Neel Shah **CSCI 4243w**
Design Document 1 **12/09/2016**

Purpose:

The purpose of the proposed project, Software-Defined Rack Scale Architecture (RSA), is to provide a software defined means to implement RSA to computers. RSA was originally conceived by Intel who fabricated hardware components to enable one operating system to use the hardware resources of many different physical systems in a server rack. Intel's RSA hardware is expensive, and it requires many physical changes to existing server rack configurations. This project, a software-defined approach to RSA, brings the foundations to eliminate the need for extensive hardware reconfigurations and also decreases the costs to provide, and maintain RSA in a server room. Through the use of intelligent memory swapping algorithms, this project will determine when to swap pages of memory between the host of the operating system and a remote memory server. This project requires the Xen hypervisor to host the target operating system since it coupled with LibVMI, a virtual machine introspection library, enables easy detection and management of an operating system's memory. The net goal is to provide an interface that enables transparent memory page swapping between the host of the operating system and a remote memory server host.

Users:

This project will provide use to systems administrators working in a data center. A data center administrator wishing to adopt RSA in their racks, but are limited by finances or physically refactoring their architecture would benefit. Software-Defined RSA will not prove to be useful to a broader audience of developers since it requires access to a specific set of hardware-data center servers and racks. Furthermore, this project can provide use to systems/virtualization enthusiasts since it provides a novel memory swapping algorithm. Since Software-Defined RSA will be kept open source, any interested party can provide improvements to the platform. Furthermore, the memory swapping algorithm can be used in different contexts, one being providing new algorithms of virtual machine memory paging.

Use Cases:

Software-Defined RSA will be used for providing a software-defined approach to obtain Intel's RSA. This architecture provides a means to enable an operating system to have a greater amount of hardware resources than its physical host alone. In the scope of memory, the RAM that the operating system has is not limited to the physical chip installed on the operating system's host, the operating system sees all the RAM across each host in the rack as one giant, contiguous memory region. This project can also impact the growing Internet of Things (IoT) movement. Servers that IoT devices communicate with tend to do one, trivial, task but serve requests to many devices. The volume of devices communicating with servers requires the servers to have more hardware. This project can eliminate the work required by data center system administrators to compensate for this trend. Furthermore, another one of many communities that this project will benefit is biology. Biologists tend to use high-powered computing clusters to run DNA sequencing programs. These programs need super-computer level

hardware. Data centers that service the biologist's hardware can benefit from Software-Defined RSA since it eliminates expensive configuration needed for individual and infrequent tasks.

Components:

The following table will describe the various components of this Software-Defined RSA application:

Component (Completion Date)	Description	Functional Requirement	Non-Functional Requirement
RSA Application (11/30/2016) -- LibVMI detecting memory events (11/30/2016) -- LibVMI integration with RSA Net Lib (12/15/2016) -- LibVMI copying DomU memory data (01/15/2017) -- Memory Swapping algorithm complete	LibVMI application that identifies pages to be swapped, and does the swapping to the memory server.	● Detect read and write access on a specified page of memory ● Copy page data to and from OS and Memory Server	● Leverage LibVMI to detect memory events ● Leverage LibVMI to copy data to and from an OS ● Leverage RSA Network Library to send/receive page data between the OS and the Memory Server ● Use memory swapping algorithm to determine when to swap pages
Memory Server (11/30/2016)	An application which handles requests for memory pages. It also maintains records of pages that are being used by a remote operating system.	● Maintain record of memory pages and which OS requested them ● Swap local pages or provide a page to a requesting OS	● Maintain hash table of requesting OS physical address to page mapping ● Use basic C commands to allocate memory and send it to the OS using the RSA Network Library
RSA Network Library (11/09/2016)	A network library that enables sending and receiving of memory pages between the RSA Application and the Memory Server.	● Provides the means for the RSA Application and Memory server to talk to each other ● Provides a client to	● Standard C Socket code to transmit and receive data between the RSA Application and the Memory Server

		request page swaps ● Provides a server to listen for and serve requests	● Custom buffer to serialize page requests from the RSA Application ● Buffer provides deserialization of page request ● Buffer provides serialization of a page response
User Space Application (12/30/2016)	A user space application that accesses	● An application that uses memory in the OS ● Sends single read and write request to memory ● For proof of concept purposes	● Provides a character device kernel module which allocates a kernel space buffer ● Provides a user space application to read and write from the kernel space buffer ● Allocates a 1 page (4kb) size buffer

Please see future sections that describe these components in the context, composition, and logical viewpoints.

