NSF-CAREER
Introspective Computing : A Multicore Approach to Availability, Reliability and Security
2010 Annual Report

School of Electrical and Computer Engineering
Georgia Institute of Technology, Atlanta, GA 30332

Principal Investigator: Hsien-Hsin S. Lee
leehs@gatech.edu Tel: (404) 894-9483 Fax: (404) 385-3137

1 Introduction

Datacenters are deploying a growing range of "bumpin- the-wire" services that perform network packet processing transparently to applications. Deep Packet Inspection (DPI) in particular has high market growth, for services like intrusion detection, content insertion, performance monitoring, traffic classification and flow management. Packet processing services are usually provided using central deployment of custom hardware appliances that guard the datacenter boundary. By concentrating all traffic processing at centralized appliances, these solutions face performance requirements that often can be met only by using special-purpose acceleration hardware. Thus, centralized appliances can be expensive, difficult to scale incrementally, and inflexible due to hardwired functionalities. As we enter the era of multicore processors with a large number of inexpensive cores, some cores can be used in privileged partition while others are in unprivileged partition. Although the virtualization approach can be used to provide isolation between privileged partition and unprivileged partition, the growing size of the hypervisor code base makes the privielged partition vulnerable. Our approach seeks to use "hardware partitioning" in order to provide efficiency, security and transparency. The privileged partition can access all of the system's physical memory while the unprivileged partition is restricted to a subsetted region. Completely separate software stacks can run in parallel in the two partitions. For example, the privileged partition could boot an OS and run packet processing services in user level, while the unprivileged partition executes an entirely separate software stack including its own OS and user applications.
Figure **??** shows our baseline which is a simplified block diagram of a generic near-future platform architecture, built around a multicore processor with many cores sharing a shared Last Level Cache (LLC).

2 Research Activities

In the first year of invetigating the use of hardware partitioning, our goal is to provide better isolation between privileged and unprivileged partitions. The first challenge is to . Sequestering privileged cores is essential for providing isolation, since unprivileged cores are prevented from using privileged cores. In order to sequester cores at boot time, the BIOS needs to be modified. We modified the BIOS of QEMU, a full system simulator, to test the possibility of booting cores separately. The second challenge was to secure the memory region which is used by privileged cores. The corrupted OS cannot affect the software running in the prvileged partition. This guarantees the potentially vulerable software running in the unprivileged partition, such as hypervisor, operating system, will not compromise the privileged software (Deep Packet Inspection in our case). The third challenge was to intercept the packets to/from the host. Note that the privileged software is agnostic to and untrust the software running in the unprivileged partition. Therefore, it cannot rely on any software in the unprivileged partition. The solutions to all above challenges need to satisfy the transparency requirement, which means the software in privileged partition needs to be transparent to the software in unprivileged partition. We have successfully built our prototype system "Ally" in QEMU, which models a

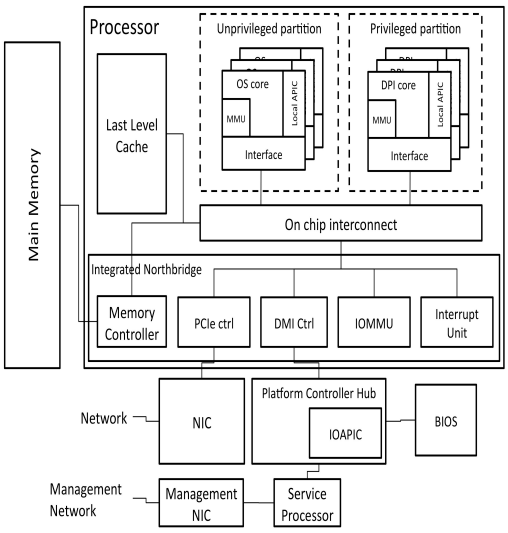


Figure 1: Target platform architecture

dual-core system. The prototype includes a packet analyzer running on a sequestered core while the other core boots Linux. The Linux is unaware of the existence of the privileged core as well as the packet analyzer.

