# Virtual Machine Effects on Network Traffic Dynamics

J. Martin V. Rajasekaran J. Westall

Department of Computer Science
Clemson University
Clemson, South Carolina 29634-0974

## Abstract

*Although virtual machine performance has been widely studied in the context of CPU utilization, the effects of virtual machines on network traffic dynamics has received far less attention. In this study, using VMware's GSX server on a Linux host, we evaluate the impact of VM overhead on TCP performance in both a LAN and emulated WAN evironment. It is shown that when multiple VM hosted TCP senders compete on a LAN, sustained aggregate throughput decreases significantly and that ack arrival distributions are strongly affected. In the emulated WAN environment, TCP is shown to exhibit increasingly bursty behavior with associated increases in loss rate as the number of virtual Web servers increases.*

## 1 Introduction

A virtual machine facility (VM) permits a single computer system to emulate multiple computer systems. IBM pioneered the commercial development of the VM facility in the 1960s. Their VM/370 platform supported OS/360, OS/370 and CMS operating systems. Renewed interest in VM systems has been generated in the last five years by the emergence of systems such as VMware[1], the VirtualPC product developed by Connectix and now owned and marketed by Microsoft[4], and User Mode Linux[2]. These systems run on commodity personal computers and provide an environment that allows under-utilized servers to be virtualized onto a smaller number of physical machines thereby reducing acquisition and management costs.

### 1.1 Benefits of VM technology

In addition to cost savings, the VM environment provides potential benefits in software development and testing, security, and in the deployment of the utility computing paradigm. Just as the Java virtual machine "sandbox" concept isolates Java applications from each other and from the host computer [7], the VM environment provides strong isolation allowing a single server to run untrusted applications. Because a rogue or defective application must breach two levels of security, VMs are generally viewed as more secure than a conventional multitasking operating system environment [5]. Since virtual machines are readily customized and can be dynamically created, suspended, and restarted, the VM environment is well aligned with the utility computing paradigm in which services and resources are requested and utilized on demand and then released when no longer needed [6].

### 1.2 VM performance issues

Performance is a recurring concern in every application of virtual machine technology. The layers of abstraction inherent in a VM can add significant CPU overhead. Consequently, reduction of CPU overhead has been a consistent theme of VM related research, and recent performance studies [6, 10, 11] report that the increase in kernel mode CPU time consumed on behalf of a typical application in the VM environment can now be limited to 50%.

Perhaps because of the obvious significance of the CPU overhead problem, the impact of the VM environment on network performance has received little attention in the research literature. Based upon insights obtained in previous studies of TCP dynamics we suspected that characteristics of the VM environment could, under heavy loads, have a significant and negative impact upon TCP performance. This impact arises in two important ways.

First, all obvious approaches to managing multiple competing virtual machines engaged in bulk TCP transfer have clear disadvantages. On a 100 Mbps network the time required to send a single full-sized segment is 0.12 msec, and an *ack* is normally received for every other segment. If the VMM attempts to provide a reasonable approximation of round-robin scheduling

[1] http://www.vmware.com
[2] http://user-mode-linux.sourceforge.net

of outgoing frames among multiple competing VMs, it is necessary for the VMM to context switch among them at a rate of about 4000 switches/second. We will show in the next section that VMWare uses this general approach, and that when four or more competing VMs are active, even a modern dual processor system can sustain at most 60 Mbps aggregate throughput on a 100 Mbps channel.

Second, even when only a single bulk transfer is being performed, performance problems related to *ack* compression can arise. TCP maintains a window of outgoing full size segments that are pre-buffered in kernel space. When an arriving *ack* allows one or more full size segments to be sent, one or more pre-buffered segments can be transmitted almost immediately either in the context of the hardware interrupt associated with the receipt of the *ack* or (in the case of Linux) very shortly thereafter by a high priority kernel thread. In this way, TCP's *ack* clocking causes segments to be transmitted at the correct rate as determined by network properties.

However, in a VM environment the delivery of the *ack* does not occur until the target virtual machine can be dispatched. Under heavy loads with multiple competing VMs the dispatch can be significantly delayed. Such delays not only contribute to the loss of aggregate throughput cited above, but when the VM is finally dispatched, the delivery of multiple *acks* in a burst produces a burst of transmissions. This phenomenon is known as *ack* compression [8], and it is well-known to be detrimental to TCP performance[1].