Dependent Modules and Interfaces:

Dependent Modules:

This application is dependent on some third party modules. The first third party module is the Xen Virtual Machine Monitor. Xen provides the capabilities to run an operating system while managing the hardware resources that it provides. Xen's architecture defines a management operating system, Dom0, which can interface with and manage all virtual machines running on Xen, these virtual machines can be referred to as DomU. Dom0 and DomU can communicate with each other through a series of "event channels" that Xen exposes. One particular event channel is the memory event channel. Through a second dependency, LibVMI, a virtual machine introspection library, Software-Defined RSA is able to detect when memory-events occur in a DomU and manipulate the memory contents of that DomU.

LibVMI is the main backbone of Software-Defined RSA. With its abstractions, this project is able to detect when a page of memory is accessed. Then, upon the page access, LibVMI provides capabilities to pause that DomU. When the DomU is paused, LibVMI is able to manage the memory contents of the DomU at the target page. These two capabilities of LibVMI are heavily relied on to provide Software-Defined RSA's functionality.

Both of these dependencies need to be built, installed, and configured. There is documentation to do so in the Github repository at: https://github.com/gw-cs-sd/sd-2017-rack-scale-vms. Both of these

modules has specific hardware requirements as well. This is something that the user will need to be aware of.

Interfaces:

Software-Defined RSA is a command-line based application. A user would firstly activate the Memory Server on a remote machine. The command to launch the server is not defined concretely, but it will be along the lines of:

```
sudo ./memory-server SIZE_OF_MEMPOOL
```

Second, the user would initialize a DomU that is running the operating system to be leveraging Software-Defined RSA. To create a new DomU, a Xen utility called libxl will be used:

```
sudo xl create PATH_TO_VM_CONFIG
```

Next, the kernel module and the user space application within the target OS, or the DomU, need to be set up. Within the DomU, first compile and install the kernel module, get the address of the kernel buffer, and then run the user space application (note: these steps assume you are within the kernel module directory):

```
make
sudo insmod chr_dev_udev.ko
sudo dmesg (note: take note the buffer address)
sudo ./test_chrdev.c
```

Next, the user would run the Software-Defined RSA application inside the Xen management OS, Dom0. To run the Software-Defined RSA application, the user must provide the target memory address inside the DomU and the Xen domain name of the DomU. The former is obtained from the previous step within the DomU. The latter is acquired by running `` `sudo xl li` `` and noting the Xen domain name. With this information, start the Software-Defined RSA application (note: this assumes execution from the LibVMI examples directory):

```
sudo ./rsa-app DOMU_NAME MEMORY_ADDRESS
```

