Secure Mobile SDN

by

Ankur Chowdhary

A Thesis Presented in Partial Fulfillment
of the Requirements for the Degree
Master of Science

Approved June 2015 by the
Graduate Supervisory Committee:

Dijiang Huang, Chair
Hasan Davulcu
Hanghang Tong

ARIZONA STATE UNIVERSITY

August 2015

ABSTRACT

The increasing usage of smart-phones and mobile devices in work environment and IT industry has brought about unique set of challenges and opportunities. ARM architecture in particular has evolved to a point where it supports implementations across wide spectrum of performance points and ARM based tablets and smart-phones are in demand. The enhancements to basic ARM RISC architecture allow ARM to have high performance, small code size, low power consumption and small silicon area. Users want their devices to perform many tasks such as read email, play games, and run other online applications and organizations no longer desire to provision and maintain individual's IT equipment. The term BYOD (Bring Your Own Device) has come into being from demand of such a work setup and is one of the motivation of this research work. It brings many opportunities such as increased productivity and reduced costs and challenges such as secured data access, data leakage and amount of control by the organization.

To provision such a framework we need to bridge the gap from both organizations side and individuals point of view. Mobile device users face issue of application delivery on multiple platforms. For instance having purchased many applications from one proprietary application store, individuals may want to move them to a different platform/device but currently this is not possible. Organizations face security issues in providing such a solution as there are many potential threats from allowing BYOD work-style such as unauthorized access to data, attacks from the devices within and outside the network.

ARM based Secure Mobile SDN framework will resolve these issues and enable employees to consolidate both personal and business calls and mobile data access on a single device. To address application delivery issue we are introducing KVM based virtualization that will allow host OS to run multiple guest OS. To address the security problem we introduce SDN environment where host would be running bridged network of guest OS using Open vSwitch . This would allow a remote controller to monitor the state of guest OS for making important control and traffic flow decisions based on the situation.

Index terms— BYOD (Bring Your Own Device), ARM, Virtualization, Hypervisor, VM(Virtual Machine), SDN (Software Defined Network), Controller, OpenFlow, GRE(Generic Routing Encapsulation), oVS (OpenFlow Virtual Switch), KVM(Kernel Based Virtual Machine)

ACKNOWLEDGEMENTS

I would like to thank Dr. Dijiang Huang for providing me opportunity to work on this research project. He is a great mentor and source of inspiration to all graduate students working under him.

During this thesis work I received a good deal of feedback from other members in the lab so I would like to thank members of our lab for their invaluable inputs. I would like to thank the committee members who took out the time from their busy schedule to be present here for my defense. I was greatly benefited by the work done by VirtualOpenSystems towards Virtualization on ARM platform. I was helped a lot technically by team of VirtualOpenSystems. I am grateful to them and other groups online who are supporting Open source development.

Lastly I would like to thank my family and friends without whose encouragement and support I would not have been able to do this.

TABLE OF CONTENTS

CHAPTER Page

LIST OF TABLES

LIST OF FIGURES

Figure Page

Chapter 1

INTRODUCTION

1.1 Background

The mobile network in future would be composed of smart-phones and hand-held devices and it will play big part in defining the working environment in corporations. The new buzz word Bring Your Own Device (BYOD) has been coined to describe the consumerization of the IT. IT organizations are no longer interested in providing and provisioning the IT equipment for the individuals. Most of the companies in past had a separate department for maintaining individual IT equipment so the approach of offloading the task of hardware / device to the individual can significantly benefit the organizations in terms of cost cutting, delays in providing IT services due to communication gaps between various departments, etc. Technology (2012). The IT organizations can simply provide cash incentive to individuals to bring their own device for work, and use organization approved Operating System, device image, secured emulation environment and anti-virus software when working for company using the device. The second aspect of IT consumerization is that advent of mobility smart-phones, tablets, blackberry devices means that work doesn't stop at 5:30 PM. Also the dramatic growth in computing services and mobility trends e.g.-3G/4G on smart phones means consumers/ workers want to use their devices on the go as well as during normal work hours. This would come across both as an opportunity as well as challenge for the organizations. The opportunity is present in the sense that it would lead to productivity increase and significant cost reductions. However this also brings forth a lot of challenges especially from security perspective for organizations. The organizations have to make sure that the devices are secured from external threats and at the same time prevent data ex-filtration and unauthorized access to the resources SpA (2012). To explore this research problem we need a suitable platform which can serve as proof of concept for

implementation on a large scale. We are selecting the ARM CPU to serve as our base model. The reason for selecting the ARM platform is performance and ubiquity provided by the ARM devices. The solution would be a KVM/ARM virtualization solution. The other benefit of using KVM is that its an in built kernel module kvm.ko, so we can keep KVM/ARM in lines with new kernel releases without the additional maintenance costs. The solution will have host operating system running KVM environment and unmodified guest OS would be running in the emulation environment provided by the host OS Dall and Nieh (2010). Virtualization can in future be extended to provide other features such as high availability, load balancing features. The thesis work will make use of Open vSwitch to establish a networking framework between the host OS and guest OS. This will bring in the scope of introducing programmability into the network using SDN solution. Programmability in the network will help us build a flexible OpenFlow based secure mobile SDN framework for our system. The control plane of SDN will have direct control over the Data Plane elements Oltsik (2010). This will help deal with issues in traditional networks like devices from different vendors and two separate devices from same vendors. So individuals can bring any smart phone/ tablets of their choice as long as they have the virtualization feature support. The network of emulated virtual machines would be easily managed by the organization. We plan to establish guest OS using Open vSwitch framework and remotely host controller such as OpenDayLight in cloud environment. Controller can then communicate with the ovsdb database of Open vSwitch through SSL or GRE mechanism. We will compare our solution with other mechanism of introducing security in IT enterprises and see why this solution will be significantly better than other proposed solutions such as Prioritized Defense Deployment for BYOD; feedback based strategic sampling security, application sand-boxing, etc.

1.2 Contribution

The novel aspect of the solution is that using Open vSwitch we can introduce the secure mobile SDN framework and correspondingly programmability in our network. It will make

the solution vendor agnostic. So we can virtualize and manage the devices from different vendors as long as they support virtualization feature. Since ARM CPUs are common in many smart-phone devices E.g- Samsung Galaxy S5 uses ARM Exynos 5250, the proof of concept can then easily be deployed in real world scenarios, and tested for performance and security aspects. Other important contributions of the work are that it will have significant cost savings, secured environment for managing the VMs and better scalability as compared to some of the existing BYOD solutions. The result expected of this work is a base model/prototype serving as experimental platform for mobile networking framework. The ARM CPU provides Hardware Virtualization Extensions so we make use of KVM virtualization on base ARM Development board for booting up the host OS and make use of KVM and emulation software QEMU to bootup the guest OS via call to

```
KVM_VCPU_RUN ioctl()
```

in userspace. The ioctl() call handler is present in file arch/arm/kvm/arm.c which issues HVC(hypercall) instruction. The Open vSwitch feature will be installed on top of Host system so that we can introduce secure mobile SDN framework on host OS and manage guest OS.

1.3 Organization of Thesis

The thesis work is organized as follows Chapter 2 describes the related work in the field such as End User Virtualization, Yocto Project, etc. In Chapter 3 we discuss the System components, hardware requirements and architecture of our solution. Chapter 4 discusses the setup of Host and Guest OS, the hardware platform, Open vSwitch and detailed implementation of the solution. Finally we conclude the Thesis discussion in Chapter 5 and provide details about future work in this field. Appendix section at the end discusses some roadblocks, errors encountered during the development phase and other potential development platforms.

Chapter 2

RELATED WORK

According to a ESG survey Oltsik (2010) performed on 315 security professionals working in enterprise organizations about most difficult Mobile Security challenges brought to light the main concerns are:

- Scope of policy enforcement on devices.
- Data loss in case device is stolen.
- Breach of confidentiality and integrity.

The subsequent sections deals with various approaches to deal with these security challenges using schemes such as Prioritized Defense Deployment, Virtualization, Mode of Usage Separation, etc.

2.1 End User Virtualization

The Virtualization technology allows multiple operating systems on a Physical device to share resources such as memory, I/O devices and storage.

The scope of virtualization is not limited to a particular area. For instance Green Computing makes use of computing and IT resources efficiently to provide energy efficiency, and has became an interesting research domain Liangli *et al.* (2012).

The paper discusses current frameworks and security perspectives in desktop and application vitualization. It also highlights many standards adopted by companies such as Cisco and Citrix following the available technologies.

There are two main modes which are not necessarily independent of each other:

- **Streaming Mode** The application and desktop both runs on end user's device and are synced with the data center image.
- **Hosted Mode** Desktop and applications run on data center while image is sent to end user devices.

The authentication is required by end user in order to access the company resources. This authentication is provided by a security broker. This process helps standardize and unify the company wide tools and management process and is used by many leading companies.

OS, user profile and applications are present on separate layers, so user can get any device anywhere kind of environment and provides fair bit of security via resource isolation and authentication.

The main flaw in this kind of approach is that no useful dynamics of the network can be captured on a organization wide basis. Also this framework is not dynamic enough to deal with security threats. We need a robust framework since dynamics of attack keep on changing.

Another drawback is that since the individual VMs are compartmentalized, it is not possible to observe the processes running at hardware and OS levels by a system such as Network Intrusion Detection System.

2.2 Prioritized Defense Deployment

Prioritized defense deployment discusses a solution intermediary between security provision and mechanism intrusiveness/cost. It makes use of T-dominance - a distributed algorithm approach to capture the temporal-spatial pattern in enterprise environment. Based

on this distributed algorithm, strategic sampling is used for malware detection and prioritized patching is used as a recovery mechanism. Association logs of smart-phones and their wireless access points are used to fetch the collocation information.

Smart-phones can be considered as nodes of the graph and their collocation is used to assign them a reachability value r(u,v), where u and v represent two smart-phones, which serves as edge weight between the nodes. The structure would be like a weighted undirected graph.

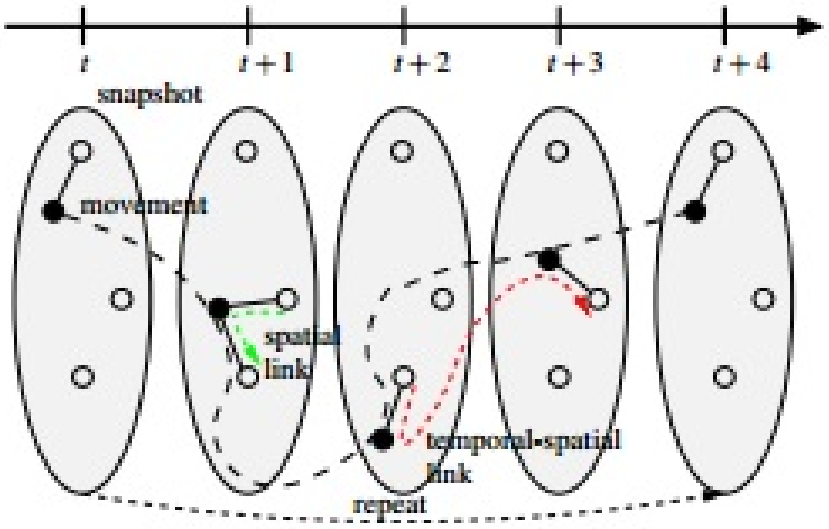

Figure 2.1: Prioritized Defense Deployment using T-Dominance 'Peng *et al.* (2013)'

This graph is known as reachability graph G(P) for smart-phones P=u,v,w,.... Filtered reachability graph G^T(P) is derived from this graph with edges having weight less than T.

The smart-phones are divided into two classes agents and non-agent. This decision is based on the security representation in the network. Agents are given priority over non-agents for defense mechanism deployment.

- Strategic sampling like honey-pot. They attract and expose propagating malware. Agents are chosen based on their security representativeness to get quantifiable security provision for malware detection.

- In prioritized patching agents are considered good deliverers of security patches because of their better connectivity, for instance an important authority in a social network based graph.

Various mechanism of selecting agents are used. Agents with good reachability are selected with higher probability for defense deployment compared to randomly selecting a phone as agent once in a while. Results indicate that strategic sampling and prioritized defense deployment achieves better performance cost wise compared to patching all smart-phones deterministically or patching smart-phones using a random selection process Peng *et al.* (2013).

While this approach looks reasonable in context of mobile phones, it's scope doesn't cover other devices that can be connected via physical media such as Ethernet cable. Also it can be complex to manage the data for collocation information and dynamically changing the structure of graph on addition and removal of devices.

There are no practical implementations of this solution, so there are concerns regarding the robustness and scalability of this solution.

2.3 MOSES

This paper discusses policy enforcement mechanism on the smart-phones. According to this paper the OS environment of smart-phones are mostly hard coded and cannot be configured as per user requirements. Mode of Use SEpeation approach makes use of soft-virtualization through controlled software isolation Zhauniarovich *et al.* (2014).

Some features and advantages of this approach are:

- End user can configure the security profile dynamically according to his/her requirement.
- Profile switching can be user driven or automatic
- Security profile is based on the context. The features for the context can be defined either in terms of low level feature variables e.g-time and location or high level variables e.g-trust, reputation

Compartments where applications and data are stored are called Secure Profiles. Smartphones raw sensors such as GPS, camera and logical sensors can be used to define the context in terms of boolean variables. If context evaluates to true, profile associated with context is activated. In case of conflicts, MOSES chooses the SP with highest priority. Application and data separation occurs at kernel level through file system virtualization approach.

This approach is limited only to Android smart-phones currently.

2.4 KVM/ARM and Virtual Open System

Virtual Open System and Linaro Networking Group have focused their efforts on Virtualization of ARM. KVM/ARM project started as a research project at Columbia University and is also supported by Virtual Open Systems. Linux kernel 3.9 and beyond provide KVM support for ARM architecture Dall and Nieh (2013b).

Since ARM CPU have become common in mobile devices, tablets, servers, so there is a growing demand of utilizing the Virtualization benefits for ARM devices. Split mode CPU virtualization offered by ARM/KVM solution allows hypervisor to split execution across CPU modes Dall and Nieh (2014). The architecture introduces three modes - the normal user mode USR, the kernel mode SVC for running privileged instructions, and a new mode HYP which is more privileged than SVC mode. There are several changes in terms of OS architecture, for instance HYP mode only maintains a single page table base register, and there is no address space split between user mode and kernel mode.

The kernel is by default booted in HYP mode. This makes the architecture backward compatible with the legacy systems since kernel always boots in SVC mode in legacy systems. Pre boot stub known as "decompresser" decompresses kernel image into memory. On detecting that it booted in HYP mode, a temporary stub must be installed which would allow the kernel to fall back to SVC mode and run the decompresser code. This is made possible using the HYP stub "arch/arm/kernel/hyp-stub.S".

Other architectural details of KVM/ARM such as hardware trap, Virtual GIC and timers have been discussed in the article Dall and Nieh (2013b).

Various development boards have been used as base models by the organization Virtual Open Systems Systems (2013) such as TI - OMAP 5432 based on ARM v7 architecture, Samsung Exynos 5250 based on ARM v7, Fast Models based simulation platform, Juno development board Inc. (2014b) that could serve as good potential solution for KVM/ARM.

The model used as part of thesis is based on the same KVM/ARM architecture. The fundamental difference is that while this model only plans to boot Virtual Machines on top of ARM based development boards, we plan to make use of Open vSwitch to create a bridged network of guest VMs on top of Host VM. This would allow us to implement network wide policies, firewall rules, load balancing solutions on Guest VMs using a smart OpenFlow controller such as POX, Floodlight etc.

2.5 Yocto Project

Some other approaches for developing KVM/ARM based solutions were considered to act as base development models for this thesis project such as Yocto Build system. This project provides infrastructure for building complete complete Linux OS using package metadata. It uses OpenEmbedded (OE) build system, tools such as "bitbake" to develop Linux images and associated user space applications Project (2014b). It is a layered based architecture for extension and customization of base system. For instance the base layer is OpenEmbedded Core (oe-core). There is yocto specific metadata layer on it's top, followed by hardware specific Board Support Packages (BSP) and UI specific BSP Figure 2.2.