3 Major Findings

We describe the design of our prototype system "Ally" as following.

3.1 Core Sequestration

Core sequestration enables distinct software environments to be booted on different CPU cores. In the conventional booting process of BIOS, a core (called BSP core) wakes up the other cores (called AP cores). Each AP core runs code loaded from BIOS which executes a self-test and enters the core's unique identifier (Local APIC ID) into the ACPI's Multiple APIC Description Table (MADT) and the MP configuration table. Normally the OS uses the MADT to and all information needed to discover and communicate with a core, but the MP table will be used instead when the MADT is not present due to ACPI disabled. After initialization, each AP core halts and waits for an Inter-Processor Interrupt (IPI) to resume execution.

Ally leverages this boot process to launch the DPI environment on dedicated cores (DPI cores) and conceal them from software on the other cores (OS cores). Our modified BIOS selects some AP cores as DPI cores. The DPI cores run a different initialization procedure which deletes their corresponding entries in the MADT and the MP configuration table rendering DPI cores invisible to OS cores. The DPI cores then load the cus- tom Interrupt Descriptor Table and initialize the DPI application.

3.2 Memory Protection

Ally hardware extensions protect DPI cores from OS cores' software. The physical memory address space is split in two contiguous regions, in which one region is accessible only to DPI cores. To prevent OS cores from accessing DPI memory, on each hardware page table walk the processor validates the resulting physical address. Ally uses a modified TLB Miss Handler (TMH), shown in Figure **??**, which verifies that any physical address to be loaded in the TLB on an OS core is not in the range of DPI memory. Ally adds a memory-mapped I/O (MMIO) configuration register to the MMU for each core that stores the boundary address between DPI and OS memory. The miss handler compares the final translated physical addresses to this register value to determine which memory region is the access target. As this operation is just one step of a time-consuming page table walk and requires only a comparison as opposed to an additional level of

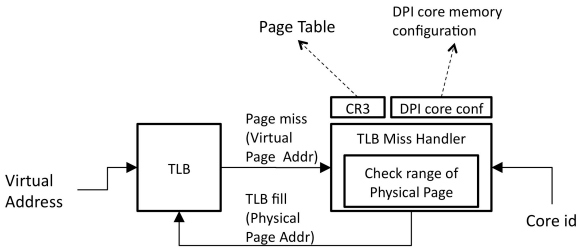
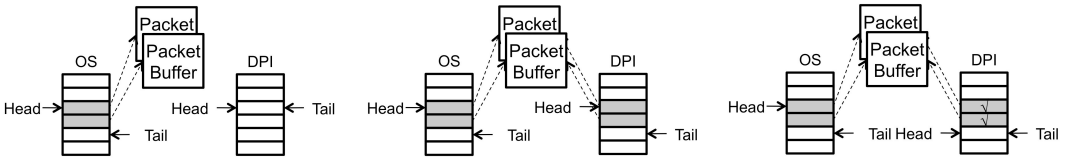


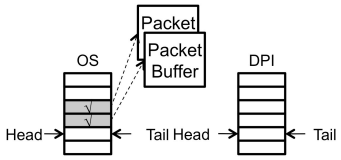
Figure 2: Modified Memory Management Unit.



(a) Descriptors are made available by the OS in the OS descriptor queue. OS increases OS Tail Pointer

(b) DPI gets interrupted by the OS core, copies descriptors, and updates DPI Tail Pointer

(c) NIC copies received packets and descriptors, marking descriptors as completed in the DPI queue. NIC updates DPI Head Pointer. **DPI processes packets**



(d) DPI marks descriptors in the OS queue as completed. DPI updates OS Head Pointer. The OS can thus consume the packets and clean the OS queue

Figure 3: Receive: Evolution of the descriptor queues in Ally

translation, this simple range check has negligible impact on performance. Similarly, Ally extends the I/O Memory Management Unit (IOMMU) to prevent non-authorized devices from performing DMA writes to DPI memory. Further mechanisms are provided to protect DPI memory in x86 real address mode (usually only needed early in the boot process).