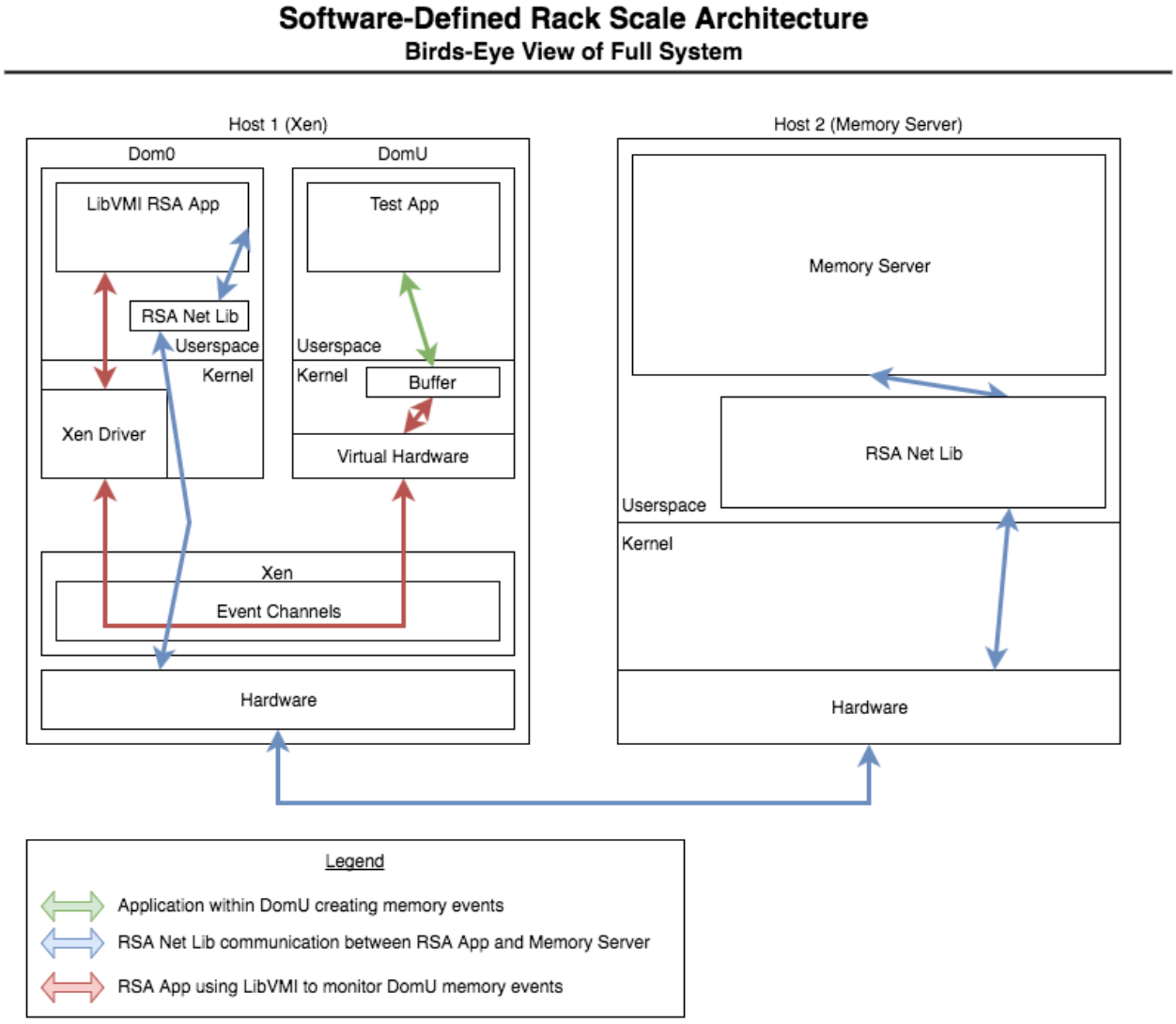
After running the Software-Defined RSA application, go back to the DomU and hit ENTER which will prompt the user space application to send memory accesses to the kernel buffer. This will then prompt the Software-Defined RSA application to print out information regarding the memory accesses.

Each access will send a command using the RSA Network Library, either “download_more_ram” or “upload_some_ram”, to the Memory Server. The Memory Server will then handle the appropriate API call and respond to the Software-Defined RSA application with either a new page of memory, or a “swapped” page of memory--swapped page of memory means the Memory Server has been storing a page of memory for the OS.

As of now, this is unintelligent. Once the intelligent memory paging algorithm is implemented, the user will not need to specify a memory address, rather the user would specify a process ID. With this ID, the memory paging algorithm will monitor how the process is using its memory and swap pages appropriately to optimize for performance.

System Diagrams:

Overview:



This system is asynchronous so there is no order to which “colored arrow” in the diagram above executes first.

Context and Composition Viewpoint: