EVALUATION OF UML BASED WIRELESS NETWORK VIRTUALIZATION

BY SHRUTI SINGHAL

A thesis submitted to the

Graduate School—New Brunswick

Rutgers, The State University of New Jersey

in partial fulfillment of the requirements

for the degree of

Master of Science

Graduate Program in Electrical and Computer Engineering

Written under the direction of

Professor D. Raychaudhuri

and approved by

_____

_____

_____

New Brunswick, New Jersey

October, 2007

ABSTRACT OF THE THESIS

Evaluation of UML based Wireless Network Virtualization

by Shruti Singhal

Thesis Director: Professor D. Raychaudhuri

Virtualization of wireless networks is recognized to be a difficult problem due to the fact that radios interact with their neighbors at various layers of the protocol stack, making strict isolation of virtual networks ("or slices") quite challenging. The goal of virtualization is to support concurrent experiments, both long-running services as well as short-term experiments on shared wireless network. In a wireless network, the radio resources that can be shared and hence virtualized are in time, space and frequency. Efforts have been going on to modify the ORBIT control structure to accommodate different forms of virtualization including VMAC, SDMA, FDMA and TDMA. Among different possible wireless virtualization techniques, this work is focused on allowing a node to run more than one experiment simultaneously using different frequencies i.e. Frequency Division Multiplexing (FDM). Each node in the ORBIT test bed is provided with two physical wireless cards. FDMA virtualization is achieved by running two concurrent User Level Operating Systems (ULOS) on each node and providing each operating system access to a radio card. Thus an experimental end user would view a single node as two virtual nodes, each equipped with one wireless card.

Experimental results are provided to compare the performance of a virtualized radio node with the non virtualized one for basic point-to-point experiments using TCP and UDP. Bounds on performance metrics of throughput, delay and jitter are determined

and cross-coupling effects between two virtualized experiments are examined. We also look at transient behavior associated with sudden changes in traffic on one of the virtual networks. Finally, the uncertainty in performance measurements for a few typical usage scenarios is investigated, leading to guidelines for use of virtualized radio nodes for simultaneous ORBIT experiments.

Acknowledgements

It is my pleasure to express my sincere gratitude to my advisor Professor D. Raychaudhuri for providing me with the opportunity to work on this project. He has been a great source of guidance and support throughout and I thank him for sharing his expert competence and for his time, patience and understanding. I consider myself fortunate and honored to have worked with him.

I wholeheartedly thank Ivan Seskar for his extensive help and advice at various stages of this thesis work. I cannot thank him enough for his help as this work would not have been possible without his guidance and expert comments and analysis.

I continue to be deeply indebted to Dr. George Hadjichristofi who has literally walked me through my thesis journey. His comments, motivation and encouragement have been instrumental in the completion of this work. There could not have been a better mentor.

Special thanks to Dr. Dekai Li from Thomson CR, who unknowingly helped me out with an implementation issue in my thesis. I enjoyed the company and support of many fellow WINLAB students including Sumathi Gopal, Haris Kremo, Sachin Ganu, Kishore Ramachandran, Gautam Bhanage, Sanjit Kaul and specially Rajesh Mahindra whose presence and great friendship made working on this project all the more fun.

For all my friends whom I fail to mention here, I'm thankful for their friendship and support throughout my life.

Finally, my deepest regards for my parents and grandmother for their inspiration and support through the years and I humbly dedicate this work to them.

Dedication

This work is dedicated to my late grandfather, *Lt. Jagmohan Lal Singhal* who taught me the value of education and always encouraged me to do my best

Table of Contents

List of Tables

List of Figures

List of Abbreviations

AP	Access Point
BSS	Basic Service Set
BSSID	Basic Service Set Identification
CSMA-CA	Carrier Sense Multiple Access - Collision Avoidance
CPU	Central Processing Unit
DCF	Distributed Coordination Function
DIFS	DCF Interframe Space
DSSS	Direct Sequence Spread Spectrum
ESS	Extended Service Set
ESSID	Extended Service Set Identification
FDMA	Frequency Division Multiple Access
FH	Frequency Hopping
GENI	Global Environment for Network Innovations
IEEE	Institute of Electrical and Electronics Engineers
IP	Internet Protocol
LAN	Local Area Network
MAC	Medium Access Control
MT	Mobile Terminal
NSF	National Science Foundation
OFDM	Orthogonal Frequency Division Multiplexing
OML	ORBIT Measurements Library
ORBIT	Open Access Research Testbed for Next Generation Wireless Networks
OS	Operating System
OTG	ORBIT Traffic Generator
OTR	ORBIT Traffic Receiver
PHY	Physical Layer
RTP	Real Time Protocol
SDMA	Space Division Multiple Access
TCP	Transmission Control Protocol
TDMA	Time Division Multiple Access
UDP	User Datagram Protocol
ULOS	User Level Operating System
UML	User Mode Linux
VINI	Virtual Network Infrastructure
VLC	Video Lan Client
VM	Virtual Machine
VMAC	Virtual Medium Access Control
VMM	Virtual Machine Monitor
WLAN	Wireless Local Area Networks