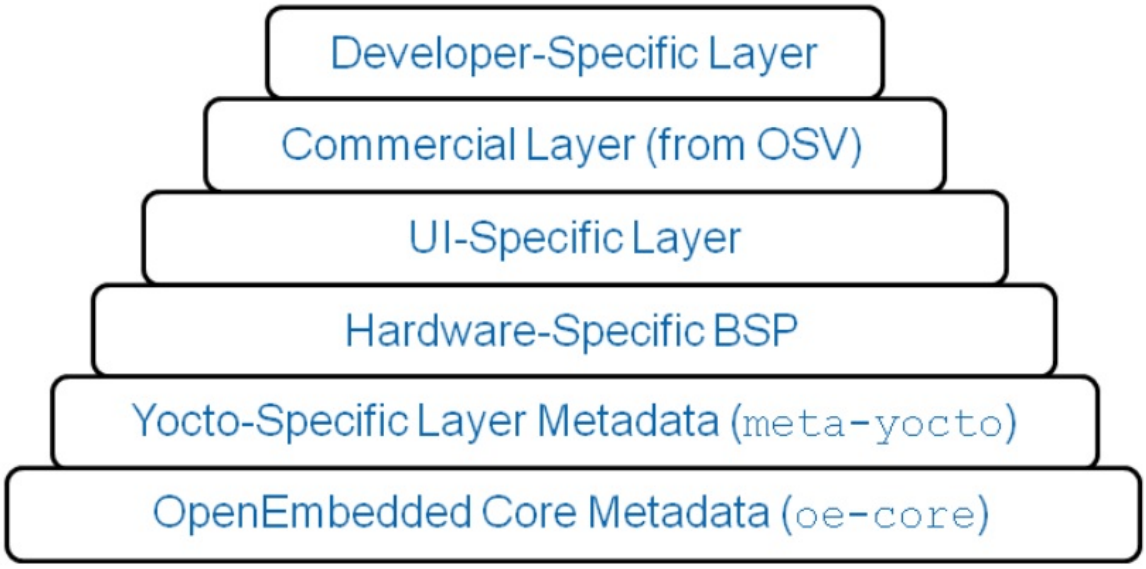

Figure 2.2: Yocto Project Layered Framework 'Project (2014b)'.

The layers *meta-ti* Dmytriyenko (2014) and *meta-virtualization* Project (2014a) have been developed to provide support for virtualization on TI based development boards. The paper Papaux *et al.* (2014) discusses use of Yocto Project to configure valid virtualization architecture on top of TI-OMAP 5432 and performance analysis of hardware virtualization extensions.

This looks like a very good approach and I developed meta-ti and meta-virtualization based Yocto Build for project, but it failed to boot in Hypervisor mode either because of faulty packages in the build system or improper bootloader. So this can be considered for future development boards for the project.

2.6 XEN Virtualization on ARM

XEN is a type-1 hypervisor running on top of hardware. Dom0 is the privileged virtual machine to drive the on board devices. All other virtual machines are of type DomU or unprivileged VMs. Devices such as SATA and network cards are assigned to Dom0. Unprivileged VMs make use of the paravirtualized back-ends run by Dom0 to access the device resources. Shared page memory and ring protocol are used for communication between

front-end and back-end. Specialized driver domains for running drivers corresponding to a special class of device are used so that Dom0 doesn't have to run all the drivers Foundation (2013).

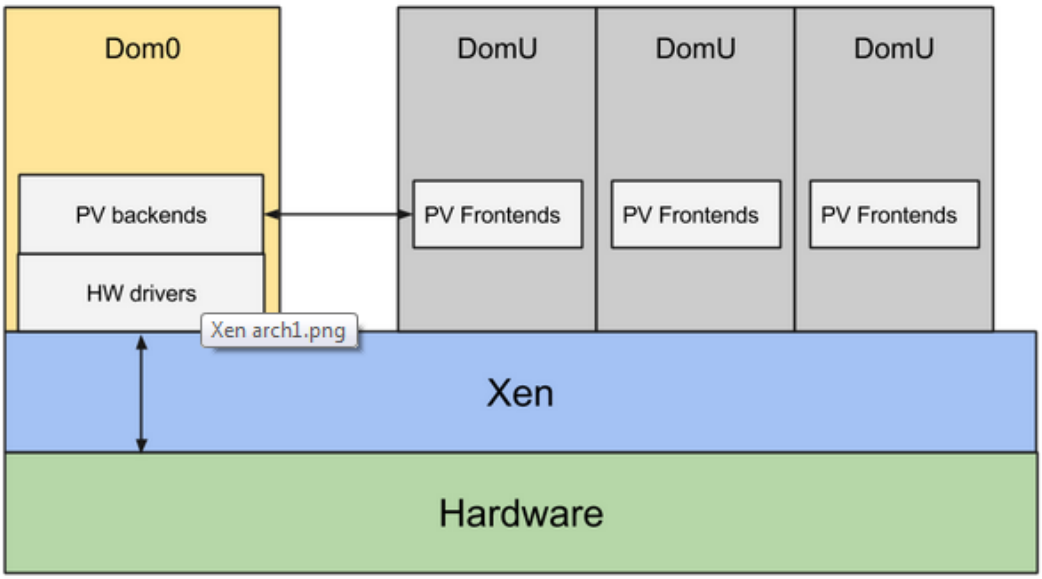

Figure 2.3: Xen Virtualization over ARM 'Foundation (2013)'.

XEN on ARM eliminated the need for QEMU as there is no emulation feature in XEN over ARM. It incorporated paravirtualized interfaces for IO. XEN for x86 had two types of guests: HVM and PV guests. This has been replaced with only one type of guests: PV in XEN over ARM. The architecture is divided into levels: EL0, EL1, EL2 and HVC is the instruction used to switch between HYP and Kernel modes Foundation (2013). The solution

for XEN over ARM solution has been discussed in over ARM (2014) for OMAP5432. This solution was tried in TI OMAP5432 but the solution is not stable and fails to load the kernel. Another problem with this solution is that paravirtualized front-ends and back ends for GPS, Camera etc. don't exists commercially as of now. The problem arises when multiple VMs need to access these devices simultaneously. In that scenario we will need to write PV front end and back-end drivers which is a non-trivial task.

All of the approaches to achieve a secured networking framework for Mobile devices discussed above have been compared below.

Technique	Features	Advantages	Drawbacks
End User virtualization	Multiple systems on same physical machine. App can be running in end user device or data-center.	Security broker mediated authentication. Clean Separation of OS and user profile.	Difficult to capture useful network dynamics. Difficult to observe state of processes inside sandboxed VMs.
Prioritized Defense Deployment	Used distributed algorithm to capture the temporal and spatial characteristics in enterprise environment.	Significant cost gains in terms of threat detection and patching mechanism.	No practical implementations suggesting scalability and robustness.
MOSES	Mode of Separation through controlled software Isolation. Make use of policy enforcement.	Context based security gives a good view of security profile. User requirement based profile.	Limited only to Android platform. It has not been tested on other platforms.
KVM/ARM Virtual Open Systems	Virtualization on KVM based ARM platform. Patch to u-boot for booting kernel in HYP mode.	Tested on various platforms, e.g- Exynos 5250, TI OMAP 5432, Samsung Chromebook.	Makes use of a bridged network, less scope for programmability.

Yocto Project	Provides complete Linux build infrastructure through package metadata. Support layers for virtualization.	Widely supported by community, Board support packages to support variety of vendors.	Very few practical deployments, need to be tested for stable deployments for KVM/ARM solution.
XEN Project	Type-1 hypervisor, runs directly on hardware, everything else runs as VM on top of XEN. Dom0 is privileged VM, DomU is normal user level VM.	It does not require an emulation environment, and as a result is faster and much more secured.	Virtualization support for mobile platform devices such as GPS, camera for more than one VM, require new paravirtualized font ends and back ends.

Table 2.1: Comparison of Various Approaches Towards a Mobile Infrastructure

Chapter 3

SYSTEM DESIGN AND SECURE MOBILE SDN ARCHITECTURE

This chapter introduces the design methodology and approach used for setting up BYOD framework. In the next chapter experimentation and analysis have been introduced.

3.1 System Use Case Diagram

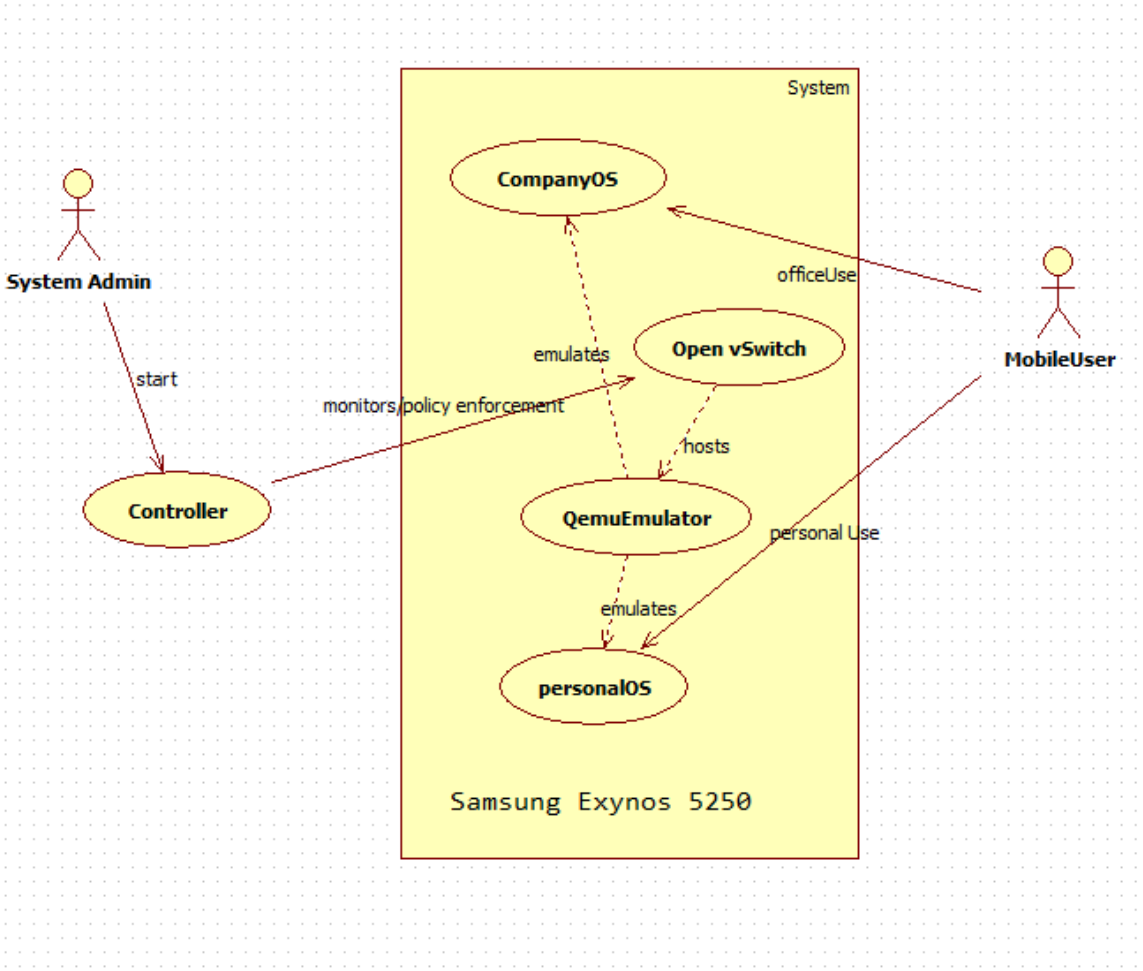

Figure 3.1: BYOD System Architecture.

The figure 3.1 shows the use case diagram for the system. As can be seen from the model, the Mobile User can use personal and company OS, but the company OS would be monitored by the system administrator for network traffic. Open vSwitch would be used to provide bridged network for the system. Qemu would be emulating both the guest Operating Systems. So in case of any new flow entry, the controller would be notified, and using controller alerts, the admin. can take decision such as allow , deny or redirection of network traffic.

3.2 System Components

Our research goal is to build a BYOD framework that would allow easy implementation of corporate policies and data capture for analysis. Two essential components of this framework are hardware assisted virtualization and Open vSwitch.

3.2.1 Hardware Assisted Virtualization

The base model for experimentation and evaluation are ARMv7 based development boards, hence we will discuss the hardware assisted virtualization that ARM offered as part of ARMv7 and ARMv8 releases in this section. The Reduced Instruction Set Computer (RISC) architecture of ARM helps achieve good balance of high performance, small code size, low power consumption and reduced silicon area ARM (2013). Also many smart-phones and handheld devices have ARM based processors-with multiple cores. ARM architecture virtualization extensions and Large Physical Address Extension(LPAE) enable efficient implementation of Virtual Machine hypervisors for ARM architecture compliant processors.

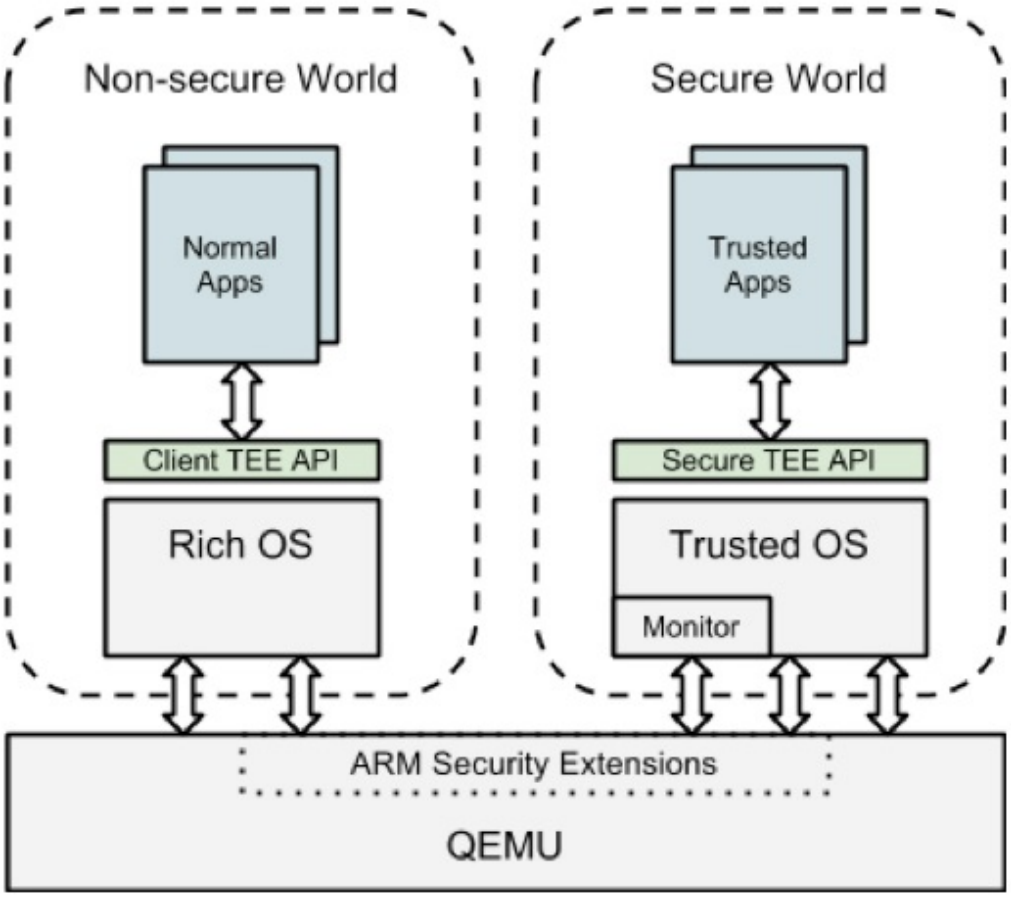

Figure 3.2: ARMv7 Security Extensions .

Another important feature that ARMv7 and ARMv8 based processors offer is TrustZone(Security Extensions) 3.2 apart from new CPU mode HYP mode discussed in KVM/ARM section in previous chapter. TrustZone splits the mode into two worlds - secure and non-secure. A special mode - monitor mode is used to switch between secure and non-secure worlds. Although secure mode doesn't work in HYP mode, since trap and emulate support is not present, still we can run sensitive applications in secure world. The paper KVM/ARM:The design and implementation of Linux ARM Hypervisor Dall and Nieh (2014) discusses secure world and other details about CPU, Memory, Timer and Interrupt Virtualization of ARM.

