]. Flooding based attacks can also exhaust other resources of the system such as CPU time.

B. TCP DoS 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 in the network or on the end host. Network-based solutions include firewall proxies, which only forward the connection request after the client side ACK is received [9]. In particular Hop Count filtering, inspects the packet’s TTL field and drops suspected spoofed packets. It has been reported that this technique can achieve up to 90% detection rate [10][11]. 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 [12]. 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 [13].

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 DoS 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

There have been many performance analyses performed on different applications and scenarios in virtualized systems [14][15][16]. In 2007, researchers from the University of Michigan and HP conducted a performance evaluation comparing different virtualization techniques for use in server consolidation [4]. 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 than Xen.

Soltész *et al.* [7] compared Xen and Linux VServer in terms of performance and architectural design. Matthews *et al.* [17] tested HVM, PVM and Container Virtualization for performance isolation. 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.* [18] conducted 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 [19]. It showed that, due to virtualization, users often experience bandwidth and delay instability.

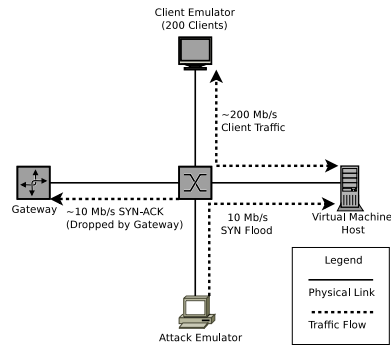


Fig. 1: Network Setup and Traffic Flow

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 chose to create our custom test system instead of using rented resources from a cloud provider for several reasons. The first is that cloud providers such as Amazon EC2 have specific rules regulating security testing on their systems. To our knowledge all large cloud providers specifically list DoS testing as a non-permissible activity. Second, using our own custom system is the only way to ensure hardware resources remain constant across several different tests. Finally, no cloud provider provides its users with direct access to the Virtual Machine host, making some measurements impossible to perform. We 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 Hardware Virtual Machines (HVM) support. The PC was equipped with 4 GB of 1333 MHZ DDR-3 SDRAM and a 320 GB 7200 RPM hard drive with 16MB 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 shown in the Figure 1. The figure also shows traffic flows for our benchmarks. All machines on the network are attached directly to a Linksys 1000Mb/s SOHO switch. The attack emulator has been configured to have a link rate of only 10Mb/s using the `ethtool` configuration tool. 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 gateway were not present, many gratuitous ARP requests would have been created by the virtual machine host, as it searches for a route to deliver the packets to the external network. Our choice to place the Client Emulator and Attack Emulator inside the gateway may seem unintuitive. However this is done for the following reason: To ensure the accuracy of our tests we must only allow the DoS attack to degrade the virtual machine host and not other network components. Also large networks which perform load-balancing or traffic filtering, often have different routes for received and transmitted data, making it not uncommon to find similar network setups in practice.

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. Yet most of our observations and conclusions are general, not simply confined to TCP SYN flood. In our experiments, we assume that a 100Mb/s distributed SYN flood is being performed on our network and we have successfully filtered out 90% of the attack. A total of 10Mb/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 effecting legitimate clients [10]. 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` 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 addresses. We target an open port on the target machine. In the case of our synthetic benchmark, it is the SSH port 22 and in our comprehensive benchmarks, it is Apache2 running on port 80. The DoS traffic originates from our attack emulator, which can be seen in Figure 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. We now describe the system setup of these virtualization solutions.

1) *Xen System Setup*: We installed the Xen 4.0 Paravirtualization Hypervisor on our test system following closely the Debian guide. 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 as we already had LVM running on our host machine. To install a base system into the image, we used the utility `xen-tools`, which automates the install procedure. We set the number of virtual CPUs (VCPU) to 4 and the amount of RAM to 2048 MB. The virtual machine host ran the 2.6.35-5-Xen kernel.

2) *OpenVZ System Setup*: We installed the OpenVZ container-based virtualization package from the Debian repository using its guide. 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 4 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. The virtual machine host ran the 2.6.35-5-OpenVZ kernel.

3) *KVM System Setup*: KVM is relatively simple to install since it only requires a kernel module instead of a special patched kernel. We used KVM version 0.12.5 compiled from the official source repository. We manually installed an instance of Debian Squeeze from a mounted CD image into the KVM virtual machine. Once again the virtual machine was given access to all 4 processor cores as well as 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 [6]. Debian kernel 2.6.35-5-amd64 was used in the virtual machine to stay consistent with the other tests.