This paper reports upon two studies of these phenomena. In the first study, the behavior of always-on TCP workloads are used evaluate the effect of context switching on sustainable throughput in a LAN environment and to gain basic insights on the interaction of the VM environment with the dynamics of *ack* arrivals.

In the second study, using a network testbed to emulate Web traffic on a WAN, we examine the effects of the VM environment on TCP dynamics under realistic traffic loads. The *Surge* [2] traffic generator provides realistic Web traffic loads, and the *dummynet* facility [9] controls bandwidth and latency within the emulated WAN. VMwares's GSX server is used to host Linux kernels on which Apache Web servers run. We show that as load and the number of VM hosted Web servers increase, substantial increases in both TCP burstiness and the packet loss rate become evident. At high loads, the effect is significant enough to cause a doubling of response time at the application level.

The remainder of the paper is organized as follows. In section 2 we present the results of a study of competing always-on TCP connections in a VMWare LAN environment. The testbed used in the WAN emulation study is described in section 3, and the results of that study follow in section 4. We conclude with a summary of what has been learned and identify possible directions for future research.

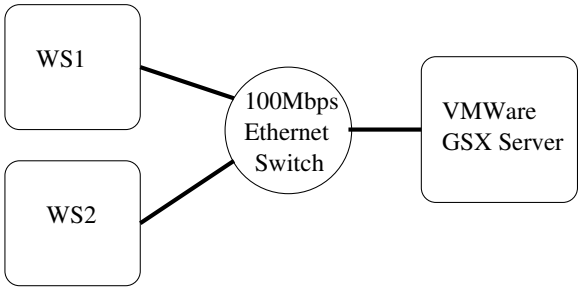


Figure 1: Network testbed

## 2 VM effects on always-on TCP connections

In this section we show how the scheduling policy used in the VMWare GSX Server version 2.5.1 affects both the maximum aggregate throughput and the timing of *ack* arrival when multiple VMs host competing always-on TCP connections. The VMware GSX Server version 2.5.1 provides the virtual machine environment. This system requires an underlying host operating system and an unmodified Linux 2.4.18 kernel fills this role. The VMWare layer intercepts system calls made by applications running on a guest OS and, through a combination of emulation and virtualization, presents the virtual PCs that it hosts with a set of basic I/O devices. Special drivers enhance performance by optimizing the interaction between the guest OS, the host OS, and the system hardware. All guest VMs reported upon in this section and in the emulated WAN study are configured with 256MB of virtual RAM and running a Linux 2.4.18 kernel.

### 2.1 Network configuration

The network testbed is configured as shown in figure 1. The machine labeled VMWare GSX Server is a Dell Precision 450n workstation equipped with 2 GBytes of RAM and dual Intel Xeon processors running at 2.4GHz. The machines WS1 and WS2 are identical Dell Optiplex GX250 PCs equipped with 512 MBytes of RAM and an Intel Pentium 4 processor running at 2.4GHz.

In these studies a variable number {1, 4, and 7 } of always-on TCP senders run concurrently on the VMWare GSX Server. These processes send 250

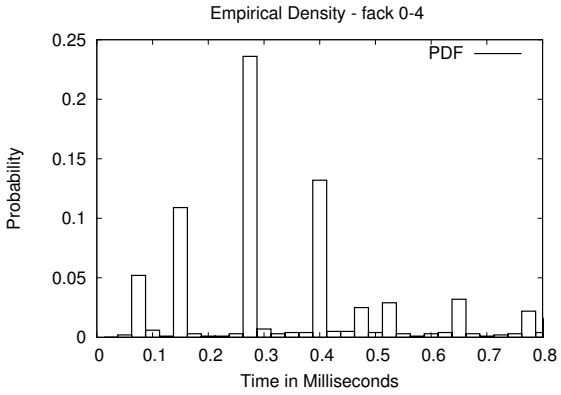


Figure 2: Four senders: *ack* arrivals

megabytes of data in full-sized segments from a memory resident buffer to receiver processes running on WS1 and WS2. Since no disk access is used at either endpoint, the aggregate transfer rate is limited only by the speed of the 100 Mbps link.

A custom, low-overhead monitor called *tcpmon* performs the data capture. It uses Linux *netfilter* hooks to inspect individual packets at the boundary between the IP layer and the Linux *dev* layer. For each study, a single TCP connection is monitored, and the arrival and departure times of each packet belonging to that connection are captured by reading the hardware cycle counter. Although the cycle counter is virtualized by VMWare[3] it has been found in testing to provide repeatable timings on a virtual machine at overhead of only 800 cycles.

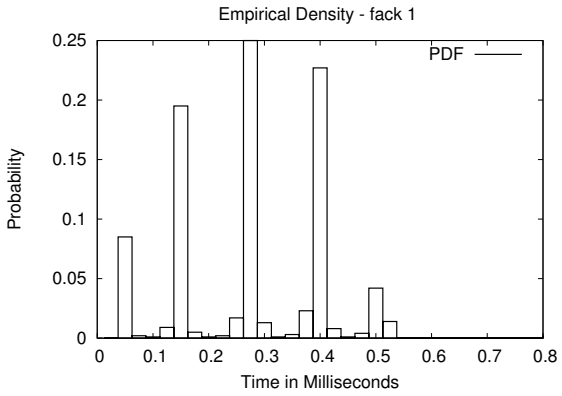


Figure 3: One sender: *ack* arrivals at host Linux

[3]Private communication from VMWare engineers.

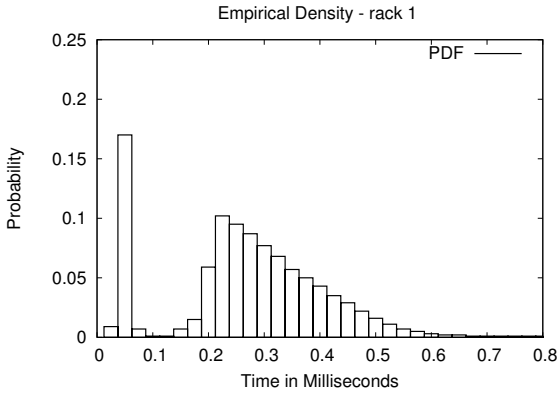


Figure 4: One sender: *ack* arrivals at VM

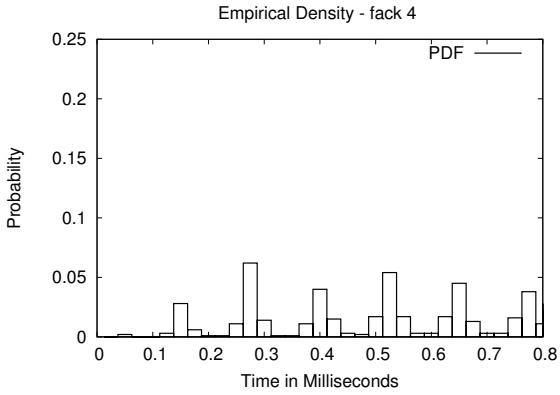


Figure 5: Four senders: *ack* arrivals at host Linux

## 2.2 Baseline *ack* interarrival densities

To establish a baseline for comparison, the TCP senders were first run on the *host* Linux. No VMs were running during these tests. Observed aggregate throughput was approximately 95Mbps for all tests. The interarrival density of one the *ack* streams when four concurrent senders were active is shown in figure 2. Each bar has a width of 0.025 ms. Bars are centered at (0.0125, 0.0375, ...) msec. The time to send a full segment is approximately 0.125 msec. Therefore, the leftmost bar in the graph represents *acks* that have been compressed in time. The remaining density is highly modal with the modes occurring in multiples of segment transmit time.

Since TCP normally returns an *ack* for every other segment received, it is not surprising that the mode which occurs at two segment times is the largest. The two leftmost bars can be attributed in part to both $quick - acks$ in Linux wherein an *ack* is sent for every TCP segment and to expiration of the delayed *ack*

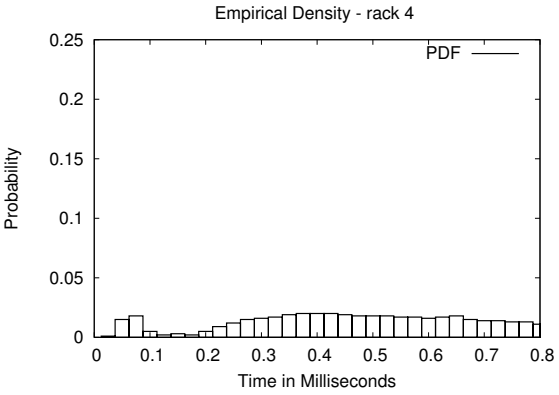


Figure 6: Four senders: *ack* arrivals at VM

timer before a second segment is received. As more senders compete for the single channel, modes of significant size appear at larger and larger multiples of segment time. Measured mean *ack* interarrival times for the monitored connection were 0.256, 1.072, and 1.752 msec for 1, 4, and 7 concurrent connections.

### 2.3 *Ack* interarrival densities in the VM environment

We now show that virtualization has considerable impact on both the achievable throughput and on the arrival distribution on the *ack* stream seen by both host and guest operating system. For these experiments, each TCP sender runs on a separate Linux VM. Aggregate throughput remains at 95Mbps on the dual processor system with a single active VM. However, when the number of VMs exceeds the number of real processors, performance degrades as context switches begin to delay *ack* delivery and the sending of new segments. Aggregate throughput drops to about 60 Mbps when both four and seven VMs are active.

The fact that identical aggregate throughput is achieved with both four and seven VMs is to be expected because the aggregate number of segments sent per unit time is coupled to the context switching rate and not the number of VMs in the round-robin queue. The fact that there is no major decline in aggregate throughput between four and seven VMs also demonstrates that the 2GB of real memory is sufficient to run all seven without inducing paging.

Figure 3 and 5 shows the *ack* interarrival density as seen at the Linux host. The distribution shown in figure 3 is virtually identical to its baseline counterpart, but for four VMs significant differences are evident. Although these distributions remains modal, the large mode at 2 segment times has disappeared. The distribution shown in figure 5 also reveals the scheduling policy is not perfectly round-robin. For four senders a truly round-robin schedule would produce a large mode at eight packet times. A random scheduling model better fits the observed data, but it is clear that scheduling at packet time granularity is taking place.

Figures 4 and 6 show the interarrival densities of the *ack* stream at the guest Linux. With only one traffic source the guest sees approximately twice as many compressed *acks*, but the large mode at 0.25 msec in the host distribution is smeared at the guest and the other modes have disappeared. With four and seven active VM's, the *ack* arrival process assumes a nearly uniform appearance with no evidence of significant *ack* compression.

In summary, we have shown that the VMWare environment affects always-on TCP senders in major ways. For the system studied, context switching overhead with four or more active VMs limits throughput on a 100 Mbps link to 60% of capacity. Since this effect is not related to link speed, it has even stronger negative implications regarding the efficient use of gigabit speed links.

The TCP *ack* stream is also clearly perturbed. *Acks* that arrive for a VM guest while it is waiting to be dispatched are delivered in a batch. This batching behavior doubled the level of *ack* compression with one active VM. active. However, *acks* that are not batched are subject to random delay whose magnitude depends on the system load and produces an approximately uniform distribution of interarrival times.

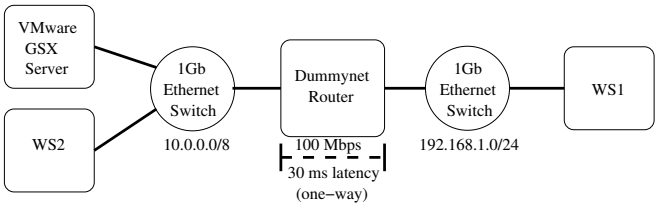


Figure 7: Network testbed

## 3 Emulated WAN configuration

We next consider VM effects in an environment that is more reflective of real-world traffic than that of the preceding study. The network testbed used in the emulated WAN study is shown in figure 7. The VMware GSX Server on the 10.0.0.0/8 network acts as a farm of Web servers in which each VM hosts exactly one instance of the Apache Web server. A virtual bridged network within the VMware host provides each virtual Web server with a unique 10.0.0.0/8 address that is visible to the outside network. The machine hosting

the VMware GSX server is the same physical machine used in the always-on study.

The machine labeled Dummynet Router is a Dell GX 250 with 512 MB of RAM and a single 2.4 GHz Pentium 4 processor. It runs FreeBSD 4.8 and uses *dummynet*[9] to control path latency and bandwidth. For these studies, *dummynet* was configured as two unidirectional pipes with a bandwidth of 100 Mbps and a latency of 30ms. The capacity of the *dummynet* queues was set to 40 packets each.

### 3.1 Test workloads

The system labeled WS1 is physically identical to the server system and WS2 is physically identical to the router system. Both run the *Surge*[2] client software on Red Hat Linux 8.0 to simulate the behavior of a large number of Web clients. As is shown in the figure, only the traffic generated by WS1 traverses the simulated WAN.

For the studies reported upon here, the number of simulated Web clients on WS1 takes on the values {100, 200, 300, 400 and 600} and the number of clients on WS2 takes on the values {100, 200, 300, 400 and 300}. The asymmetric assignment with 900 total clients was necessary to provide increased traffic over the simulated WAN when the 400/400 workload was found to consume 100% of both server CPUs with 7 VM hosted servers.

The number of VMs used is {0, 2, 4, and 7}. For each of the twenty possible combinations of Web clients and virtual servers, the values reported are obtained from a run of approximately ten minutes of elapsed real time. The system is configured to equally distribute the load among the VM hosted servers. Because of the stochastic behavior of the *Surge* clients and the relatively short run lengths the actual loadings are only approximately equal.

## 4 Results and Analysis

In this section we present empirical evidence that under heavy loads the burstiness and packet loss rate of TCP connections and consequently the mean response time of *http* transactions tend to increase as the number VM hosted servers increases.

### 4.1 Evaluation of burstiness

It was conjectured in the introduction that the use of VM hosted servers might introduce bursty TCP behavior whose severity would increase as a function of both load and the number of servers. Absent VM effects, one would expect a TCP session in equilibrium not to experience large variations in bytes transferred per RTT. Therefore, we use 60ms as a baseline RTT and characterize the burstiness of a TCP session in the following way. Let $\{t_i\}$ = the number of bytes acknowledged during the $i^{th}$ 60ms subinterval of the TCP connection and $\mu_t$ and $\sigma_t$ represent the mean and standard deviation of the $\{t_i\}$. We characterize the burstiness of the session as $\sigma_t/\mu_t$, the coefficient of variation of the series $\{t_i\}$. Burstiness values by load level and number of VM hosted servers are shown in figure 8. Each point represents the mean value of $\sigma_t/\mu_t$ for the 5 largest downloads that occurred for the corresponding number of VM hosted servers and load level.

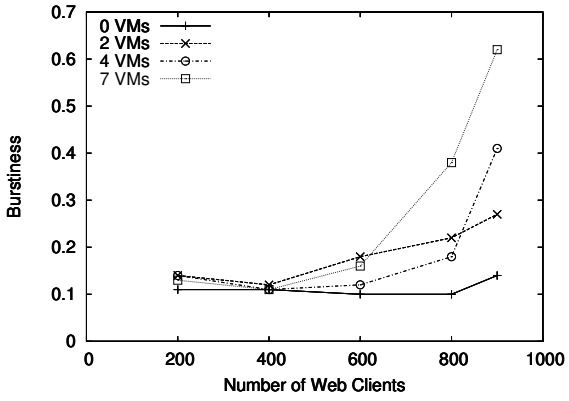


Figure 8: Server burstiness

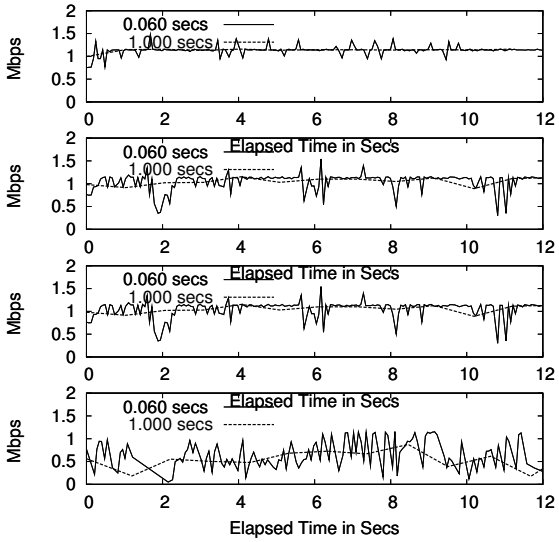


Figure 9: Single session throughput

This effect is also illustrated in the four graphs of figure 9 that show the observed throughput of individual long-running TCP sessions with 0, 2, 4, and 7 VM hosted servers respectively and a workload of 800 *Surge* clients. The line labeled 0.060 shows throughput in Mbps as described above on a 60ms scale during

a 10 second interval. The much smoother line labeled 1.000 secs shows observed throughput in Mbps on a scale of 1 second. The first of these graphs shows that with no VM hosted servers the short-term throughput converges rapidly to the long-term throughput and exhibits only minor excursions thereafter. All of the other graphs show continuing excursions from the mean. The frequency of the excursions also increases with the number of VM hosted servers. Furthermore, it was found that the Gigabit network regularly delivered bursts of packets to the router at bit rates exceeding 200 Mbps when sampled at millisecond resolution with 7 VM hosted servers in use.

### 4.2 Loss rate and response time

The packet loss rate was insignificant until the number of surge clients reached 800. Under heavy loads the loss rates become ordered by the number of VM hosted servers. Furthermore, the loss rate of 5.7% observed with 7 VM servers at a load of 80 Mbps is more than 40 times the loss rate of 0.13 obtained with no VM hosted servers at a higher load of 86 Mbps. The loss rate of 0.13 at 86% load on the bottleneck link might seem surprisingly low, but it is consistent with the results reported in [3]

## 5 Conclusion

In section 2 we showed that when multiple always-on TCP session compete for bandwidth in a LAN environment, the result is that packets from each VM are interleaved in a fine-grained but quasi-random way. Delays associated with context switching overhead produced *ack* compression when only a single VM was active and loss of aggregate throughput when more than one VM was active.

The results of the emulated WAN study provides qualitative evidence that the use of VM hosted Web servers can negatively affect TCP performance producing bursty behavior increased packet loss. The evidence suggests that the effect is magnified as the number of VM hosted servers is increased. In ongoing work we are repeating these studies using more replications of tests of longer duration to better quantify the effects in statistically robust ways.

One disadvantage of the use of VMWare is that its closed architecture makes it not possible to definitively identify what aspects of its design produce the effects we observe. To more definitively connect cause and effect, we are presently repeating these studies with the open source User Mode Linux system.

## Acknowledgment

This work was supported in part by the IUCRC program of the National Science Foundation under award EEC-0116924.

## References

[1] Hari Balakrishnan, Venkata N. Padmanabhan, Srinivasan Seshan, Mark Stemm, and Randy H. Katz. TCP behavior of a busy internet server: Analysis and improvements. In *IEEE INFOCOM98 (1)*, pages 252–262, 1998.

[2] Paul Barford and Mark Crovella. Generating representative Web workloads for network and server performance evaluation. In *Proceedings of Performance '98 / ACM SIGMETRICS '98*, pages 151–160, Madison, WI, July 1998.

[3] Mikkel Christiansen, Kevin Jeffay, David Ott, and F. Donelson Smith. Tuning RED for Web traffic. *IEEE/ACM Transactions on Networking (TON)*, 9(3):249–264, 2001.

[4] J. Dalrymple. Microsoft acquires VirtualPC from Connectix. *InfoWorld*, February 2003.

[5] George W. Dunlap, Samuel T. King, Sukru Cinar, Murtaza A. Basrai, and Peter M. Chen. Revirt: Enabling intrusion analysis through virtual-machine logging and replay. *SIGOPS Operating Systems Review*, 36(SI):211–224, 2002.

[6] R. Figueiredo, P. Dinda, and J. Fortes. A case for grid computing on virtual machines. In *Proc. International Conference on Distributed Computing Systems (ICDCS)*, May 2003.

[7] Gary McGraw and Edward Fellen. *Java Security: Hostile Applets, Holes, and Antidotes*. John Wiley & Sons, Inc., New York, NY, 1997.

[8] J. Mogul. Observing TCP dynamics in real networks. In *ACM SIGCOMM '92*, pages 305–317, Baltimore, MD, Oct 1992.

[9] Luigi Rizzo. Dummynet: a simple approach to the evaluation of network protocols. *ACM Computer Communication Review*, 27(1):31–41, 1997.

[10] Peter M. Chen Samuel T. King, George W. Dunlap. Operating system support for virtual machines. In *Proceedings of the 2003 Annual USENIX Technical Conference*, June 2003.

[11] J. Sugerman, G. Venkitachalam, and B. Lim. Virtualizing I/O devices on VMware workstation's hosted virtual machine monitor. In *Proceedings of the USENIX Technical Conference*, June 2001.