The KVM ARM architecture achieves comparable and in some cases better performance than traditional KVM x86 based devices. There have been some significant changes in ARM architecture. The table compares some key features of ARM and x86 architectures.

ARM introduces split mode virtualization feature. The virtualization support is provides by the Hyp mode and the kernel mode runs core Linux kernel services without any intrusive code modifications. The hypervisor is split into two components to provide this functionality, namely: highvisor and lowvisor.

Lowvisor deals with setting up of execution context, isolation between contexts, context switches between host and guest execution modes. Lowvisor also deals with interrupts and exceptions that trap to hypervisor. Highvisor on the other hand scheduling, locking mechanism and memory allocation functions. Stage-2 page fault handling from the VM and instruction emulation are also handled by the highvisor Dall and Nieh (2013a).

Whenever there is requirement of switch between the hypervisor and VM, the process involves mode transitions. If the user is running the VM and a trap to the hypervisor mode is required, VM will trap to lowvisor running in Hyp mode, followed by lowvisor trapping to highvisor mode. On the other hand if we need to switch from hypervisor to VM mode, first a trap to Hyp mode occurs and then switch to the VM takes place.

Feature	x86 Architecture	ARM Architecture
Modes of Operation	It has root and non root modes orthogonal to CPU protection modes. Trap and emulate from root to non root is possible.	Hyp mode is separate and more privileged than user and kernel mode.
Privilege Mode	Root mode has full support for kernel and user mode functionality.	ARM Hyp mode has its own set of features that provide much more complex options.
Save/Restore Operations	Hardware support for VM control block and auto save/restore for switching to and from root mode is available.	ARM provides no hardware support and any save or restore state feature needs to be implemented in software.

Table 3.1: Comparison of x86 and ARM Architecture

3.2.2 Open vSwitch and its Comparison with other Switching Techniques

Linux provides three kinds of software switches:

- **bridge** It has key components FDB(Forwarding Database), STP(Spanning Tree), etc. It uses promiscuous mode to receive all packets Makita (2014). Many common NIC filters make use of unicast without promiscuous mode if destination is not it's MAC address.
- **macvlan** Four types of modes are : private, vepa, bridge, passthru. It uses unicast filtering if supported instead of passthru.
- **Open vSwitch** It is switching technique that supports OpenFlow. It can be used

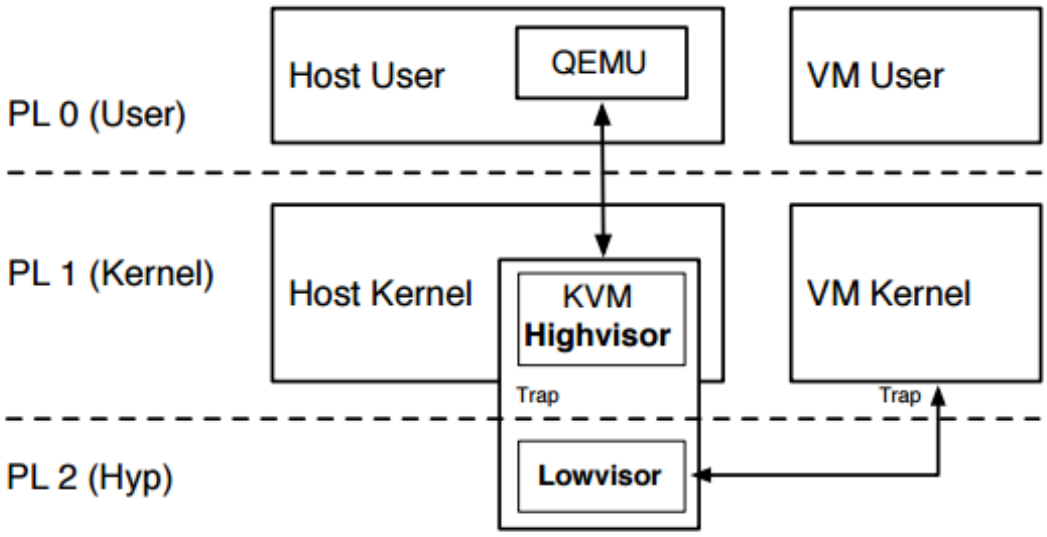

Figure 3.3: ARM System Architecture. 'Dall and Nieh (2013a)'

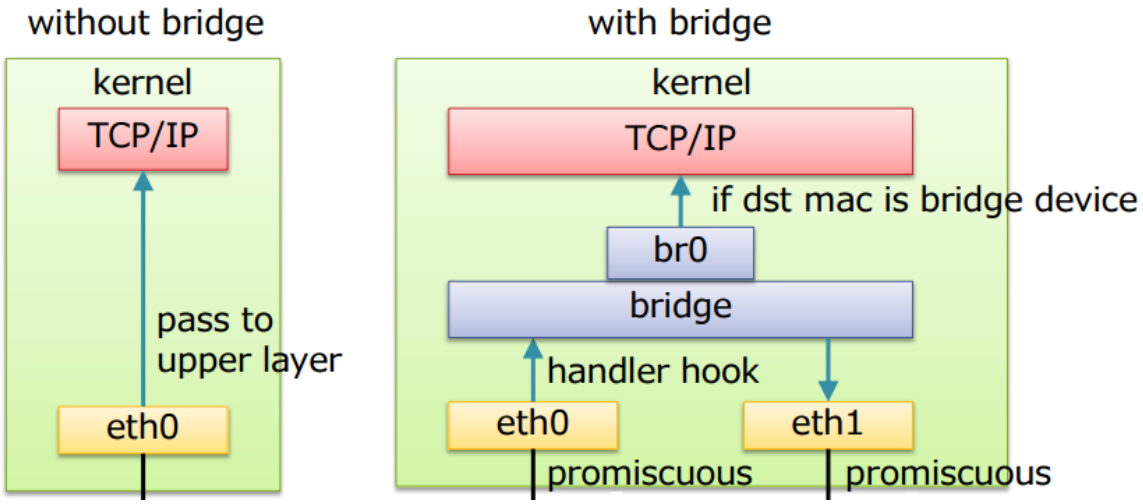

Figure 3.4: Switching Using Bridge 'Makita (2014)'.

as normal switch and also has many features such as GRE, VXLAN. It provides forwarding based on Flow. User space has control plane and data plane is present in kernel space. It there is a flow miss-hit, upcall to userspace daemon is made.

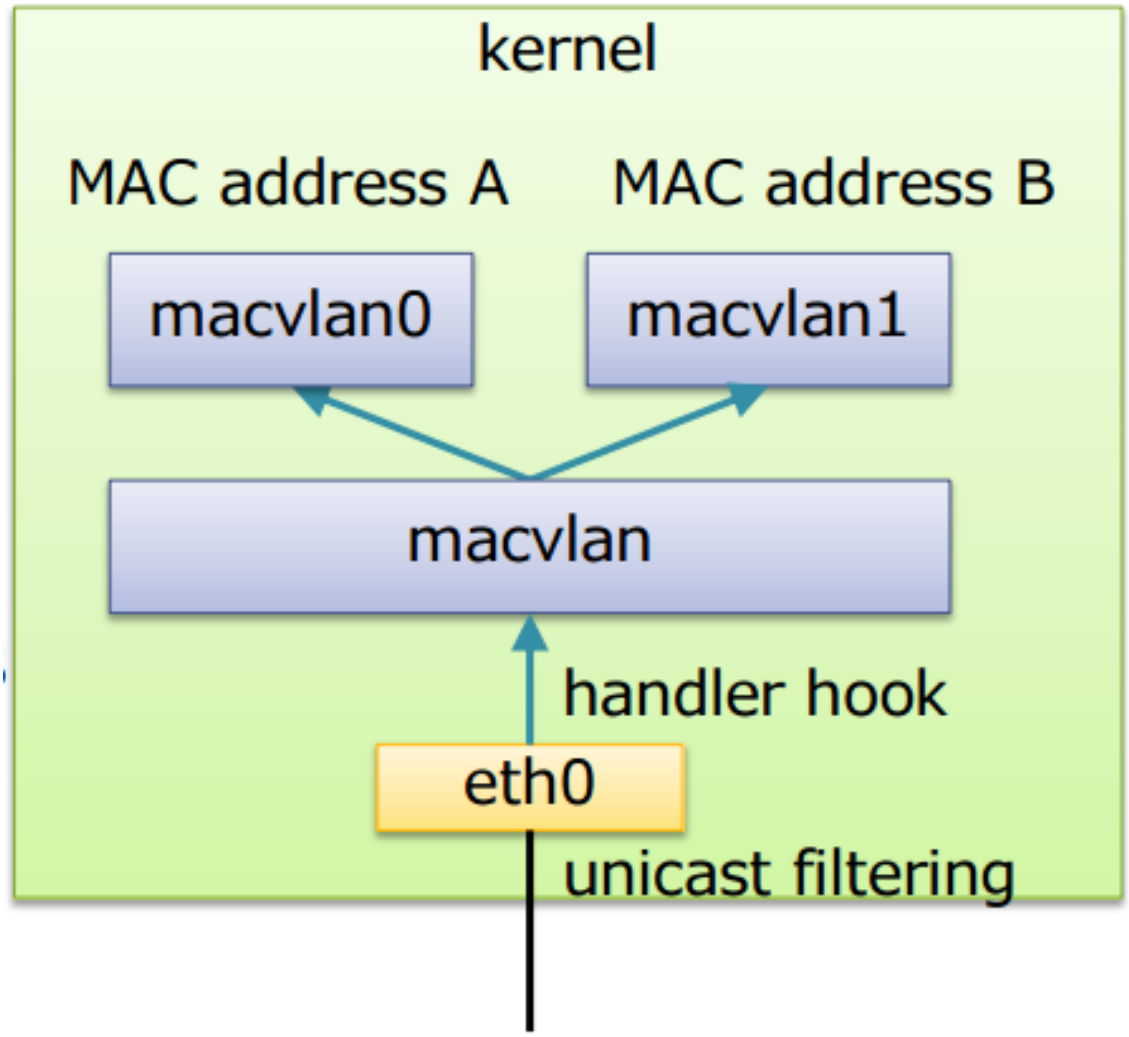

Figure 3.5: Switching Using Mac VLAN 'Makita (2014)'.

Considering the use of OpenFlow for our solution and need of a centralized controller in the architecture which cannot be achieved without Open vSwitch, we are making use of Open vSwitch to construct the underlying network.

Virtual switches connect the interfaces of Virtual Machines and establish connection to outer network with help of Physical Network Interface Card (pNIC). Open vSwitch is used extensively in OpenStack and OpenNebula. We can explore broad range of OpenFlow features via software switches, that cannot be provided by hardware switches. Two

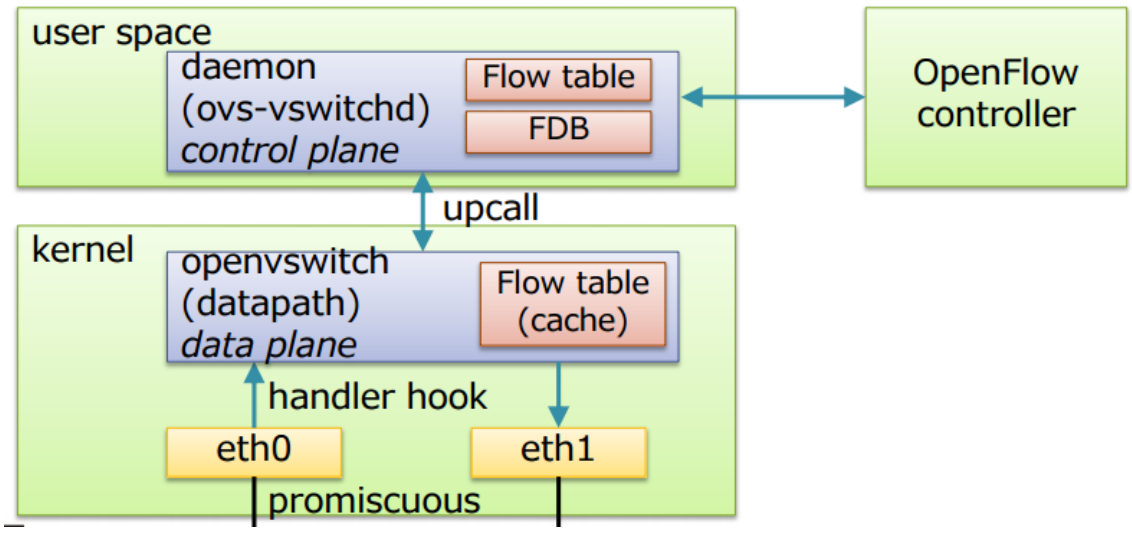

Figure 3.6: Switching Using Open vSwitch 'Makita (2014)'.

important parts of OvS are ovs-vswitchd daemon that controls switch and is responsible for implementing OpenFlow protocol, and datapath kernel module to implement the packet forwarding Emmerich *et al.* (2014).

The performance comparison of various forwarding techniques as shown in table 3.2 with single CPU core per VM and switch discussed in paper Emmerich *et al.* (2014) suggests that Open vSwitch proves to be fastest Linux kernel packet forwarding application.

Application	pNIC-pNIC[Mbps]	pNIC-vNIC[Mbps]	pNIC-vNIC-pNIC[Mbps]	pNIC-vNIC-vNIC[Mbps]
Open vSwitch	1.88	0.85	0.3	0.27
IP Forwarding	1.58	0.78	0.19	0.16
Linux bridge	1.11	0.74	0.2	0.19
DPDK vSwitch	11.31	-	10.5*	6.5*

Table 3.2: Open vSwitch Performance Analysis

Also rule based system used by Open vSwitch would make it easier to configure a generic OpenFlow controller for VMs connected to Open vSwitch and implement access control based on the flows.

For instance blocking packets from a compromised VM or redirecting packets to a different destination. The Software Defined Network model thus formed can be made more scalable by configuring a controller that is logically centralized but physically distributed, e.g.- HyperFlow Tootoonchian and Ganjali (2010). Open vSwitch across two host systems can also communicate via a Generic Routing Encapsulation (GRE) tunnel which would be discussed as part of future work for this project and is beyond scope of Thesis work.

3.3 System Architecture

The system architecture for BYOD framework uses Samsung Exynos 5250 as base platform. The board was first introduced in 2012 and features two Cortex-A15 cores clocked at 1.7 GHz InSignal (2013b). It offers 50 percent higher per MHz performance compared to commonly used Cortex A9 architecture. It is lightweight (150g) and is common in many handheld devices such as Google Nexus Tablet. It is also very cost effective ($150). So the goal of project is to develop the framework on this platform so it can later be used for deployment in a bigger testing environment or code base from this platform to be deployed on other tablets or mobile devices compatible with ARM architecture.

The architecture consists of three basic features:

- As shown in the Figure 3.7 below the Host OS should have hardware assisted virtualization enabled and should boot in HYP mode enabled. We are choosing Ubuntu Precise (12.04) as the Host operating system. The host operating system will need a device tree blob exynos5250-arndale.dtb which contains description of hardware. It is specific to the development board. Another component required is the kernel itself uImage. Details of generating both will be discussed in next chapter.

- Open vSwitch should run on top of Host Platform. This would allow us to make use of OpenFlow APIs along with other advantages of Open vSwitch as discussed briefly under Open vSwitch subsection.