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

VI. BENCHMARK SETUP

We have chosen a broad set of synthetic benchmarks and a single comprehensive benchmark to evaluate the impact of DoS attacks on different components; specifically CPU, Network, Memory, and File System performance under normal and attack conditions.

A. CPU Benchmark

We chose the SysBench CPU test to measure the CPU performance, which is well known and regularly used for gauging raw CPU performance.

It continues to calculate prime numbers 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 find all primes less than 100,000 and assigned 4 threads, so that each of our 4 cores would be involved in the benchmark. We then recorded the total amount of time it took to find the primes in the first 100,000 integers.

B. Memory Benchmark

For memory benchmarking, we chose the SysBench memory bandwidth test. It allocates a segment of either global or thread local memory and performs read or write operations on it, outputting the total time taken as well as the memory bandwidth in MB/s. In our experiments, we assign a single thread to perform a 20GB read of main memory.

System	TCP DoS		UDP Flood	
	10Mb/s	100Mb/s	10Mb/s	100Mb/s
KVM	~102%	~205%	~64%	~272%
Xen	~98%	~187%	~46%	~145%
OpenVZ	~8%	~100%	~1%	~1%
Vanilla	~6%	~90%	~1%	~1%

TABLE I: CPU Usage While Under Attack – System Idle

C. File System Benchmark

The file system performance was tested using the SysBench 'fileio' test, which 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 performed random read and write operations on the files. We assign 16 threads each attempting to perform random blocking read or write operations. The results are given as MB/s.

D. Network Benchmark

To test network performance we use `Iperf`, which attempts to find the maximum TCP or UDP bandwidth (in Mb/s) between two network hosts. In our experiments, the client emulator in our network was chosen as the server and TCP was used as the protocol. We run each test for the default time setting of 10 seconds.

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 MySQL Server 5.1. To create a web application representative of a real-world service, we installed the RuBBoS bulletin board benchmark. We chose the PHP version of the RuBBoS and installed the necessary Apache extensions for PHP. We then installed the RuBBoS data into our MySQL database.

Although RuBBoS comes with its own client simulator, we used the Apache benchmark instead. The latter has been more commonly used for web server stress testing. Although the RuBBoS simulator can perform tests specific to RuBBoS, we only require maximum request rate, which are more straight forward to extract with the Apache Benchmark.

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. Figure 1 shows the network configuration and the traffic flows during this experiment.

VII. EXPERIMENTAL RESULTS

A. CPU Usage During DoS

We first measure the impact of DoS traffic on CPU usage while the system is idle. To provide a target for the TCP DoS, we ran a Secure Shell (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 Mb/s and 100 Mb/s TCP SYN

floods. We also included the CPU usage from a 10 Mb/s and 100 Mb/s UDP flood. The results are given in Table I.

Under a 10Mb/s TCP DoS, 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 6-8% of CPU time on a core to service the same attack. If we increase the attack rate to 100Mb/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 it is exposed to higher attack rates.

Though our focus is on TCP SYN attack, we have also devised a UDP flood 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 10Mb/s and 100Mb/s of UDP traffic targeted the system. To further simplify the test we did not randomize the source address and the packets contained no data. By not randomizing the source address we can determine if it is the cause of the CPU overhead. As can be seen from Table I, the high CPU usage is present for both KVM and Xen in the UDP case as well, suggesting that this happens 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 packets sent at a high rate reaching the end system leads to a similar phenomena. Other researchers have taken notice of this overhead during processing packets and have managed to reduce the CPU usage by 29% [20]. Even with this reduction, however, DoS attacks against hypervisor-based virtual machines still generate substantially more CPU usage than their counterparts.

Next, we benchmark our systems under a 10Mb/s TCP DoS to quantify the performance degradation experienced by the systems.

B. Synthetic Benchmarks Results

For each test, we ran each benchmark 4 times and calculated the mean. To ensure our results are statistically significant, we also calculated the standard deviation for each measurement and display it as error bars on our graphs. To quantify performance degradation, we calculate percentage change from baseline to attack conditions.

1) *Result CPU Benchmark:* The SysBench Prime benchmark is given in Figure 2a and a lower completion time is better in this case. We can see that in the non DoS case all virtualization techniques perform within 5% of each other. This is not surprising, since VM CPU scheduling is quite similar to OS process scheduling.