Chapter 1

Introduction

1.1 Overview of the Problem

ORBIT [1] is a 400 node wireless test bed, sponsored by NSF [2] and designed to support evaluation of protocols and applications in real world settings and also to achieve reproducibility of experiments. Extending the concept of test-bed for protocol and software investigation, NSF has instantiated the GENI [3] project that intends to provide a flexible and programmable shared experimental infrastructure integrating wired and wireless networks. The project GENI aims at

- Virtualization of the wireless network (ORBIT) resources to provide capabilities for simultaneous support of multiple concurrent experiments ("slices") on the same set of radio devices.
- Integration of wired and wireless test bed control and management so that experimenters can use a single programming interface and experiment methodology.

This work concentrates on the wireless network virtualization aspect of the GENI project. The goal of virtualization is to support concurrent experiments, both long-running services as well as short-term experiments on shared wireless network. In a wireless network, the radio resources that can be shared and hence virtualized are in time, space and frequency. Efforts have been going on to modify the ORBIT control structure to accommodate different forms of virtualization[1] including VMAC, SDMA, FDMA and TDMA. Among different possible wireless virtualization techniques, our current work is focused on allowing a node to run more than one experiment simultaneously (essentially frequency division multiplexing) using different radio cards. Each node in the ORBIT test bed is provided with two physical wireless cards. One way of

virtualization is by running a maximum of two user level operating systems on each node and providing each operating system access to a radio card. So to a user, a single node would appear as two nodes with one physical wireless card each.

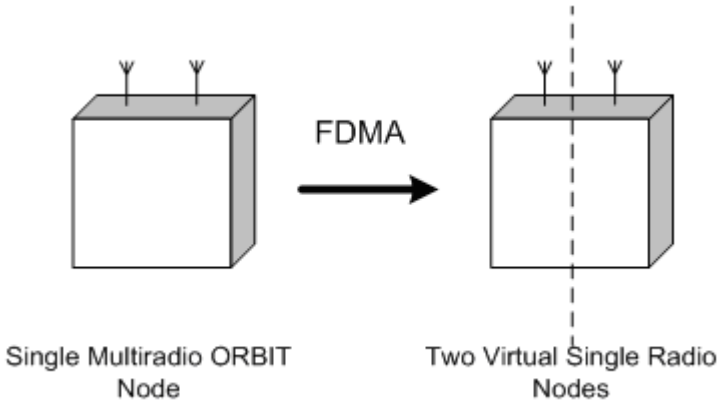

Figure 1.1: Virtualization of ORBIT Node

User level operating systems (ULOS) [4] are being widely used as an OS virtualization technique as it provides secure and isolated systems on shared hardware. These operating systems run as user level processes over the host kernel. One of such ULOS is User Mode LINUX (UML) [5]. The UML operating system runs as a regular user process or "guest" on an operating system. We call the operating system that accommodates UML the 'host' operating system. Its hardware is virtual and is constructed from the resources provided by the host. The UML is structured exactly the same as a HOST kernel. The functions in a UML are implemented by the means of system calls.

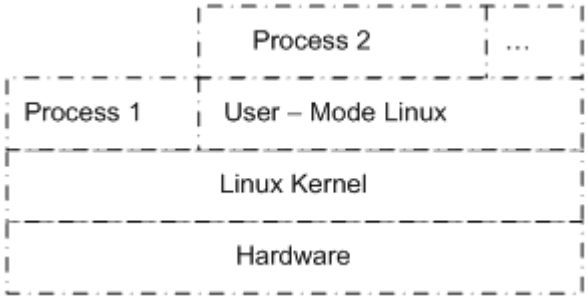

Figure 1.2: UML Architecture

We study the performance of ORBIT wireless test bed by running two instances of (UML) on the nodes. A UML offers the advantages of a typical virtual machine and

is easy to install and configure. On the other hand, UML suffers network penalties like decrease in throughput and increase in network latencies as they implement kernel functions using system calls to the host kernel, which introduces comprehensive overheads in the system. Also, while running multiple UML's on a single host, each UML process is context switched like any other user level process.

1.2 Motivation

The way to use ORBIT right now is to time share the GRID (400 wireless device network). But due to the extensive GRID utilization, there is a need for virtualization of the GRID resources so that multiple experimenters can use the GRID at the same time. Also investigation of virtualization of wireless networks is of great importance as it is a key component of the GENI project [3]. Virtualization of wireless networks is recognized to be a difficult problem due to the fact that radios interact with their neighbors at various layers of the protocol stack, making strict isolation of virtual networks ("or slices") quite challenging. Efforts are being put in to investigate the performance of the wireless network with different virtualization techniques so as to come up with a set of techniques that can be deployed on the wireless networks as a part of GENI project.