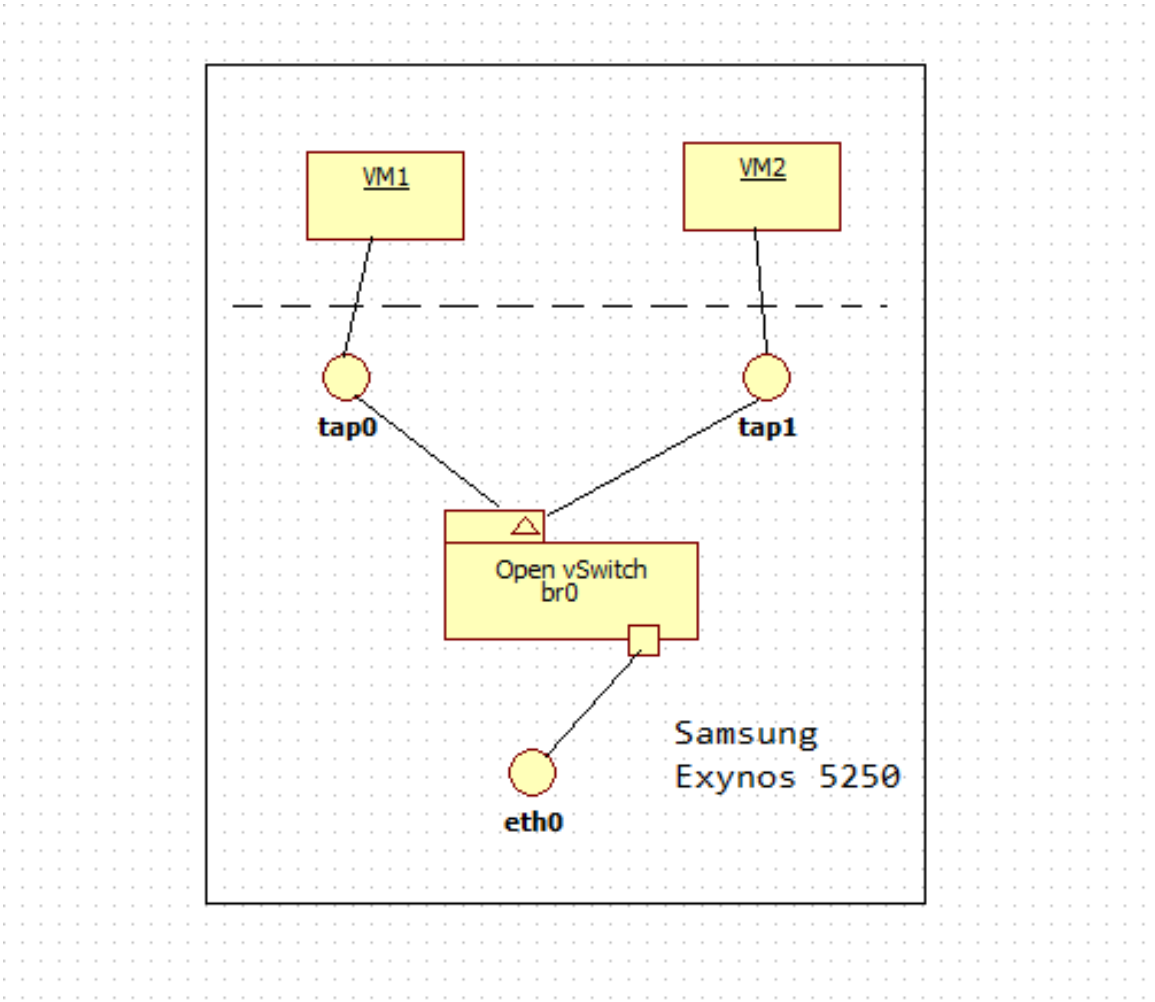

Figure 3.7: System Design Using Open vSwitch.

An important consideration for this project is compatibility of Open vSwitch with the Linux Kernel. The 3.3 shows the compatibility of Linux Kernel with Open vSwitch versions. Therefore we are choosing Linux Kernel 3.14.32 and Open vSwith version openvswitch-2.3.1.

Open vSwitch	Compatible Linux Kernel
1.4.x	2.6.18 to 3.2
1.5.x	2.6.18 to 3.2
1.6.x	2.6.18 to 3.2

1.7.x	2.6.18 to 3.3
1.8.x	2.6.18 to 3.4
1.9.x	2.6.18 to 3.8
1.10.x	2.6.18 to 3.8
1.11.x	2.6.18 to 3.8
2.0.x	2.6.18 to 3.10
2.1.x	2.6.18 to 3.11
2.3.x	2.6.18 to 3.14

Table 3.3: Open vSwitch and Compatible Linux Kernel

- The guest operating system should boot using the bridged network provided by Open vSwitch. Additionally the guest operating system requires the kernel image to boot guest OS zImage, the DTB file, file system image to boot guest, a modified version of qemu to emulate and drive KVM from userspace.

3.3.1 Hardware Requirements and Features

Samsung Exynos 5250 uses ARM v7 Cortex A15 CPU. Some important features of the hardware platform are:

- Advanced Single Instruction Multiple Data version2 (SIMD v2).
- High performance single or double precision Floating Point Unit (FPU).
- Security Extensions for enhanced security.
- Virtualized Interrupts with Multicore ARM Trust Zone architecture.
- Multiprocessing facility through multiprocessing extensions.
- Virtualization extensions for developing virtualized systems.

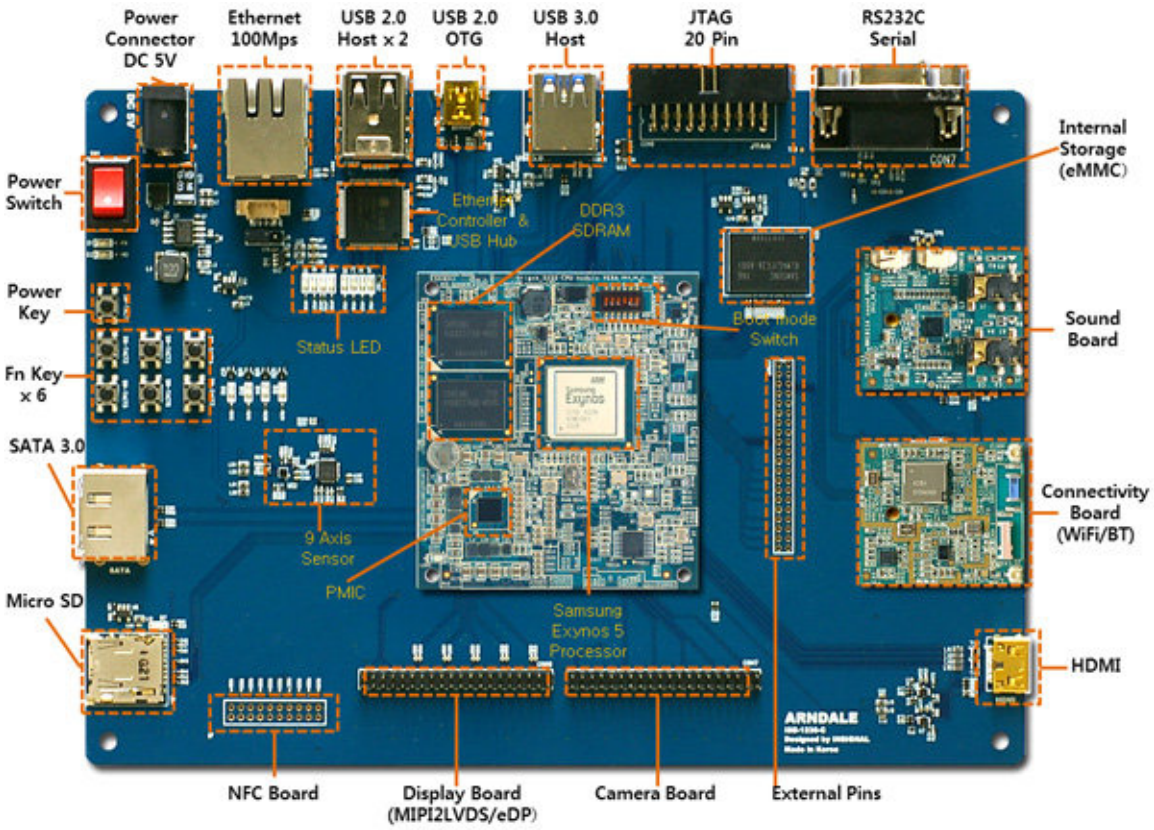

Figure 3.8: Arndale Development Board.

- Ability to reach 1.7 GHz processing capability to meet requirements for performance optimized consumer applications.

Other requirements for setting up the test environment include:

- System to prepare the host, guest images, qemu. A system with sufficient disk space and Ubuntu-12.04 is suggested.
- Samsung Exynos 5250 Arndale Board 3.8.
- A memory card (16 Gb suggested) to load host and guest OS.
- A serial port cable to communicate with board using Minicom.
- Power supply (5V).
- LAN cable connected to the Arndale development board.

Chapter 4

IMPLEMENTATION AND EVALUATION

In this chapter we will discuss the system setup and user experiments for our BYOD framework. ARM processor is present in many tablets such as Samsung Chromebook, Nexus 9, etc. Windows is also talking about Windows over ARM. It's been said Windows over ARM could kill desktops World (2014). All these reasons support our choice of base development platform. The steps for this part are similar to Virtual Open System guide for KVM over Arndale Systems (2013) and tutorial from Linaro Linux group Lim (2013). The contribution in this section is mainly building a Open vSwitch compatible kernel and booting up pre built guest OS using Open vSwitch on top of host OS as shown in figure 4.1.

The overall system activity includes booting up of Linux kernel in Hyp mode, start of Open vSwitch on top of Host Linux Kernel and then starting the guest kernel on top of bridged network created by Open vSwitch.

4.1 Host and Guest Setup

We are using Ubuntu 12.04 LTS as base system to setup Linux Kernel and rootfs for both host and guest OS. We will first install the following packages on host system:

- Qemu: a generic opensource machine Emulator Systems (2013).
- debootstrap: a tool which installs base system into subdirectory of another already. installed system.
- An ARM Cross Compiler for cross compiling the packages.

```
$  sudo apt-get install -y gcc-arm-linux-gnueabihf
$  sudo apt-get install -y qemu qemu-user qemu-user-static
$  sudo apt-get install -y debootstrap
```

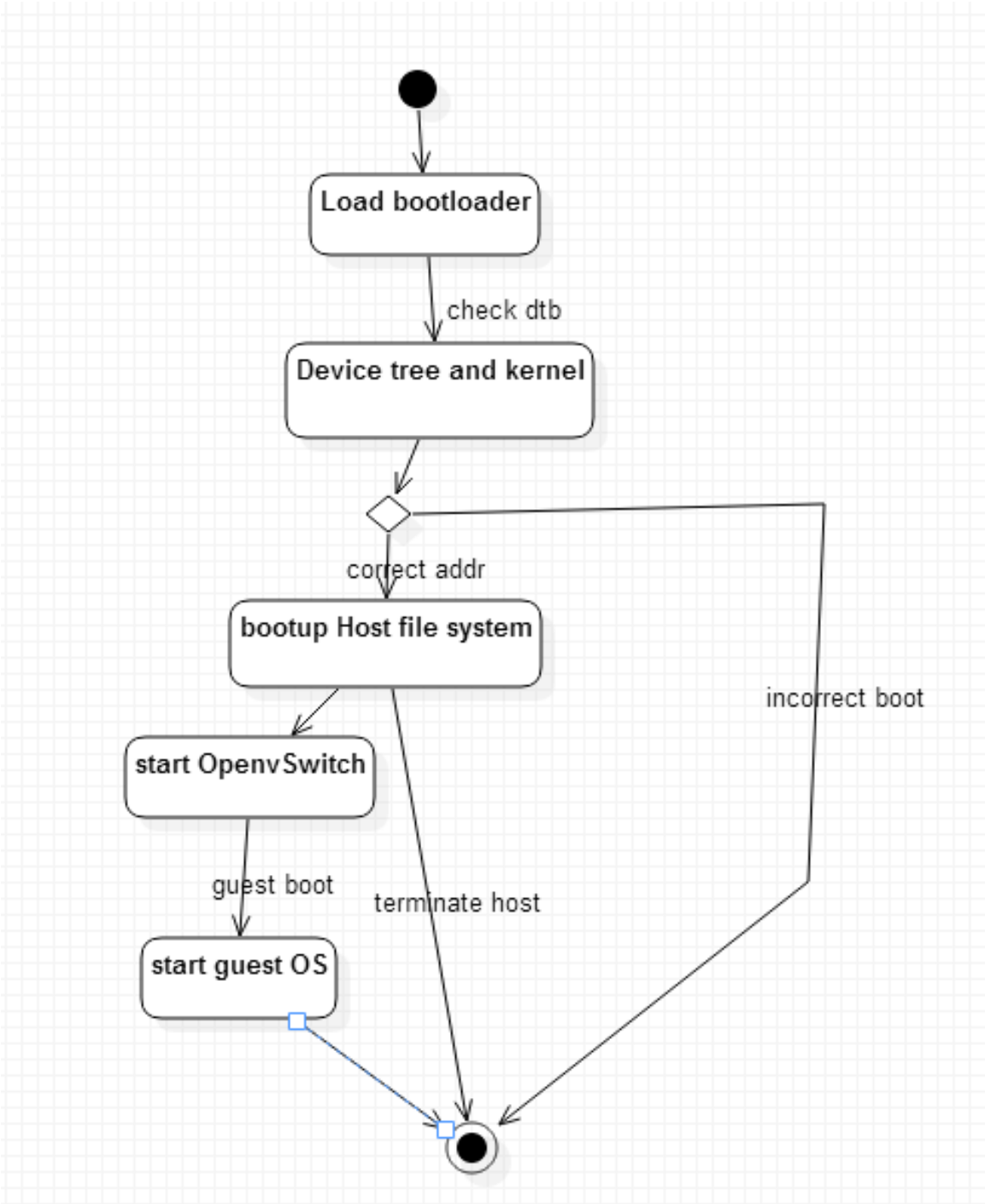

Figure 4.1: System Activity Diagram.

4.1.1 Host File System and Kernel Setup

We will install arm-precise-root using qemu-debootstrap and setup root password for arm-precise-root Lim (2013).

```
$  sudo qemu-debootstrap --arch=armhf raring ./arm-precise-root
$  sudo chroot ./arm-precise-root
```

Next copy the file etc/init/tty1.conf to ttySAC2.conf and change tty1 to ttySAC2.conf. Also perform same steps to create a file ttyAMA0.conf in same directory. You will also need to change the baud rate to 115200 for serial port login. Also we need to add a line ttySAC2 to file etc/securetty.

Add the following lines to etc/source.list

```
deb http://ports.ubuntu.com/ precise main restricted universe
deb-src http://ports.ubuntu.com/ precise main restricted universe
```

Update the sources.list file, reconfigure locales and install the following packages and exit from host rootfs

```
$ locale-gen en_US.UTF-8
$ dpkg-reconfigure locales
$ apt-get install -y ssh gcc make xorg fluxbox tightvncserver
$ apt-get install -y libsdl-dev libfdt-dev bridge-utils uml-utilities
$ apt-get clean
$ exit
```

Now we need to setup kernel for the Host OS. For this part we need to download linux-linaro-lng version 3.14.32 and unzip it to act as source tree. We will use it to generate dtb

and uImage for the host file system. Run the commands below on your Linux machine. We need to download sources from tar.gz repository. If we download from git repository, we will get errors while installing the Linux Kernel modules on arndale development board.

```
$ wget https://releases.linaro.org/14.06/components
/kernel/linux-linaro-lng/linux-linaro-lng-3.14.3-2014.06.tar.bz2
$ tar -zxvf linux-linaro-lng-3.14.3-2014.06.tar.bz2
$ cd linux-linaro-lng
$ export CROSS_COMPILE=arm-linux-gnueabihf-
$ export ARCH=arm
$ mkdir ../lll-kvmhost ./scripts/kconfig/merge_config.sh
-O ../lll-kvmhost/ linaro/configs/linaro-base.conf
linaro/configs/distribution.conf linaro/configs/kvm-host.conf
linaro/configs/arndale.conf linaro/configs/ovs.conf

$ make O=../lll-kvmhost/ uImage dtbs modules

$ cp -t /tftpboot/ ../lll-kvmhost/arch/arm/boot/uImage
../lll-kvmhost/arch/arm/boot/dts/exynos5250-arndale.dtb

$ sudo make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf-
O=../lll-kvmhost/ INSTALL_MOD_PATH=${KVMHOST_ROOT} modules_install
```

This will generate uImage in the folder lll-kvmhost/arch/arm/boot and exynos5250-arndale.dtb in lll-kvmhost/arch/arm/boot/dtc

These two files would be used later.