However we can see from the CPU benchmark that even a small 10Mb/s DoS has a significant effect on the CPU performance of both KVM and Xen. KVM suffers from a massive 35.8% increase in completion time; Xen also suffers a large increase at 31.6%. For both Xen and KVM this increase is due to the amount of time the hypervisor spends on the CPU servicing the attack packets. OpenVZ and Vanilla fared much better, both with a small but still measurable amount of performance degradation of 7.5% and 5.5%, respectively.

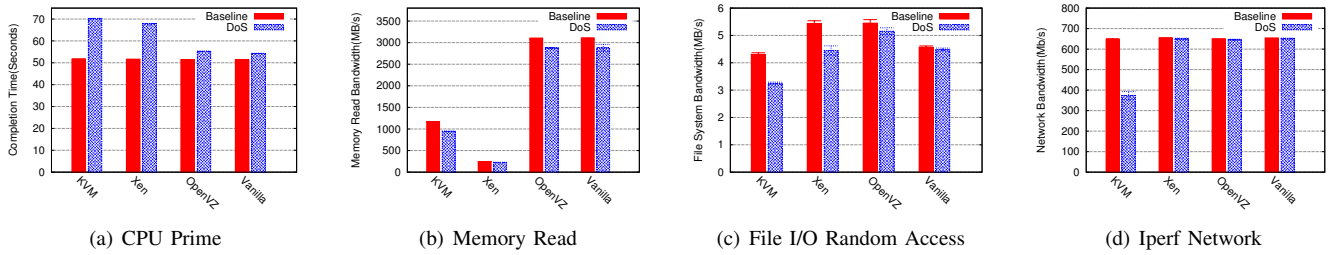


Fig. 2: Synthetic CPU, Memory, File System and Iperf Benchmark Result

2) *Result Memory Benchmark*: The memory benchmark results shown in Figure 2b are intriguing, as there is a wide variation in the base line performance. In this particular benchmark, Xen fared by far the worst, being over 12x slower than our Vanilla system setup. KVM fared much better than Xen but still managed less than half of the memory read speed of Vanilla or OpenVZ.

Under the TCP DoS, all setups showed a measurable slowdown in performance, with Vanilla, OpenVZ and Xen having all approximately 7% performance degradation. KVM, on the other hand, experienced a slowdown of 19.4%. 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 hypervisor is busy servicing I/O request created by the DoS 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.

3) *Results I/O benchmark*: The synthetic benchmark results for the SysBench I/O test are given in Figure 2c. 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 and OpenVZ showed significantly faster performance than the others tested. However, the physical location on the disk where the 2 GB of files are allocated can make a large difference in bandwidth and seek time. For this 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 degradation was within the deviation, indicating very little degradation. On the other hand, both KVM and Xen lose considerable performance. Xen suffered from 18.0% lower random access to its file system. KVM lost the most performance, which is 24.5%. 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.

4) *Result Network Benchmark*: The Iperf results are given in Figure 2d. The Iperf benchmark measures maximum transfer rate between two hosts. The maximum transfer rate base line is almost identical for all systems.

Under TCP DoS attack, the IPerf results showed no significant drop in the throughput of Xen, OpenVZ or Vanilla systems. KVM however loses a massive 42.2% of performance for a total of 272 Mb/s less throughput. This is somewhat surprising considering the attack only creates 10Mb/s of TCP SYN packets, which can stimulate the KVM VM to produce at most 10Mb/s of SYN-ACK packets. This means KVM is experiencing a total of 20Mb/s extra traffic over the baseline.

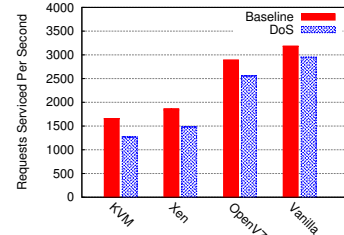


Fig. 3: Web Application Benchmark Result

The loss of bandwidth however, is several times this extra traffic; in section IX we will show this phenomena is related to a high number of interrupt requests.

C. Comprehensive Benchmark Results - Web Application

From the previous 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 bottlenecks, we further examine a more complex scenario to gauge the effect these attacks have on a real-world web system.