3.3 Packet Interception

Ally modifies the Interrupt Unit in the Northbridge to redirect interrupts that are generated by the NIC so that they are steered to a DPI core instead of to an OS core. This is independent of whether the interrupts are routed through the chipset IOAPIC or they are directed to the cores via MSI. In both cases they are predecoded in the on-chip Northbridge's Interrupt Unit, which can thus identify which interrupts must be redirected and deliver them to a core from the DPI partition. The DPI partition uses Inter-Processor Interrupts (IPIs) to mimic NIC interrupts back to the OS. In addition to interrupt redirection, any write or read by OS cores to the NIC MMIO registers related to the descriptor queue manipulation must be intercepted and redirected. These include the registers pointing to the descriptor queues and the Tail/Head registers of each queue. In addition, interrupt-related registers need to be intercepted, in order to allow the DPI cores to control the interrupt rate based on the packet processing speed and to allow the OS cores to transparently use techniques like Linux NAPI which disable NIC interrupts and switch to polling mode under high traffic

situations for saving interrupt processing cycles. OS cores cannot disable NIC interrupts, because all their accesses to the NIC register used for interrupt masking are intercepted. Instead, DPI cores are responsible for throttling IPIs to OS cores when they see OS cores try to mask NIC interrupts.

Figure **??** illustrates the operation of the descriptor queues on packet reception. The OS preallocates descriptors in the OS queue. The DPI copies the descriptors to the DPI queue and updates the DPI Tail Pointer, which is visible to the NIC, thus notifying the NIC that descriptors are available for the incoming packets. The NIC copies the received packets, completes the descriptors in the DPI queue, and updates the DPI Head Pointer. At this point the DPI processes the received packets. To allow the OS to consume the received descriptors, the DPI marks the descriptors as complete in the OS queue and updates the OS Head Pointer. Finally, the DPI sends an IPI to an OS core to notify it that reception is complete. The OS can thus proceed to consume the received packets.

The interception of packet transmission is similar to the interception of packet receive. First, the OS posts descriptors in its transmit queue and advances the Tail Pointer. Ally intercepts the write to the Tail Pointer which is not propagated to the NIC, but to the copy kept in DPI memory, i.e., the OS Tail Pointer. The OS cores MMU raises an IPI to a DPI core, enabling the DPI core to detect that the Tail Pointer has been updated. The DPI copies the newly posted descriptors from the OS queue to the DPI queue. It processes the packets referenced by the descriptors, and updates the DPI Tail Pointer. This Tail Pointer is visible to the NIC, which then fetches the descriptors from the DPI queue and also the corresponding packets. After transmission is complete, the DPI core can mark the descriptors complete in the OS queue and provide notification to the OS core through an IPI.

4 Contribution

We demonstrate the possibilty of using hardware partitioning to provide isolated, efficient DPI integrated server platform. Compared to DPI integration through virtualization, hardware partitioning does not depend on any hypervisor or other software to provide isolation. Instead, through simple and low overhead hardware modifications, Ally can achieve strong isolation and better performance, which are very important properties of datacenter management services. Compared to centralized DPI appliances, Ally leverages the cheap and abundant computing resources in the future many core system to provide scalable and flexible packet processing services.

5 Future Work

We are currently investigating serveral possible extensions to Ally. We plan to extend this work to support and evaluate packet processing services using emerging multi-context NICs (PCIe SR-IOV NICs) [23]. Given the low cost of queue virtualization observed in our current prototype, it is highly likely that the cost of Ally's interception mechanisms will be a function dominated by the the link bandwidth or packet rate, and not by number of contexts. In addition, we plan to evaluate Ally for larger-scale multicore systems. Ally should easily leverage future processors equipped with several tens of cores by distributing the packet inspection engine. Last but not least, we plan to investigate other privileged applications besides packet processing services. Although we used packet processing service as an example of privileged application, Ally should be easily applied to other types of I/O devices, such as hard disk, given the similar nature of PCI devices.