4.1.2 Guest FileSystem and Kernel Setup

Guest file system that will be used as VM on top of host file system can be downloaded from Virtual Open System. We are using Versatile Express - an ARM release as guest kernel. For this we can download guest-zImage and guest-vexpress.dtb from the website Systems (2013). We will then need to create a bootable guest OS image and copy the precise file system used for host OS in that image. Following steps are sed for this

```
$ dd if=/dev/zero of=./ubuntu.img bs=1MiB count=512
$ mkfs.ext3 ./ubuntu.img
$ sudo mount -o loop ubuntu.img mnt/
$ sudo cp -a precise/* mnt/
$ sudo umount mnt/
```

4.1.3 *Bootloader for ARM*

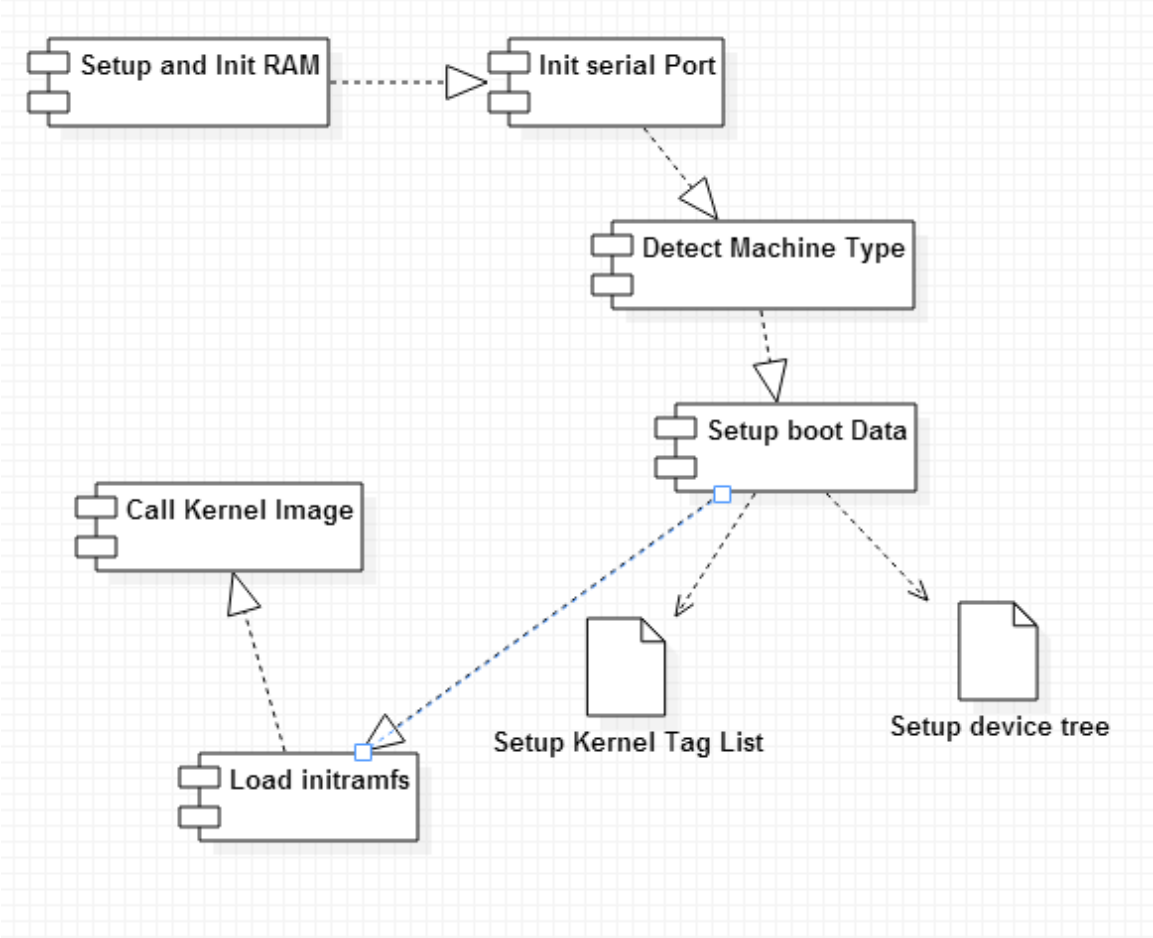

Figure 4.2: Linux Arm Bootloading Process.

The following steps are followed by ARM bootloader traditionally to boot in host mode:

- The kernel finds and stores all the RAM needed by it for volatile data storage.
- The option console="" is used by bootloader to enable one serial port on the target.
- `MACH_TYPE_xxx`

  value is provided by bootloader to the kernel through register r1. The file linux/arch/a4rm/tools/mach-types stores these values.

- Register r2 is used to pass physical address of boot data to kernel. The bootloader must also create and initialize kernel tagged list or device tree . The tag-list must be placed it in region of memory where it cannot be overwritten. If dtb option is used instead of tag list, it should be put in RAM at assigned address, and initialized with boot data.

- If initramfs is used, it should be placed in memory region where it cannot be overwritten.

- Finally the kernel can be called via zImage from the flash or RAM location.

To boot the ARM kernel in HYP mode directly the bootloader has been modified. The bootloader used for booting up the Host file system in Hypervisor mode and two additional files can also be downloaded directly from Virtual Open System Systems (2013).

```
u-boot.bin
arndale-bl1.bin
smdk5250-spl.bin
```

4.1.4 Qemu for ARM Setup

On the x86 Laptop/PC run the commands below:

```
sudo apt-get install -y pkg-config-arm-linux-gnueabihf

cat | sudo tee /etc/apt/sources.list.d/armhf-raring.list <<END
deb [arch=armhf] http://ports.ubuntu.com/ubuntu-ports precise main
restricted universe multiverse
deb-src [arch=armhf] http://ports.ubuntu.com/ubuntu-ports precise
main restricted universe multiverse
END

sudo xapt -a armhf -m -b zlib1g-dev libglib2.0-dev libfdt-dev
libpixman-1-dev
sudo dpkg -i /var/lib/xapt/output/*.deb
git clone git://git.qemu.org/qemu.git
cd qemu
git checkout -b v1.6.0 v1.6.0
git submodule update --init dtc
mkdir build; cd build
../configure --cross-prefix=arm-linux-gnueabihf- --target-list=arm-softmmu
--enable-kvm --audio-drv-list="" --enable-fdt --static
make
```

This will generate qemu that would be used to emulate the ARM guest OS on top of Host OS using the Open vSwitch bridge.

4.2 Booting Up Arndale Board

We will be using SD card (16 Gb suggested) as a bootup medium to run Host OS Ubuntu-12.04 on top of Arndale board. Use a SD card reader and plug it into Laptop or desktop for copying file system.

Copy bl1, spl and u-boot to the SD card. The SD card will be present as a storage device on PC and can be viewed by command "fdisk -l". In my case it is /dev/sdb.

```
$ sudo dd if=arndale-bl1.bin of=/dev/sdb bs=512 seek=1
$ sudo dd if=smdk5250-spl.bin of=/dev/sdb bs=512 seek=17
$ sudo dd if=u-boot.bin of=/dev/sdb bs=512 seek=49
```

Now we will copy host kernel and dtb file. These values will depend upon the size of device tree and the Linux kernel. For instance my Linux Kernel size was (4.6 Mb) so I have copied dtb after sufficient space so it doesn't overlap with address space of kernel (uImage).

```
$ sudo dd if=uImage of=/dev/sdb bs=512 seek=1105
$ sudo dd if=arndale.dtb of=/dev/sdb bs=512 seek=13393
```

Now format the SD card and copy precise file system, and source tree onto the SD card. Remember to format from start block address after dtb file, so leave sufficient space.

```
$ sudo fdisk /dev/sdX
$ n
$ p
$ 1
$ 16384
$
$ w
```

```
$ mkdir mnt
$ sudo mkfs.ext3 /dev/sdb1
$ sudo mount /dev/sdb1 mnt
$ sudo cp -a ./arm-precise-root/* mnt/
$ sudo umount /dev/sdb1
```

Now we need to copy the Guest file system, guest device tree, guest Kernel and qemu generated for ARM to SD Card. So simply copy these files to root folder of the SD card. Alternatively you can use scp to copy these files once the board boots up and gets an IP Address via DHCP.

Next step would be to insert SD card into the appropriate slot on the board and boot it up.

Follow the steps below to boot up the board:

- Connect the serial port cable provided with Arndale board to USB port of your PC/Laptop.
- Download and install minicom or a similar application on your PC.
- Check the DIP switch settings on Arndale board. It should be 001000 from left to right.
- Press the bootup key present on Arndale board. This link InSignal (2013a) is a more detailed version of booting up Arndale board.
- As soon as board starts loading kernel and reads device tree properly, press escape key and enter commands below to configure boot arguments and environment variables. These commands will vary depending upon location of your kernel and device tree.

```
$ env edit bootargs
$     root=/dev/mmcblk1p1 rw rootwait earlyprintk
```

```
        console=ttySAC2,115200n8 --no-log
$ env edit bootcmd\
$       mmc read 40007000 451 3000;mmc read 42000000
        3451 100;bootm 40007000 - 42000000
$ env save
$ boot
```

If the configuration is correct, board would bootup in HYP mode as can be seen in the figure 4.3 below.

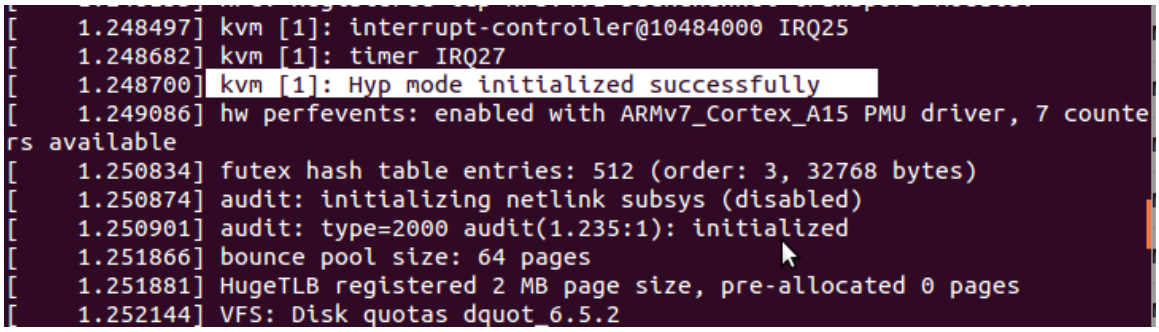

Figure 4.3: Host OS Booting in Hyp Mode.

Once this is done board would boot up successfully. Enter username and password to login on the board as root user. Once the system boots up and gets the IP address, run the following command in root mode. The board has a time skew and this command syncs it with online ntpserver.

```
$ ntpdate -s time.nist.gov
```

4.3 Open vSwitch with KVM

This section deals with installation and configuration of Open vSwitch on Arndale board, and running KVM on top of Open vSwitch. First check kvm driver has been successfully

installed with command

```
$ ls /dev/kvm
```

The installation of openvswitch daemon depends upon the modules : stp.ko, llc.ko bridge.ko and vxlan.ko, so before configuring Open vSwitch make sure these modules have been installed properly. You can use the command "lsmod modulename" to check if a module is present or absent.

Next we need to build Open vSwitch. It would require linux-headers or source tree to build openvswitch.ko module. For this we can copy sources from linux-linaro-lng-3.14.32 into the directory /lib/modules/3.14.32/build so that Open vSwitch is able to find dependent modules for it's build. The version for Open vSwitch used for build is openvswitch-2.3.10 as it is compatible with Linux kernel 3.14.32.

Steps below are followed in order after this:

```
$ apt-get update
$ apt-get install -y git automake autoconf gcc uml-utilities
  libtool build-essential git
$ wget http://openvswitch.org/releases/openvswitch-1.10.0.tar.gz
$ tar zxvf openvswitch-2.3.10.tar.gz
$ cd openvswitch-2.3.10
$ ./boot.sh
$ ./configure --with-linux=/lib/modules/‘uname -r‘/build
$ make && make install
$ insmod datapath/linux/openvswitch.ko
$ mkdir -p /usr/local/etc/openvswitch
$ ovsdb-tool create /usr/local/etc/openvswitch/conf.db
  vswitchd/vswitch.ovsschema
$ ovsdb-server -v --remote=punix:/usr/local/var/run/openvswitch/db.sock \
```

```
                    --remote=db:Open_vSwitch,manager_options \
                    --private-key=db:SSL,private_key \
                    --certificate=db:SSL,certificate \
                    --pidfile --detach --log-file
$ ovs-vsctl --no-wait init
$ ovs-vswitchd --pidfile --detach
$ ovs-vsctl show
```

The figure 4.4 below show the expected outcome after correct configuration of Open vSwitch.

```
/openvswitch/db.sock \itch-2.3.1# ovsdb-server --remote=punix:/usr/local/var/ru
n_vSwitch,manager_options \
                    --private-key=db:Open_vSwitch,SSL,private_key \
                    --certificate=db:Open_vSwitch,SSL,certificate \
                    --bootstrap-ca-cert=db:Open_vSwitch,SSL,ca_cert \
                    --pidfile --detach>                          --remote=db:Open_\
>                     --private-key=db:Open_vSwitch,SSL,private_key \
>                     --certificate=db:Open_vSwitch,SSL,certificate \
>                     --bootstrap-ca-cert=db:Open_vSwitch,SSL,ca_cert \
>                     --pidfile --detach
root@kvmhost:~/openvswitch-2.3.1# ovs-vsctl --no-wait init
root@kvmhost:~/openvswitch-2.3.1# ovs-vswitchd --pidfile --detach
2015-05-14T18:32:30Z|00001|reconnect|INFO|unix:/usr/local/var/run/openvswitch/d.
2015-05-14T18:32:30Z|00002|reconnect|INFO|unix:/usr/local/var/run/openvswitch/dd
root@kvmhost:~/openvswitch-2.3.1#
```

Figure 4.4: Open vSwitch Configuration.

Now we need to create custom versions of qemu-ifup and qemu-ifdown scripts that would be used in KVM configuration of guest OS. The custom versions will make use of Open vSwitch bridges vSwitch (2013). The configuration files /etc/ovs-ifup and /etc/ovs-ifdown should be created for this purpose.

```
$ vim /etc/ovs-ifup
  #!/bin/sh
  switch='br0'
```

```
  /sbin/ifconfig $1 0.0.0.0 up
  ovs-vsctl add-port ${switch} $1

$ vim /etc/ovs-ifdown
  #!/bin/sh
  switch='br0'
  /sbin/ifconfig $1 0.0.0.0 down
  ovs-vsctl del-port ${switch} $1
```

After this we need to add a bridge using Open vSwitch and add a port to the bridge over which the guests can communicate. Steps for establishing this connection and configuring IP address for the bridge and gateway are as follows

```
ovs-vsctl add-br br0
ovs-vsctl add-port br0 eth0
ovs-vsctl add-port br0 tap0
ovs-vsctl list port

#ifconfig eth0 0
ifconfig eth0 0.0.0.0 up
ifconfig tap0 0.0.0.0 up



ifconfig br0 10.218.108.16 netmask 255.255.248.0
route add default gw 10.218.104.1 br0
```

The command below can be used to verify the correct configuration for Open vSwitch.