To perform our benchmark we first find the maximum throughput of each system. We express the maximum throughput as maximum number of requests serviced per second, which can be seen in Figure 3. OpenVZ suffers a measurable 9.3% decrease in throughput when compared to Vanilla. However both KVM and Xen have significantly lower performance in the base line than both OpenVZ and Vanilla. Xen can service 41.2% fewer requests per second. KVM is the slowest, servicing 48.0% fewer request per second than our non-virtualized Vanilla system.

As expected, under DoS conditions, all systems experienced measurable performance degradation. Vanilla was the least susceptible falling by 7.2%, followed by OpenVZ at 11.4%. Xen suffered a 20.0% performance degradation and KVM lost 23.4%. When the systems are under attack, KVM and Xen provided nearly 50% less throughput than the Vanilla host provided while using the same amount of system resources.

VIII. A CLOSER LOOK: CAUSES OF DEGRADATION

In order to better understand the causes of the performance degradation that we have found, we take a closer look at these hypervisor-based virtualization systems. To accomplish this, we use hardware performance counters to measure interrupt requests (IRQs), processor cache activity, and CPU context switch events. Our goal is to compare the frequency of these events on virtualized and non-virtualized systems under denial of service attacks. With this information, we made modifications to the virtualization software in order to improve performance of the virtualized system under a DoS attack.

	Vanilla	KVM
LLC References	206,835,021	1,292,316,729
LLC Misses	11,486,718	36,060,083
Context Switches	956	708,335
IRQs/Second	8,500	37,000

TABLE II: Performance Metrics Under DoS – System Idle

A. KVM Profiling

KVM was chosen as our representative hypervisor-based virtualization system in these tests, for several reasons. The first is that KVM runs on the unmodified Linux kernel, and is compatible with the standard performance profiling tools; Xen on the other hand, has a specialized kernel and therefore may have compatibility problems with these tools. Also, because KVM does not require a modified kernel, it is easier to modify the hypervisor and drivers. Second, KVM supports the VirtIO drivers, a para-virtualization package intended to be standard across various virtualization techniques, including Xen. By using VirtIO, any driver changes that are required in order to enhance performance will be easier to adapt to different virtualization systems.

To collect the system level information, we used `perf`, a performance analysis tool for Linux. It uses built-in Linux kernel support and hardware performance counters to measure system-wide or process-level events. To collect the data, we disabled all non-operating system processes on our test system, effectively creating an idle state. The systems operated under the same specifications previously described. We then configured `perf` to track system wide events on all cores and collected our data during a 10Mb/s random source SYN flood DoS attack. To get an accurate reading we sampled the system performance 5 times, for 10 seconds each time and calculated the mean and standard deviation for each of our metrics. We then used the `perf top` command to obtain the current system statistics, in particular, IRQs per second.

As can be seen in Table II, the samples show considerable difference in these metrics between the Vanilla system and KVM. In terms of LLC(last level cache) performance, KVM while under attack is considerably less efficient with the use of its LLC. KVM references its cache over 6 times more than the Vanilla system. In terms of last level cache misses, KVM is over 3 times worse. The number of cache misses and cache references is not surprising considering how much extra memory copying is required to deliver a packet from the network interface to its final destination in the KVM virtual machine.

Context switches also provided some interesting insights into the differences between virtualized and non virtualized systems. The Vanilla system, during our 10 seconds of measurements, had 956 context switches only, compared to a staggering 708,335 with KVM, an increase of over 700 times. This massive difference can be explained by how these two systems handle network packets. In the Vanilla system, services to the network stack and the driver domain are often handled by the same kernel process. Because of this, there are much fewer context switches as the same process is responsible for both actions. KVM however is in a strikingly different situation, in which a packet and its response must traverse multiple processes in order to be serviced. This is because when the packet is received on the physical device, it is copied

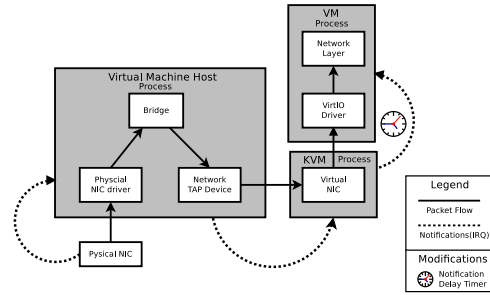


Fig. 4: KVM Network Architecture and Modifications

by the host machines kernel into the memory space of the virtual machine; the virtual machine is then notified of the incoming packet and it is scheduled to run on the processor; the virtual machine delivers the packet to its network stack where a response SYN-ACK is generated and sent on its virtual interface; the host kernels process then collects the packet from the virtual machine and delivers it to the network. All these extra steps, when applied to each packet, contribute to the sheer number of context switches we observed. Figure 4 illustrates the path a received packet must traverse and the figure also shows a performance modifications we will discuss in the next section.

