Paper title: Virtualizing I/O Devices on VMware Workstation's Hosted Virtual Machine Monitor
Paper authors: Jeremy Sugerman, Ganesh Venkitachalam and Beng-Hong Lim
Your name: Parth Kansara (115135130)

What problem does the paper address? How does it relate to and improve upon previous work in its domain?
The paper looks at the domain of virtualizing the I/O devices for PCs, as compared to the existing systems which virtualized the mainframe-class machines. Since there is a high diversity in the kind of hardware available on PCs and the host OS is usually pre-installed, it is a challenge to implement virtualization without modifying either the guest or the host OS. Further, the Intel processor architecture is not virtualizable, since it doesn't allow the user to execute the guest instructions in a less privileged mode.

Gum describes the various hardware optimizations in the IBM System/370 architecture that focused on reducing the overhead of dealing with privileged guest instructions, guest memory address translation and multi-processing support. **Borden et al.** explains PR/SM which allows IBM 3090 series of mainframes to support VMs with dedicated I/O devices and achieve close to native performance.

Hall and Robinson elaborate their approach towards virtualizing the VAX architecture, which is not virtualizable by default. They describe the various hardware modifications done to the VAX architecture, which interferes with the existing system unlike VMware's technology. **Bugnion et al.** presents the design for a VMM for large scale NUMA machines, which can mask their nature and allow for running commodity OS that are meant for smaller scale systems. **Mach** presents the idea of microkernel-based operating systems based on emulation of OS APIs, similar to the VMware Workstation's emulation of I/O devices. **Hartig et al.** illustrates the methods for improving the performance of microkernel-based systems.

What are the key contributions of the paper?
The paper offers an insight into the VMware Workstation's method for virtualizing the I/O devices. To avoid interfering with the pre-installed OS, it hosts a user-level application component VMApp, along with scheduling a privileged VMM component. VMM virtualizes the CPU while the VMApp virtualizes the I/O devices. Once these components are installed, the physical processor is either executing in the host world or the VMM world, switching using the VMDriver. Any I/O operation by the guest OS is intercepted by the VMM and it switches to the host world to allow the VMApp to perform the I/O. This switching between the host world and the guest world results in a significant overhead. Additionally, the host OS can page out the allocated memory of a VM which can affect the performance. The VMM traps the IA-32 IN and OUT instructions, which are used for I/O operations and deal with the hardware accesses.

A network interface card (NIC) is an example of a hardware that requires high throughput and low latency. Emulating this can be connected to the host by bridging it to the same physical network or it can be connected to a virtual network created on the host. For sending and receiving the packets, intermediate I/O accesses and instructions for handling a virtual IRQ are dealt with by switching between the two worlds. The overhead and degradation in performance for the same can be dealt with using the following optimizations - dealing with modification of the data ports in the VMM instead of performing a world switch, accumulating and delaying the transmission of the packet till a world switch occurs, replacing the system calls for an IRQ notification with a bitvector. Apart from this, future optimizations can also look at reducing the CPU virtualization overheads, modifying the guest OS and the host OS, optimizing the guest driver protocol along with accessing the hardware directly through the VMM.

Briefly describe how the paper's experimental methodology supports the paper's conclusions.
The paper looks at the virtualization of the NIC and evaluates its performance using VMware Workstation. The hosted architecture results in 30.65 µs out of the 31.63 µs to be spent in world switching. However, after the applied optimizations, the TCP transmit throughput was doubled on a CPU bound machine, and reduced the CPU utilization significantly on a I/O bound machine. This highlights the efficiency of the hosted architecture of the VMware Workstation in providing virtualization of the I/O devices while not requiring any modifications on the guest OS or the host OS.

Write down one question you may want to ask the authors.
Are there any other hardware components like the NIC that might require special optimizations to be able to effectively run the VMware Workstation?