The final task for bringing up the guest Operating system requires that guest image ubuntu.img, guest device tree guest-vexpress.dtb and guest kernel guest-zImage are already

```
root@kvmhost:~# ovs-vsctl show
7fddd8f7-611b-40d4-90e2-b8537e5c316e
    Bridge br-int
        Controller "tcp:10.218.108.250:6633"
        Port "tap0"
            Interface "tap0"
        Port "tap1"
            Interface "tap1"
        Port "eth0"
            Interface "eth0"
        Port br-int
            Interface br-int
                type: internal
root@kvmhost:~#
```

Figure 4.5: Open vSwitch Interface Display.

present on the SD card. We will issue the command to qemu to bootup the guest OS on top of Open vSwitch

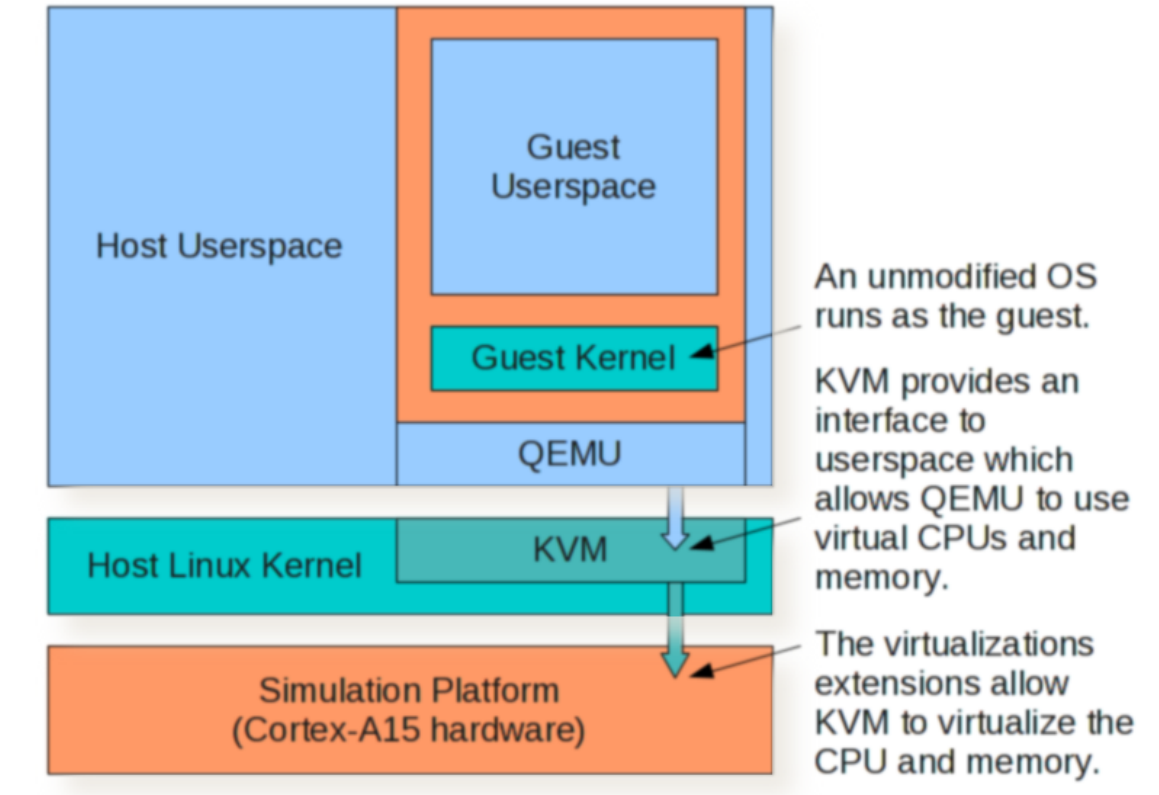

Figure 4.6: Qemu Guest Bootup on Host Linux OS.

```
./qemu-system-arm \
  -enable-kvm -kernel guest-zImage \
  -nographic -dtb ./guest-vexpress.dtb \
  -m 512 -M vexpress-a15 -cpu cortex-a15 \
  -netdev type=tap,id=net0,script=no,downscript=no,ifname="tap0" \
  -device virtio-net,transport=virtio-mmio.1,netdev=net0 \
  -device virtio-blk,drive=virtio-blk,transport=virtio-mmio.0 \
  -drive file=./ubuntu.img,id=virtio-blk,if=none \
  -append "earlyprintk console=ttyAMA0 mem=512M
   root=/dev/vda rw ip=dhcp --no-log
   virtio_mmio.device=1M@0x4e000000:74:0
   virtio_mmio.device=1M@0x4e100000:75:1"
```

The guest OS will bootup and get IP address via DHCP, as can be seen 4.7

```
[    1.279118] usbcore: registered new interface driver usbhid
[    1.280553] usbhid: USB HID core driver
[    1.281698] TCP: cubic registered
[    1.282499] NET: Registered protocol family 17
[    1.283654] Key type dns_resolver registered
[    1.284776] VFP support v0.3: implementor 41 architecture 4 part 30 variant f
 rev 0
[    1.286706] ThumbEE CPU extension supported.
[    1.287783] Registering SWP/SWPB emulation handler
[    1.289138] drivers/rtc/hctosys.c: unable to open rtc device (rtc0)
[    1.307887] Sending DHCP requests ., OK
[    2.330670] IP-Config: Complete:
[    2.337011]      device=eth0, hwaddr=52:54:00:12:34:56, ipaddr=10.218.106.241
, mask=255.255.248.0, gw=10.218.104.1
[    2.353501]      host=10.218.106.241, domain=cidse.dhcp.asu.edu, nis-domain=(
none)
[    2.366232]      bootserver=0.0.0.0, rootserver=0.0.0.0, rootpath=
[    2.376073]      nameserver0=129.219.17.200[    2.387870] kjournald starting.
  Commit interval 5 seconds
[    2.389546] EXT3-fs (vda): using internal journal
[    2.390705] EXT3-fs (vda): mounted filesystem with writeback data mode
[    2.392307] VFS: Mounted root (ext3 filesystem) on device 254:0.
[    2.393918] Freeing init memory: 192K
```

Figure 4.7: Guest OS Bootup Using Open vSwitch.

Finally the guest OS terminal will be visible and can be used for communication with other hosts or guest OS 4.8.

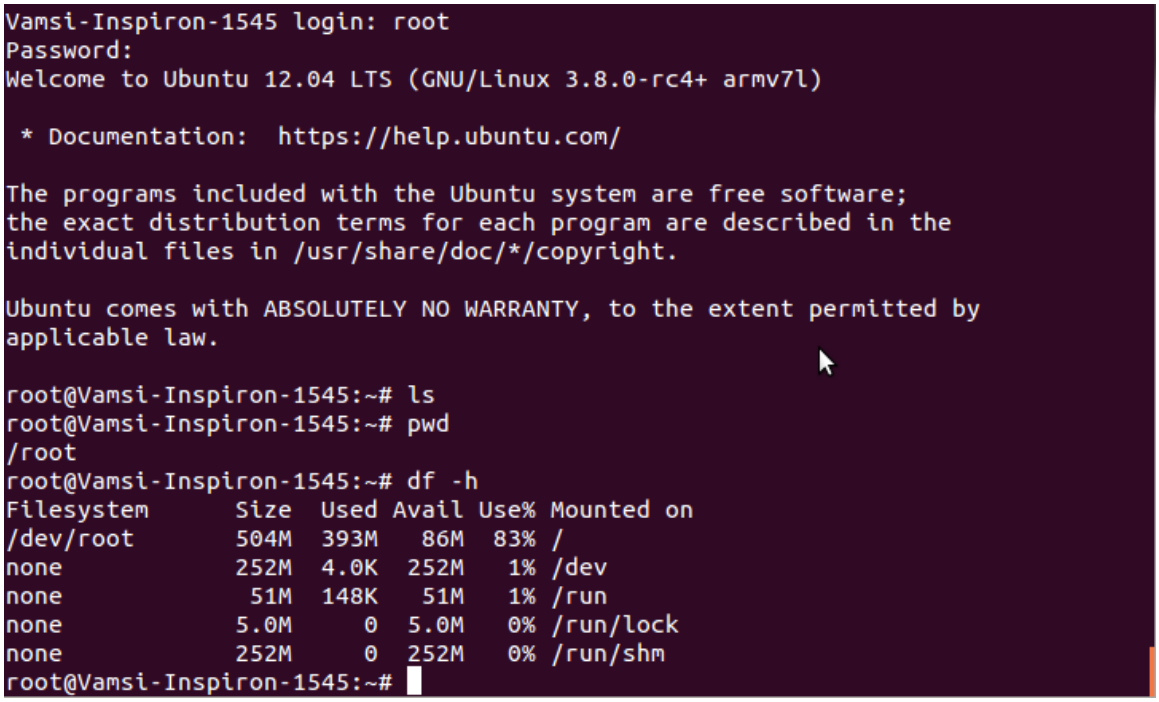

Figure 4.8: Ubuntu Guest OS Terminal .

During the development phase some files and sources were used directly as provided by various open source development groups and some file/scripts were written from scratch. List of all the modules used is as below:

File Name or Source	Purpose	Used/New
linaro-linux-lng-3.14.32	Sources for kernel	Used from Linaro Group
arm-precise-root	File system For Host OS	Generated
/etc/init/ttySAC2.conf	Serial Port Communication	Modified
/etc/securetty	Serial Port Communication	Modified
/guest	Folder for Guest OS	New
/etc/exports	x86 Host VNC communication	Modified

kconfig/merge_config.sh	Script for Host Kernel Setup	Linaro Group
linaro/configs/linaro-base.conf	Script for Host Kernel Setup	Generated
linaro/configs/kvm-host.conf	Script for Host Kernel Setup	Generated
linaro/configs/arndale.conf	Script for Host Kernel Setup	Generated
boot/dts/exynos5250-arndale.dtb	Host OS Device Tree	New
arch/arm/boot/uImage	Host Kernel	New
stp.ko, llc.ko, bridge.ko, vxlan.ko, gre.ko	Host Kernel Modules for Open vSwitch	Generated
u-boot.bin	Host Bootloader	Used from Virtual Open Systems
arndale-bl1.bin	Host Bootloader	Used from Virtual Open Systems
smdk5250-spl.bin	Host Bootloader	Used from Virtual Open Systems
qemu-system-arm	Guest Emulator	Generated
bootargs	Command to boot host	New
bootcmd	Command to boot host	New
openvswitch-2.3.10	Open vSwitch File System	Used from Open vSwitch
/etc/qemu-ifup	File to bring up qemu on Open vSwitch	New

/etc/qemu-ifdown	File to bring down qemu on Open vSwitch	New
ovs_script.sh	Open vSwitch bootup and ovsdb setup script	New
ovs_setup.sh	Open vSwitch network setup script	New
qemu-bootovs.sh	Open vSwitch Qemu Guest boot script	New

Table 4.1: List of Files Used in Development Phase

4.4 Current Work: SDN and Communication with Remote OpenDayLight Controller

SDN came into being from the need of simplifying network management and control. It has two main components, the Control Plane defines how to handle the traffic. The second element is Data Plane, which handles traffic flow based on flow entries defined by control plane. A well defined API known as OpenFlow is used by SDN to install the rules of traffic flow. OpenFlow switch such as Open vSwitch can have multiple packet handling rule tables. This provides openFlow the flexibility to behave as Firewall, switch, Network Address Translator (NAT). As discussed in the paper Feamster *et al.* (2013) discusses the need of data plane and control plane separation to provide better traffic engineering in terms of predictability, reliability, network management functions.

The goals we wish to target is enforcement of security policy separately for each part of network , for instance we may want to allow limited access to finance department servers so we can enforce security policy and isolate resources in that particular department by VLAN tagging and specific traffic flow rules for that particular VLAN as can be seen in figure 4.9.

Second advantage is that if we have abnormal traffic behavior within the network/VLAN, the controller can flag the event as Intrusion based on rules specified and quarantine a

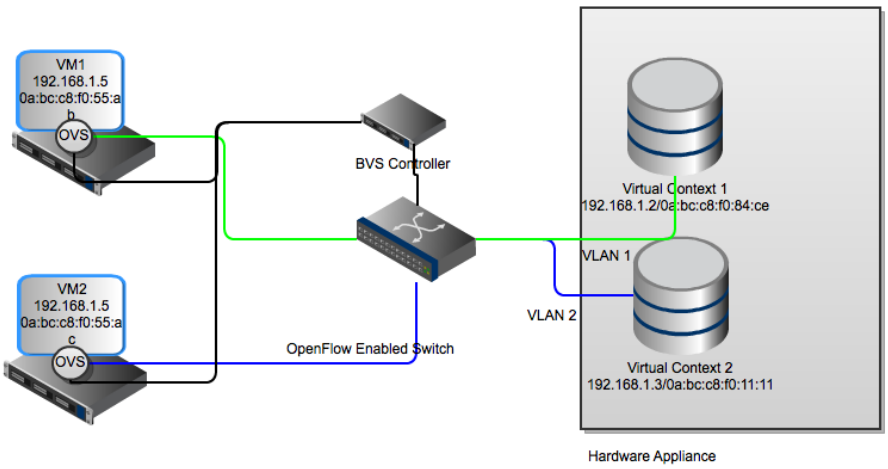

Figure 4.9: SDN Based Programmable Openflow Network.

particular VM. In context of heterogeneous mobile devices in a BYOD environment, this would be very useful, e.g- two people from same department one having Mac Air and another having Samsung Galaxy Tab can run company provided guest OS on the fly. The controller would Group them in same VLAN according to the deviceID tagging, MAC associated with a devices, etc. So policies and access rules for that particular VLAN will apply to these two tablets. The Guest OS images can be synced with secured datacenter images at a remote location.

Although performance is not out main focus, but this kind of network scenario will offer load balancing features too. So if a particular server is overwhelmed with traffic the controller can dynamically spawn another VM, server image and provide new traffic flow to distribute load over both the machines.

We can use a remote controller for managing and monitoring the network traffic flow, and achieve the functionality of a SDN network. The controller used is OpenDayLight. The modular approach provided by OpenDayLight project helps provide SDN functionality and achieve solid platform for other important features such as NFV(Near Field Virtualization).

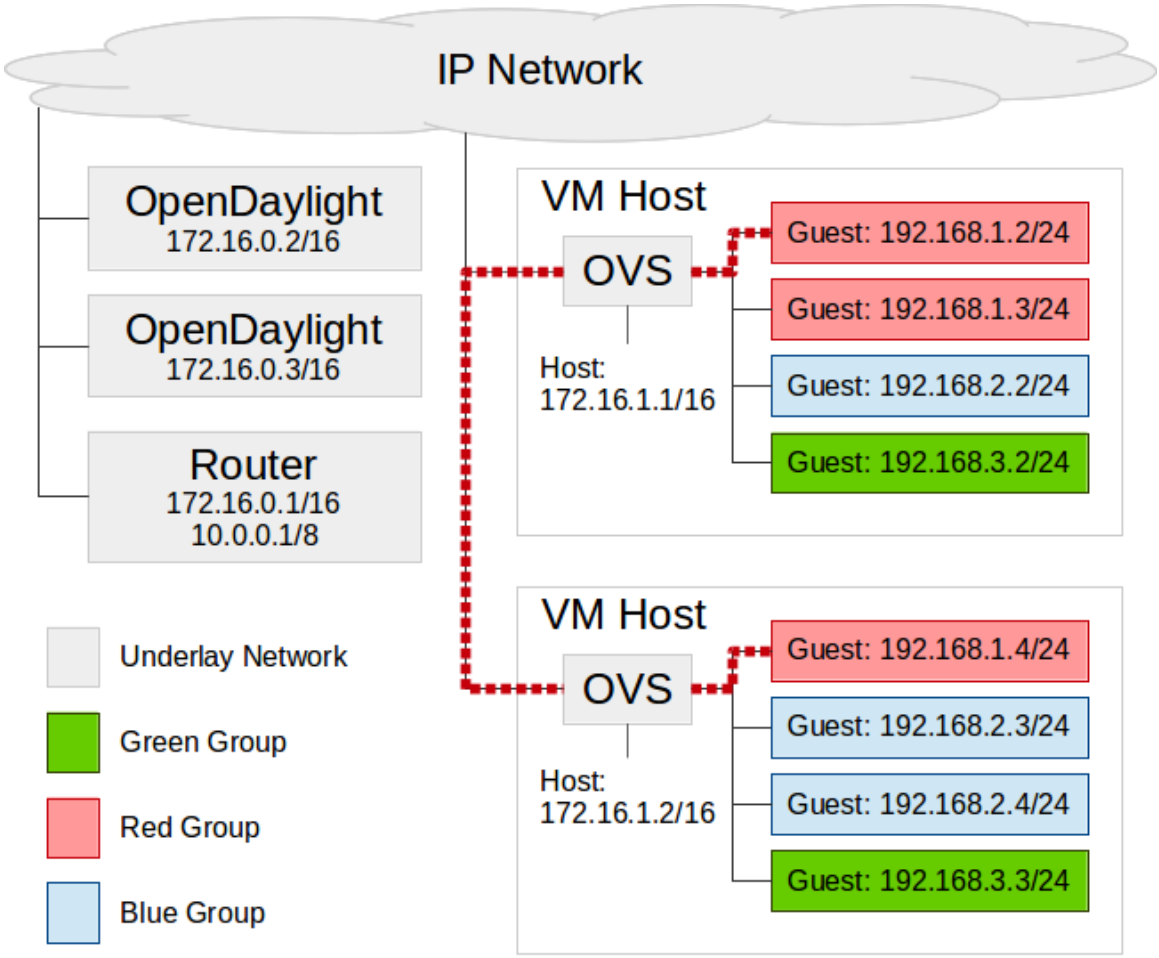

Figure 4.10: Remote SDN Controller Communication with OvS.

The Northbound APIs of the OpenDaylight controller can be used for providing the application development functionality through an abstraction layer. The southbound API will connect to the Open vSwitch present on Samsung Exynos 5250. Another good feature is that it also has plugins for Inter Controller communication, so we can achieve a distributed architecture as discussed in hyperflow Tootoonchian and Ganjali (2010). The controller

would prove very useful for managing the Virtual Machines in a BYOD scenario. In case of network events such as DoS attacks targeting a particular Mobile device the controller can have an Intrusion detection mechanism configured that looks for application protocol conformation violations or DoS attack patterns.

Chapter 5

CONCLUSION AND FUTURE WORK

5.1 Conclusion

There are various models of providing a secured framework for mobile devices, but we need a generic solution that transcends all limitations such as scalability, platform compatibility, security, etc. Mobile devices such as smart-phones and tablets will certainly play a role in future work setting of each organization. BYOD using a SDN framework can prove very efficient, scalable and secured solution for organizations in future.

The cost of implementing this framework would be significantly less than the existing infrastructure cost for providing and maintaining IT equipment to the employees. ARM being present in many target devices and new features such as Virtualization, Security Extensions, better throughput than it's peers serve as a suitable experimentation model. We analyzed drawbacks of other solutions present and KVM over ARM using Open vSwitch looks better compared to other solutions such as End User Virtualization, XEN, etc.

The model in this Thesis research work ARMv7 serves as a proof of concept that ARM is a cheap, fast and stable platform for serving as a base models for such devices. In the future work section we also discuss how the ARMv7 solution can easily be ported to ARMv8 development platforms which have better performance and larger address range but are not yet commercially available. Samsung is already using ARM and windows is also planning to use ARM as processor in their coming tablets and mobile devices. It is thus a novel idea to provide a solution/architecture which can make use of these capabilities provided by ARM and serve as a model of defining corporate network comprising of mobile devices in future. While there are many issues still to be resolved such as Controller coordination with Open vSwitch for ARM, use of VMI for finer system analysis, this research work is still a significant step towards realization of BYOD goal.

5.2 Scope for Future

ARM v7 is a good prospective platform for BYOD implementation. The future work for the project includes the deployment of ARM based BYOD platform in a testing environment. We have been working on a Mobile testbed consisting of 4 Unmanned Ground Vehicles(UGVs) and 6 Unmanned Aerial Vehicles (UAVs). This board being lightweight can be a suitable candidate to be deployed on UAVs.

The Robotic control functionality for UGVs would be provided by Arduino Uno Development board Uno (2011), which is standard for common robotic projects and UAVs would make use of Ardu Pilot Board DroneCode (2014) with compatible camera. Since Arduino platforms do not come with sufficient computing capability and the signal and network traffic analysis for testbed would require a platform with good computing capability so we Samsung Exynos 5250 can be integrated with Ardu Pilot for providing computing capability.

The Juno Development board would be considered as a platform for UGVs. Since the Juno development board has ARMv8 which comes with full scale native virtualization, and with bigger address space range. Furthermore any application developed for the platform ARMv7 can easily be ported to ARMv8 Inc. (2014c).

Another feature that would be incorporated to the system architecture is the Virtual Machine Introspection(VMI) capability. VMI capability would allow monitoring of guest OS internal state, state transitions(events) and I/O activity. As an administrator this would provide better visibility of state of VM and enforce security policies. A VMI based Intrusion Detection System can be deployed in the network which will observe the hardware state and events of a guest. Being isolated from host it would provide high degree of attack resistance even if host is corrupted.

The technique for incorporating VMI capability is an active area of research currently and one of the models we can consider is LibVMI LibVMI (2013). The qemu emulator would need to be patched like:

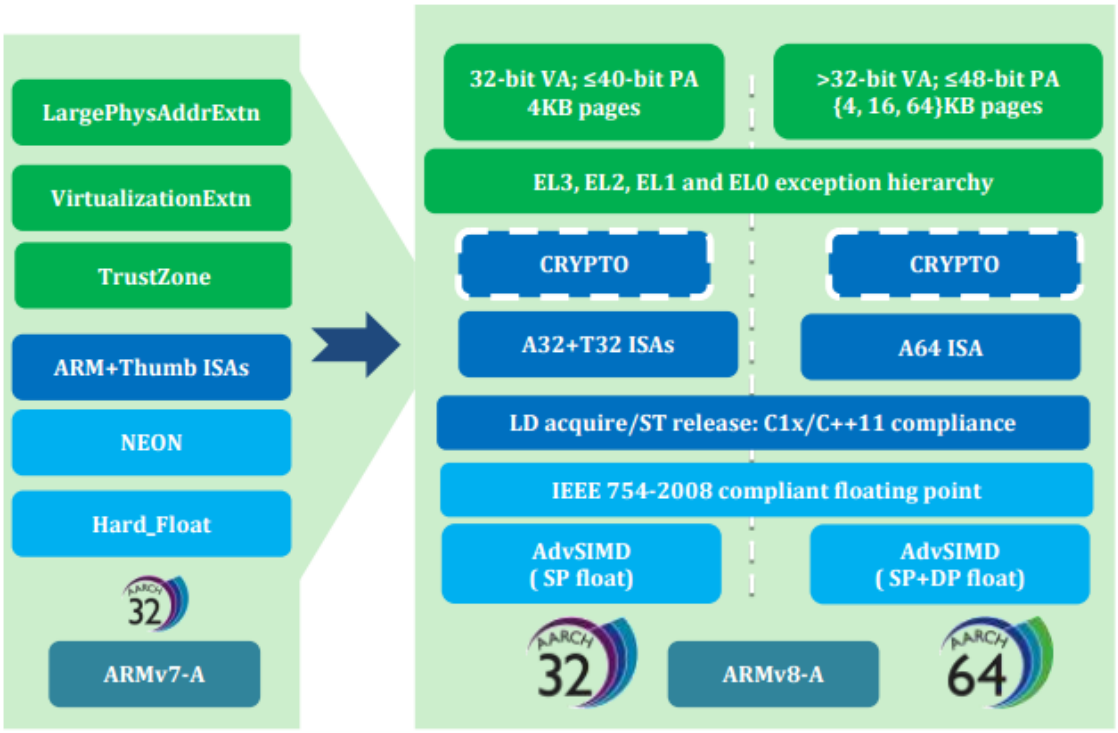

Figure 5.1: Comparison Between ARMv7 and ARMv8.

```
<domain type='kvm' xmlns:qemu='http://libvirt.org/schemas/domain/qemu/1.0'>
```

This is a very fast way to access the memory but it is also considered buggy and may crash the VM LibVMI (2013). So coming up with a good way of providing VMI capability in SDN context can be a useful research area which will be explored further in near future.

5.2.1 Surrogate as Service: vmBox Security

Another prospective scope for future is to use ARM development board for security in Mobile Vehicular Environment. The collaboration between Autonomous Vehicle in real time will put each device in a high risk position. The adversary can target open security vulnerability. For instance the speed control of vehicle can be manipulated by exploiting OS level vulnerability since the code/computing from a Vehicle in platoon is offloaded to Surrogate containers , if the Vehicle is compromised.

Other methods of attack such as hack attacks and shack attacks have been discussed in detail in Inc. (2013). We propose a security verification approach on each host device.

At hardware level, ARM TrustZone(Security Extension) will provide split mode feature of secure and non-secure environments. This will allow boot time verification of trusted components and runtime hypervisor protection LENGYEL *et al.* (2014). Execution of code on virtual CPUs is strictly managed by Configuration Register.

The core mode called monitor mode will allow context switch between two virtual processors running in time sliced fashion depending upon the offloaded code from Vehicle. Once the request reaches the monitor mode on surrogate, trusted software would route the request accordingly as can be seen in 5.2.

For instance if normal computing is required by Vehicle - IRQ (normal interrupt request) , this will be handled by Normal World on surrogate as shown in figure. The remote user will have privileges of read/write in this mode. However if operation requires access to secure world resources monitor mode would switch to Secure world, and remote user/adversary will have lower read privileges only. This will prevent adversary in platoon from masking secure world interrupts FIQ(Fast Interrupt Requests) modifying Status Registers e.g. CPSR(Current Program Status Register) Wu *et al.* (2015).

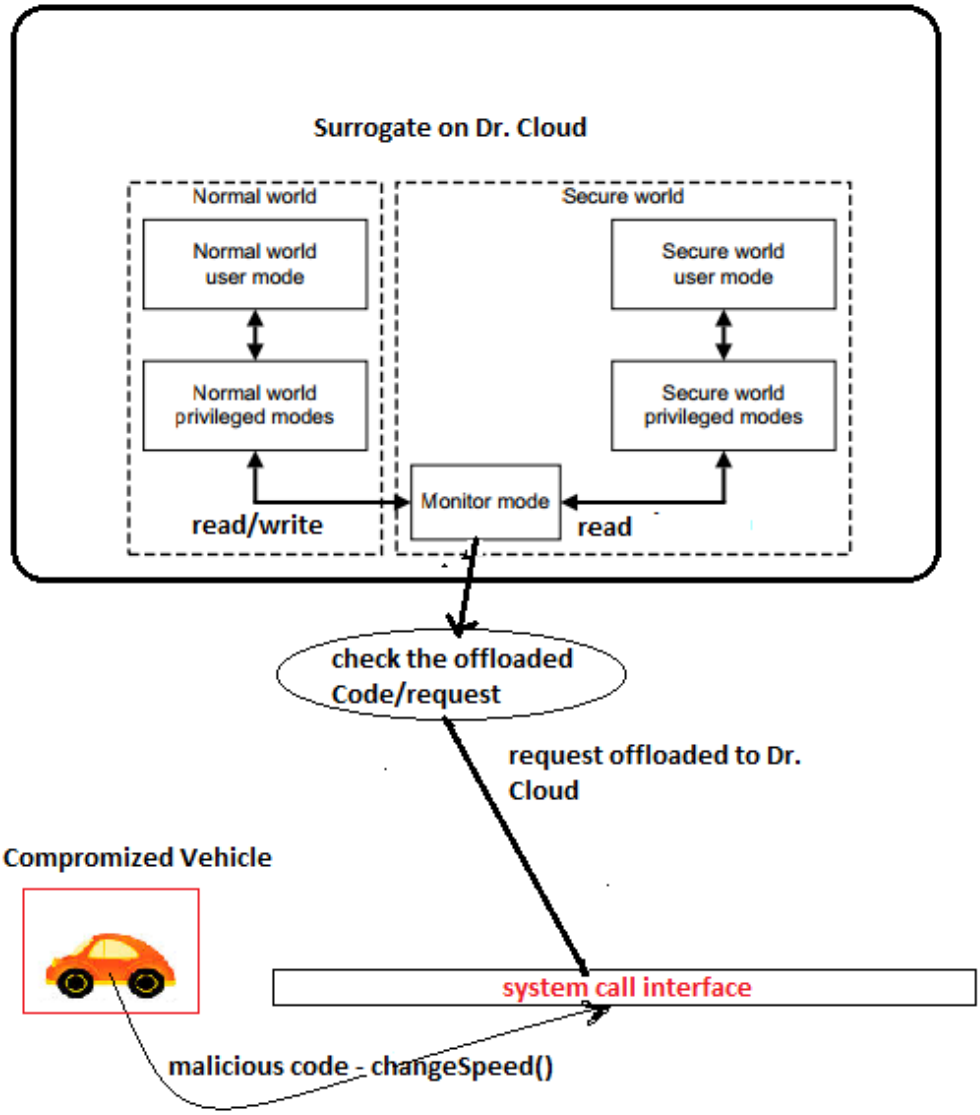

Figure 5.2: Security Vulnerability on a Surrogate Device .

ARM, *ARM Architecture Reference Manual ARM v7-A and ARM v7-R*, chap. A1.1 (ARM Inc., 2013).

Dall, C. and J. Nieh, "Kvm for arm", in "Proceedings of the 12th Annual Linux Symposium", (OLS, 2010).

Dall, C. and J. Nieh, "Kvm/arm: Experiences building the linux arm hypervisor", Tech. rep., Department of Computer Science, Columbia University, URL `http://academiccommons.columbia.edu/catalog/ac%3A162668` (2013a).

Dall, C. and J. Nieh, "Supporting kvm on the arm architecture", LWN (2013b).

Dall, C. and J. Nieh, "The design and implementation of the linux arm hypervisor", in "Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems", p. 333348 (ALPOS, 2014).

Dmytriyenko, D., "meta-ti layer", URL `http://git.yoctoproject.org/cgit/cgit.cgi/meta-ti/` (2014).

DroneCode, "Ardu pilot", URL `http://ardupilot.com/` (2014).

Emmerich, P., D. Raumer, F. Wohlfart and G. Carle, "Performance characteristics of virtual switching", in "Cloud Networking (CloudNet) 2014 IEEE 3rd International Conference", (IEEE, 2014).

Feamster, N., J. Rexford and E. Zegura, "The road to sdn: An intellectual history of programmable networks", Tech. rep., Princetion University, URL `http://queue.acm.org/detail.cfm?id=2560327` (2013).

Foundation, L., "Xen arm with virtualization extensions whitepaper", Tech. rep., XEN, URL `http://wiki.xenproject.org/wiki/Xen_ARM_with_Virtualization_Extensions_whitepaper` (2013).

Inc., A., "Arm. arm security technology", (2013).

Inc., A., "Arm juno development platform", URL `http://www.arm.com/products/tools/development-boards/versatile-express/juno-arm-development-platform.php` (2014a).

Inc., A., "Juno arm development platform", URL `http://www.arm.com/products/tools/development-boards/versatile-express/juno-arm-development-platform.php` (2014b).

Inc., A., "Porting to arm 64-bit", URL `http://community.arm.com/docs/DOC-8453` (2014c).

InSignal, "Arndale 5 base board system reference manual", Tech. rep., Arndale, URL `http://www.arndaleboard.org/wiki/downloads/supports/BaseBoard_Specification_Arndale_Ver1_0.pdf` (2013a).

InSignal, "Samsung exynos 5250 dual", URL `http://www.notebookcheck.net/Samsung-Exynos-5250-Dual-SoC.86886.0.html` (2013b).

LENGYEL, T. K., T. KITTEL, J. PFOH and C. ECKERT, "Multi-tiered security architecture for arm via the virtualization and security extensions.", in "25th International Workshop onDatabase and Expert Systems Applications", (IEEE, 2014).

Liangli, M., C. Yanshen, S. Yufei and W. Qingyi, "Virtualization maturity reference model for green software", in "Control Engineering and Communication Technology", (ICCECT, 2012).

LibVMI, "Virtual machine introspection", URL `http://www.libvmi.com/` (2013).

Lim, Z. S., "Setting up kvm", URL `https://wiki.linaro.org/ZiShenLim/sandbox/SettingUpKVM` (2013).

Makita, T., "Virtual switching technologies and linux bridge", Tech. rep., NTT Open Source Software Center, URL `http://events.linuxfoundation.org/sites/events/files/slides/LinuxConJapan2014_makita_0.pdf` (2014).

Nexus9, "Google nexus 9", URL `https://www.google.com/intl/en_us/nexus/9/` (2014).

Oltsik, J., "A multitude of mobile security issues", URL `http://www.esg-global.com/blogs/a-multitude-ofmobile-security-issues/` (2010).

over ARM, X., "Xen arm with virtualization extensions/omap5432 uevm", URL `http://wiki.xenproject.org/wiki/Xen_ARM_with_Virtualization_Extensions/OMAP5432_uEVM` (2014).

Papaux, G., D. Gachet and W. Luithardt, "Processor virtualization on embedded linux systems", in "Education and Research Conference (EDERC), 2014 6th European Embedded Design", (IEEE, 2014).

Peng, W., F. Li, X. Han, Keesook J.and Zou and J. Wu, "T-dominance: Prioritized defense deployment for byod security", in "Communications and Network Security (CNS) Conference", (IEEE, 2013).

Project, Y., "meta-virtualization layer", `https://git.yoctoproject.org/cgit/cgit.cgi/meta-virtualization/` (2014a).

Project, Y., "Yocto project development manual", URL `http://www.yoctoproject.org/docs/1.6.1/dev-manual/dev-manual.html` (2014b).

Samsung, "Samsung chromebook", URL `http://www.samsung.com/us/computer/chromebook` (2013).

SpA, M., "New security perspectives around byod", in "Broadband, Wireless Computing, Communication and Applications (BWCCA", (IEEE, 2012).

Systems, V. O., "Kvm virtualization on arndale development board", URL `http://www.virtualopensystems.com/en/solutions/guides/kvm-virtualization-on-arndale/` (2013).

Systems., V. O., "svirt security for kvm virtualization on omap5 uevm", URL `http://www.virtualopensystems.com/en/solutions/guides/kvm-svirt-omap5/?vos=tech` (2013).

Systems, V. O., "svirt security for kvm virtualization on omap5 uevm", URL `http://www.virtualopensystems.com/en/solutions/guides/kvm-svirt-omap5/` (2013).

Technology, M., "Bring your own device", URL https://mti.com/Portals/0/Documents/White%20Paper/MTI_BYOD_WP_UK.pdf (2012).

Tootoonchian, A. and Y. Ganjali, "Hyperflow: A distributed control plane for openflow", in "INM/WREN'10 Proceedings of the 2010 internet network management conference on Research on enterprise networking", (IEEE, 2010).

Uno, A., "Uno rev 3", URL http://www.arduino.cc/en/Main/ArduinoBoardUno (2011).

vSwitch, O., "Kvm on open vswitch", URL http://git.openvswitch.org/cgi-bin/gitweb.cgi?p=openvswitch;a=blob_plain;f=INSTALL.KVM;hb=HEAD (2013).

World, P., "Windows 9 on arm", URL http://www.pcworld.com/article/2598372/windows-9-on-arm-could-kill-the-desktop-merge-windows-phone-and-windows-rt.html (2014).

Wu, H., D. Huang, Y. Xin, A. Chowdhary and Z. Wang, "Drive-in-the-cloud(dr.cloud): Surrogate service for autonomous vehicles", in "In proceedings USENIX Hot Cloud", (USENIX, 2015).

Zhauniarovich, Y., G. Russello, M. Conti, B. Crispo and E. Fernandes, "Moses: Supporting and enforcing security profiles on smartphones", in "Dependable and Secure Computing, IEEE Transactions", (IEEE, 2014).

APPENDIX A

DEVELOPMENT PHASE ERROR LOG

In this section we will discuss the Challenges Faced during the development and some of the techniques employed to resolve these challenges so that future build process can make use of this development experience for a smooth build.

- Some of the gcc libraries are dependent upon python libraries so all the python dependencies for the development environment need to be installed for gcc to build up properly A.1.

  ```
  apt-get install python-dev
  ```

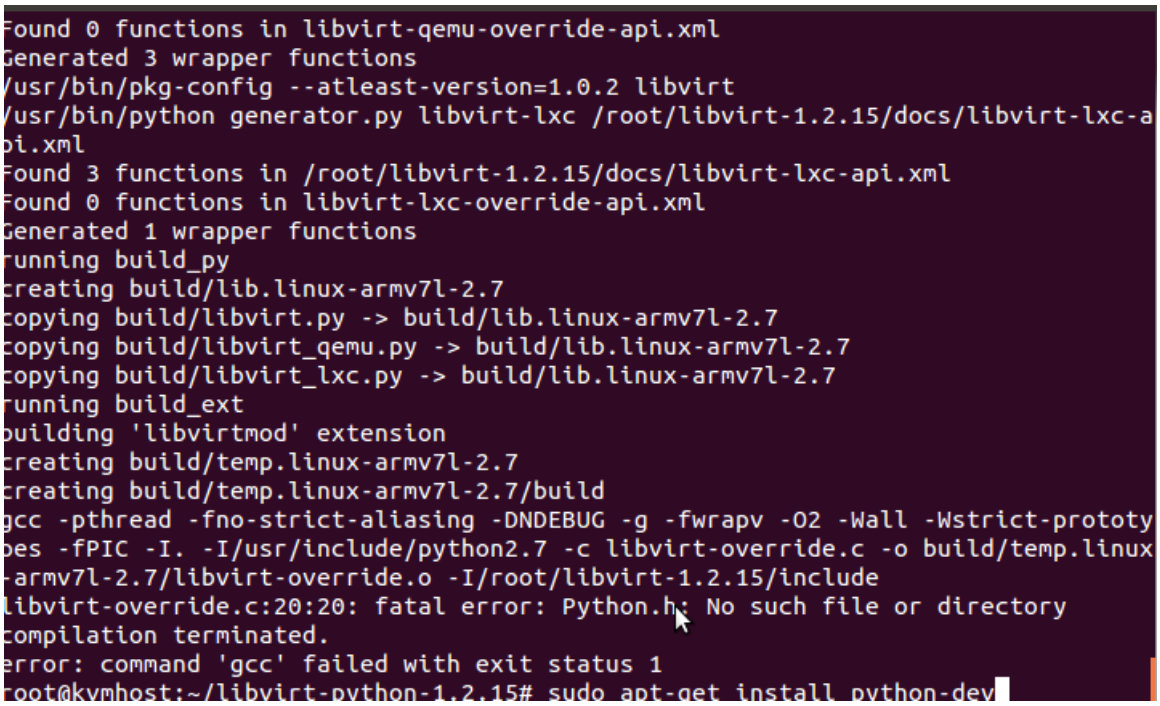

Figure A.1: Gcc Failure During Build.

- Openssh Errors during the build of Open vSwitch was another issue A.2. We need to make sure this package is present and it's version is compatible with Open vSwitch.
- Libvirt Daemon needs the firewall modules to be enabled during kernel build. We need to go to

  ```
  \lib\modules\uname -a\build
  ```

  and enable the corresponding firewall modules as external modules that are one dependencies of libvirt daemon.
- Modules bc needs to be present for kernel build.
- Kernel boot parameters need to be proper to avoid A.4, the device tree start address should not overlap with address range of uImage.

```
Setting up python-minimal (2.7.5-5ubuntu3) ...
Setting up python (2.7.5-5ubuntu3) ...
Setting up python-chardet (2.0.1-2build2) ...
Setting up python-six (1.5.2-1) ...
Setting up python-urllib3 (1.7.1-1build1) ...
Setting up python-requests (2.2.1-1) ...
dpkg: dependency problems prevent configuration of ssh:
 ssh depends on openssh-server (>= 1:6.6p1-2ubuntu1); however:
  Package openssh-server is not configured yet.

dpkg: error processing package ssh (--configure):
 dependency problems - leaving unconfigured
Setting up tcpd (7.6.q-25) ...
Setting up ssh-import-id (3.21-0ubuntu1) ...
Processing triggers for libc-bin (2.19-0ubuntu6) ...
Processing triggers for ca-certificates (20130906ubuntu2) ...
Updating certificates in /etc/ssl/certs... 164 added, 0 removed; done.
Running hooks in /etc/ca-certificates/update.d....done.
Processing triggers for ureadahead (0.100.0-16) ...
Errors were encountered while processing:
 openssh-server
 ssh
E: Sub-process /usr/bin/dpkg returned an error code (1)
```

Figure A.2: Open vSwitch Build Failure.

```
root@kvmhost:~# libvirtd &
[1] 20848
root@kvmhost:~# 2015-05-22 05:36:40.494+0000: 20859: info : libvirt version: 1.2
.15
2015-05-22 05:36:40.494+0000: 20859: error : virFirewallApplyRuleDirect:732 : in
ternal error: Failed to apply firewall rules /sbin/iptables --table filter --ins
ert INPUT --in-interface virbr0 --protocol tcp --destination-port 67 --jump ACCE
PT: iptables: No chain/target/match by that name.
```

Figure A.3: Libvirtd Build Issue.

- Modules stp.ko, llc.ko, bridge.ko and vxlan.ko need to be present for openvswitch.ko module to be installed. Also the device source tree and kernel should be compatible.
- Make sure that qemu parameters for bootup are in accordance with the parameters give in script in implementation Chapter. Otherwise the kernel may crash and build would have to be started anew A.5.

```
SD/MMC found on device 0
reading uEnv.txt
** Unable to read file uEnv.txt **
** File not found /boot/zImage **
mmc1(part 0) is current device
SD/MMC found on device 1
** Unrecognized filesystem type **
Failed to mount ext2 filesystem...
** Unrecognized filesystem type **
U-Boot# printenv
```

Figure A.4: Kernel Boot Error.

```
root@kvmhost:~# ./tunctl.sh
Set 'tap0' persistent and owned by uid 0
root@kvmhost:~# ./qemu_script_ubuntu.sh
[    0.000000] Booting Linux on physical CPU 0x0
[    0.000000] Linux version 3.8.0-rc4+ (xavier@ubuntu) (gcc version 4.6.3 (Ubun
tu/Linaro 4.6.3-1ubuntu5) ) #4 SMP Wed Mar 13 14:29:04 PDT 2013
[    0.000000] Machine: ARM-Versatile Express, model: V2F-2XV6 Cortex-A15x2 SMM
[    0.000000] bootconsole [earlycon0] enabled
[    0.000000] cma: CMA: reserved 16 MiB at 9f000000
[    0.000000] ------------[ cut here ]------------
[    0.000000] WARNING: at arch/arm/mach-vexpress/v2m.c:431 v2m_dt_init_early+0x
58/0x80()
[    0.000000] ---[ end trace 1b75b31a2719ed1c ]---
[    0.000000] vexpress: DT HBI (217) is not matching hardware (237)!
[    0.000000] PERCPU: Embedded 5 pages/cpu @c08ee000 s6656 r0 d13824 u32768
[    0.000000] Kernel command line: earlyprintk console=ttyAMA0 mem=512M root=/d
ev/vda rw ip=dhcp --no-log
[    0.000000]          virtio_mmio.device=1M@0x4e000000:74:0 virtio_mmio.device
=1M@0x4e100000:75:1
[    0.000000] PID hash table entries: 2048 (order: 1, 8192 bytes)
[    0.000000] Dentry cache hash table entries: 65536 (order: 6, 262144 bytes)
[    0.000000] Inode-cache hash table entries: 32768 (order: 5, 131072 bytes)
[    0.000000] __ex_table already sorted, skipping sort
```

Figure A.5: Kernel Panic Due to Improper Qemu Parameters.

APPENDIX B

OTHER POTENTIAL PLATFORMS FOR DEVELOPMENT

- **OMAP 5432** sVirt feature of protecting the KVM ARM guests has been discussed on TI-OMAP 5432. This can be another suitable platform for testing SDN solution in ARM virtualized environment since it offers additional security provided by libvirt. A stable bootloader and way to boot Ubuntu or Fedora on this board needs to be explored.

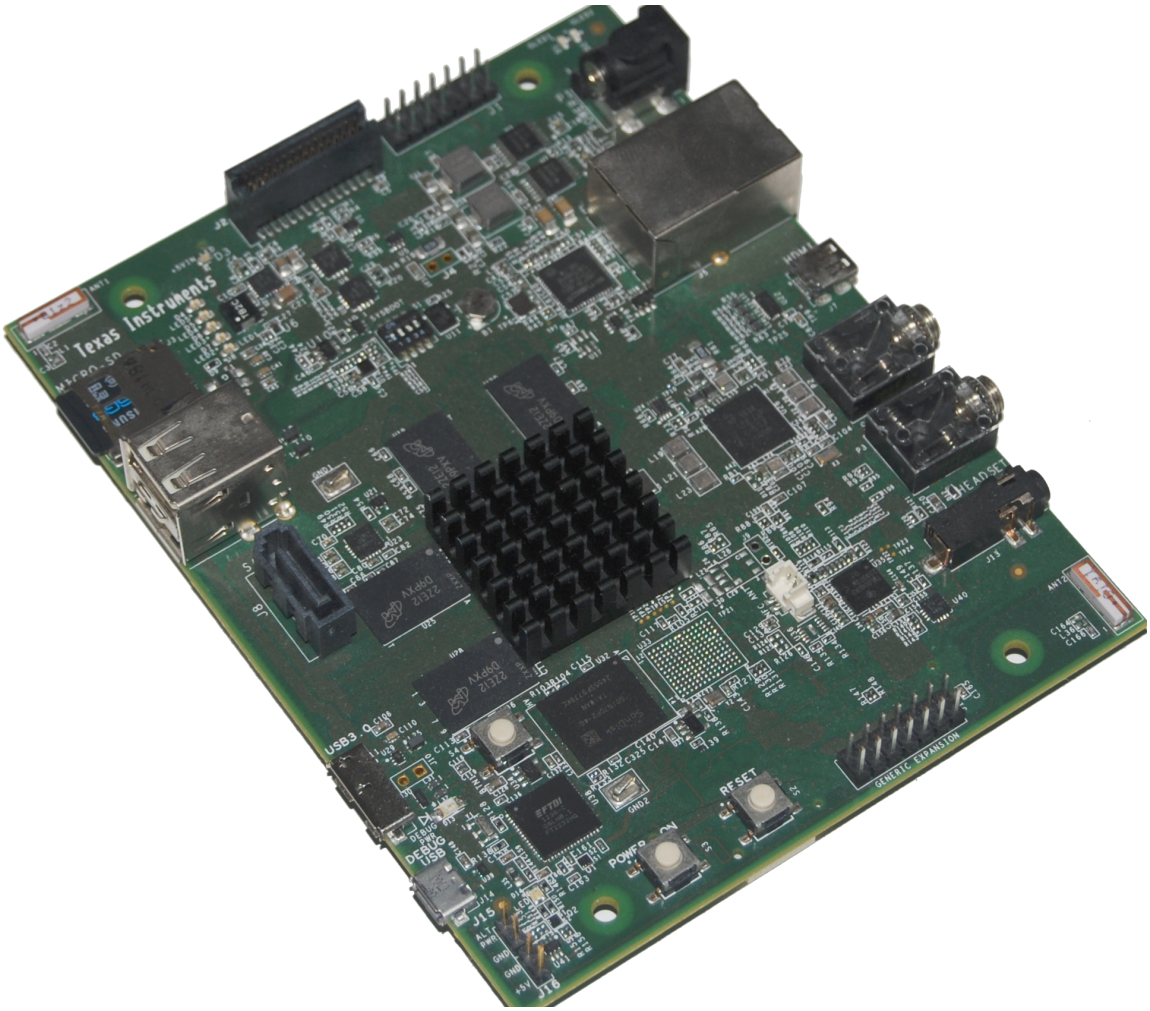

Figure B.1: TI Omap 5432 Development Board 'Systems. (2013)'

- **TI OMAP 5432 XEN SOLUTION** A stable bootloader and kernel is required, solution discussed for TI-5432 doesn't work currently.
- **ARM Fast Models** Versatile Express simulation platform is a good model that can be tried for eperimentation.
- **ARM Juno** It comes with ARMv8 and it is a 64-bit CPU. Currently price of this board is 4200$and if license for Developer Studio is included, it costs 700$almost 35x price of arndale Exynos5250.
- **Samsung Chromebook** Although it is ARMv8 CPU, many developers face known issues in Virtualization over this platform.

Figure B.2: Arm Juno Development Board 'Inc. (2014a)'

- **Nexus 9** A good, and powerful model providing ARMv8, has to be explored further.

Figure B.3: Samsung Chromebook with 64 Bit ARMv8 'Samsung (2013)'

Figure B.4: Google Nexus 9 with 64 Bit ARMv8 'Nexus9 (2014)'