The final metric we looked at was interrupt requests. When under attack the Vanilla system generated approximately 8,500 interrupt requests per second. The KVM hypervisor generated over 4 times more requests, at approximately 37,000 interrupt requests per second. The interrupt requests are used by KVM to notify the virtual machine when it has new packets and also to notify the kernel when there are new packets to be sent. The TCP SYN flood results in a large number of small packets being sent and received by the virtual machine, which results in the large number of requests we observed.

IX. REDUCING DOS OVERHEAD KVM

Based on what we learned from our profiling of KVM, we decided to modify the VirtIO drivers in KVM to reduce the number of interrupt requests produced during a DoS attack, with the goal of improving the overall CPU performance of our virtualized system. We focused on interrupt requests specifically because they are a clear cause of context switches and ultimately affect cache efficiency as well.

A. Modifying KVM and VirtIO

As mentioned above, in KVM an interrupt request is generated whenever packets need to be sent from the host to the virtual machine, or from the virtual machine to the host. The KVM VirtIO drivers contain an option to bundle packets with a single interrupt for sending, but receiving lacks this feature. In our efforts to reduce the number of interrupts, we modified the VirtIO drivers to create a mechanism that bundles together several received packets with a single interrupt. To accomplish this, we examined the VirtIO driver and modified how KVM notifies the virtual machine that there is a packet to be received. When the packet rate is high, instead of immediately notifying the virtual machine that it has received a packet, the KVM hypervisor sets a timer. When the timer expires, the virtual machine is notified of the initial packet, and every other packet that has arrived during the length of the timer. Figure 4 shows the KVM architecture and our modification.

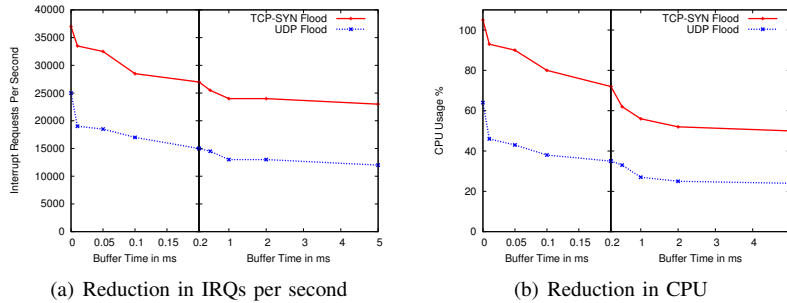


Fig. 5: Reduction in Interrupts and CPU Overhead

To make our modifications adaptive, they only become active when there is a high packet rate. To detect the packet rate, we keep a simple counter of the number of received packets and check it every second to see if the VM passes a certain threshold of packets per second; if the threshold is passed then the modifications are enabled; if the threshold is not reached we revert to the original VirtIO drivers. We applied the same technique to the sending queue as well. For our tests, we assume that a 'high' packet rate threshold of 10,000 packets per second; however this is adjustable based on the application. Next, we experiment with the length of the timer, or how long to buffer the packets before generating an interrupt request.

B. Modification results

Using the same network and system specifications described in our previous experiments, we tested the effect that increasing the buffer time of our modified VirtIO driver had on interrupt requests per second and CPU usage. We tested candidate buffer times from 0 milliseconds up to 5 milliseconds. For each candidate buffer time, we measured the IRQs per second and CPU usage of our system while under a 10Mb/s TCP random source SYN flood and a 10Mb/s UDP flood. The results for IRQs per second are shown in Figure 5a, and the results for CPU usage are shown in Figure 5b. As can be seen, when we increase the buffer time, we see a decrease in both interrupts and CPU usage. Moreover, the interrupts and CPU usage show a very similar pattern of decrease, and have an R^2 value of over 0.97 when plotted against one another in a linear regression, for both TCP and UDP floods. This confirms that interrupts are closely related to CPU usage in KVM under attack conditions, and that decreasing interrupts will meet our goal of increasing CPU performance. UDP shows a very similar result, once again confirming that high CPU usage is tied to high packet rate and small packet size. From our results it is clear that any amount of buffering time improves CPU performance, but that the improvement levels off after a certain point. We chose a buffer time of 0.5 ms for the rest of our experiments, which is a reasonable amount of delay and results in considerably lower CPU overhead. It is important to note that the delay choice affects both sending and receiving; therefore the round trip time of packet and its response can be delayed by at most 1 ms based on our choice.

Taking our modified VirtIO driver with the buffer time at 0.5 ms, we re-ran the profiling tests and compared this modified KVM to the original KVM. As Table III shows our modifications considerably reduced all of our metrics. There was a 47% decrease in cache references, and a 23%

	KVM	Mod-KVM
LLC References	1,292,316,729	684,267,176
LLC Misses	36,060,083	27,684,718
Context Switches	708,335	422,471
IRQs/Second	37,000	25,500

TABLE III: Performance Metrics Under DoS – VirtIO Modifications

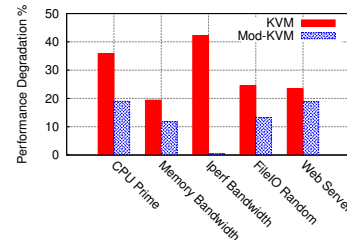


Fig. 6: Improvement In Performance with 0.5 ms Buffer Time

decrease in cache misses. This decrease occurs because the virtual machine is now using its cache more efficiently by processing more packets sequentially. Our modifications also reduced the number of context switches over 40% compared to the unmodified KVM, again because more packets are being processed at once.

We also re-ran the benchmarks from our previous experiment on our modified KVM virtual machine, again with the buffer time set to 0.5 ms. In Figure 6, we compare the performance degradation experienced by both KVM and our modified KVM under a DoS attack. Performance on all benchmarks was significantly improved by our modifications. In the CPU benchmark, our modifications reduces performance degradation from 35.8% to 19.0%; memory bandwidth no longer degrades by 19.4%, but instead by 11.8%. The modified drivers had the biggest impact on network bandwidth causing performance degradation to drop from 42.2% to less than 1%, effectively alleviating the problem; the file system benchmark improved from a degradation of 24.5% to 13.3%.

The modified KVM appeared to have the least improvement on the web server benchmark, improving from 23.4% degradation to 18.8%. However, a surprising result is found when we compare the actual number of connections per second: our modified KVM has a pre-attack baseline of 1,926 connections per second, whereas the original KVM has only 1,654 connections per second. This represents a considerable increase of 16.4% more connections per second. This is because our adaptive modifications to the VirtIO driver are activated even under the baseline conditions, meaning that combined traffic of the 200 simulated clients is greater than the threshold (10,000 packets per second). Therefore, while performance degradation

only decreases by a small margin with our modified KVM, the absolute number of requests per second is much higher than for the original KVM, in both the baseline and attack situations. With this improvement, KVM is now faster in this test than even Xen was in terms of both baseline and attack conditions. These surprising improvements to the baseline indicate that our modifications may have applications to situations beyond DoS attacks. Current version of our source code is available at www.sfu.ca/~rws1/mod-kvm/.

X. CONCLUSION AND FURTHER DISCUSSIONS

Despite the fact that modern virtualization has many attractive features, it is clear from our experiments that it is more vulnerable under TCP SYN DoS attack. In particular, although it has been known that virtualization entails higher overhead than traditional systems, our results suggest that even a light DoS attack can aggravate this overhead considerably. The combination of virtualization overhead and performance degradation in a DoS attack can lead to a 50% decrease in Web Server performance when compared to the non-virtualized Vanilla system using the same amount of resources. The performance implications of larger DoS attacks have yet to be quantified. We have performed some preliminary experiments and it appears that, at higher DoS rates, hypervisor-based VMs may actually become unresponsive to the network.

Although defenses such as SYN-cookies and SYN-caches have been very effective at fortifying non-virtualized systems, we showed through our experiments that these measures do not provide adequate protection for VMs. It is also important to note that even though the deployment of anti IP address spoofing technology such as ingress filtering (BCP 38) has reached the majority of autonomous systems, it is still estimated that up to 22.2% of autonomous systems allow spoofing¹; this gap in deployment means this form of attack continues to be a real threat to VMs. SYN-proxies are another technology that may provide an effective mitigation strategy for VMs, however due to the effectiveness of endpoint defenses for non-virtualized systems there are currently many networks on which this technology is not deployed.

As shown in Table I, any high packet rate with low packet size traffic pattern can cause performance issues for KVM and Xen. It remains to be discovered what effects similar non-DoS traffic have on real-world clouds such as the Xen-based Amazon EC2. Our initial trials using UDP Voice Over IP (VOIP) traffic indicate degradation will likely occur. As noted previously, our modifications to KVM actually greatly improved the baseline performance of KVM in our comprehensive benchmark. It is likely many different applications with high packet rates could benefit as well: examples include VOIP and DNS.

Recently hardware designers have also taken notice of the inherent overhead of virtualized networking and have produced some products to help alleviate the overhead. In particular, Single Root I/O Virtualization (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 an increase in throughput when run against paravirtualization drivers such as VirtIO [21]. The same research

however, indicates that CPU overhead of processing small packets remains largely unchanged. So although a future work should explore specialized hardware devices, it remains to be discovered whether they will remove a significant amount of the CPU overhead while the system is under attack.

Finally, it should be noted that 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 hypervisor-based virtualization is significantly more expensive to use on systems exposed to DoS traffic.

REFERENCES

- [1] CDW, "Cdw's server virtualization life cycle report, january 2010."
- [2] C. Newsroom, "Commvault releases results of annual end-user virtualization survey emphasizing the need for modern approach to protecting managing virtual server environments."
- [3] W. Vogels, "Beyond server consolidation," *Queue*, vol. 6, pp. 20–26, January 2008.
- [4] P. Padala, X. Zhu, Z. Wang, S. Singhal, and K. Shin, "Performance evaluation of virtualization technologies for server consolidation," *HP Labs Tec. Report*, 2007.
- [5] K. Ye, X. Jiang, S. Chen, D. Huang, and B. Wang, "Analyzing and modeling the performance in xen-based virtual cluster environment," in *2010 12th IEEE International Conference on High Performance Computing and Communications*, 2010, pp. 273–280.
- [6] R. Russell, "virtio: towards a de-facto standard for virtual i/o devices," *SIGOPS Oper. Syst. Rev.*, vol. 42, pp. 95–103, July 2008.
- [7] S. Soltész, 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, March 2007.
- [8] D. Nurmi, R. Wolski, C. Grzegorzczak, G. Obertelli, S. Soman, L. Youseff, and D. Zagorodnov, "The eucalyptus open-source cloud-computing system," in *Proc. of the 2009 9th IEEE/ACM CCGRID*, 2009.
- [9] W. Eddy, "Rfc 4987: Tcp syn flooding attacks and common mitigations," 2007.
- [10] H. Wang, C. Jin, and K. G. Shin, "Defense against spoofed ip traffic using hop-count filtering," *IEEE/ACM Trans. Netw.*, pp. 40–53, February 2007.
- [11] C. Jin, H. Wang, and K. Shin, "Hop-count filtering: an effective defense against spoofed ddos traffic," in *Proc. of the 10th ACM conference on Computer and communications security*, 2003, pp. 30–41.
- [12] W. Eddy, "Defenses against tcp syn flooding attacks," *Cisco Internet Protocol Journal*, vol. 8, no. 4, pp. 2–16, 2006.
- [13] D. Bernstein, "Syn cookies," 1996.
- [14] 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 Transactions on Parallel and Distributed Systems*, vol. 22, no. 6, pp. 931–945, 2011.
- [15] J. Ekanayake and G. Fox, "High performance parallel computing with clouds and cloud technologies," *Cloud Computing*, pp. 20–38, 2010.
- [16] 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 *2nd IEEE International Conference on Cloud Computing Technology and Science*, 2010, pp. 159–168.
- [17] 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. of the 2007 Workshop on Experimental Computer Science*, 2007.
- [18] 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 Computing*, pp. 115–131, 2010.
- [19] G. Wang and T. Ng, "The impact of virtualization on network performance of amazon ec2 data center," in *INFOCOM, 2010 Proc. IEEE*, 2010, pp. 1–9.
- [20] J.-W. Jang, E. Seo, H. Jo, and J.-S. Kim, "A low-overhead networking mechanism for virtualized high-performance computing systems," *The Journal of Supercomputing*, pp. 1–26.
- [21] J. Liu, "Evaluating standard-based self-virtualizing devices: A performance study on 10 gbe nics with sr-iov support," in *Proc. of IPDPS 2010*.

¹October 2011 estimate from the MIT Spoofer Project <http://spoofer.csail.mit.edu/>