1.3 Objectives

In this thesis we investigate the performance of ORBIT nodes with multiple instances of UML running simultaneous experiments and each UML acting as a complete operating system and sharing the underlying hardware resources of the host kernel. In this work we are limited by running two UML instances per node as each node has two wireless cards.

Experimental results are provided to compare the performance of a virtualized radio node with the non virtualized one for basic point-to-point experiments using TCP and UDP. Bounds on performance metrics of throughput, delay and jitter are determined and cross-coupling effects between two virtualized experiments are examined. We also

look at transient behavior associated with sudden changes in traffic on one of the virtual networks. Finally, the uncertainty in performance measurements for a few typical usage scenarios is investigated, leading to guidelines for use of virtualized radio nodes for simultaneous ORBIT experiments.

1.4 Previous Work

Virtual machine (VM) techniques have been explored in earlier efforts in wired networks. PlanetlabOS [6] enables a number of users to share the same hardware, providing a virtualized userlevel working environment to each user. However, PlanetlabOS, implemented by using Linux vserver [7] and SILK [8], is not a fully virtualized OS because its network subsystem is not virtualized. PL-VINI, an implementation of VINI [9] on PlanetLab uses UML to run multiple experiments on the same PlanetLab nodes.

[10]Quantifies the performance of UML in case of wired networks. It analyzes the packet-processing overheads in a ULOS through a methodical analysis of packet processing in the TCP/IP stack. Network throughput and latencies with UML are compared with those of native Linux. In our work we compare and quantify the performance of UML with those of native Linux in case of wireless networks.

1.5 Thesis Organization

Rest of the thesis is organized as follows. Chapter 2 presents an overview of the 802.11a protocol. In Chapter 3, we give an overview of Virtualization and present different schemes for wireless virtualization. Chapter 4 explains the concept and operation of UML and virtual networking. Chapter 5 is dedicated to the experiment setup and results. In Chapter 6, we summarize the main contributions and give possible directions for future work.

Chapter 2

Overview of IEEE 802.11a

IEEE 802.11 [11] is a standard for WLANs designed to provide high-speed data communication between portable devices. It is intended to allow flexible wireless networks to be created within local area without the need for the wired infrastructure and it can be used as an extension of a wired LANs. The 802.11 standard defines both the physical layer and the Medium Access Control (MAC) layer. The IEEE 802.11 standard was adopted in 1996. Since then, several extensions to the standard have been developed, and more are emerging. This section provides an overview of the original 802.11 standard and its extensions.

2.1 Network architecture

The station is the most basic element of the 802.11 WLANs. A station is any device that contains the functionality of the 802.11 protocol. The basic service set (BSS) is the basic building block of 802.11 WLAN and consists of two or more stations. Figure 2.1 illustrates the concept of the BSS when applied to two types of networks defined in the IEEE 802.11 standard: independent and infrastructure. The ovals used to depict a BSS illustrate the coverage area within which the member stations of the BSS may remain in communication. The Independent BSS, often referred as an ad-hoc, is standalone self-configuring network, providing direct communication between stations. The Infrastructure BSS uses fixed location access points (AP) to provide connectivity to stations.

In order to extend the operational range of a BSS, the 802.11 standard defines an Extended Service Set (ESS), as illustrated in Figure 2.2. An ESS consists of multiple BSS interconnected by distribution system, wired or wireless backbone network.

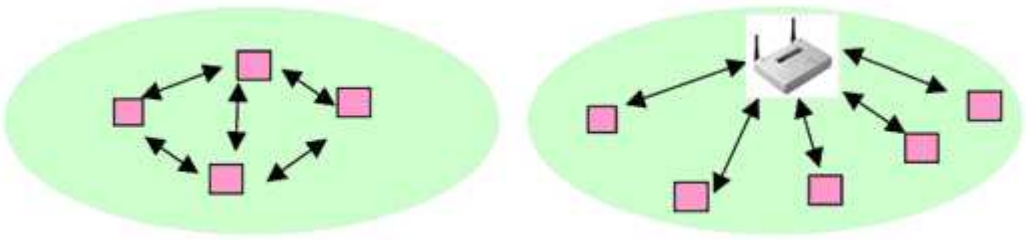

Figure 2.1: Independent BSS and Infrastructure BSS

ESS can be interconnected with other wired or wireless networks, allowing stations within this ESS access to other networks' resources. Each BSS and ESS has its unique identification called BSSID and ESSID respectively, which are required to implement addressing.

To join an Infrastructure BSS, a station must select an AP and associate with it. The association service creates a mapping between the station and the AP that can be provided to the distribution system. The stations can then send and receive messages via the associated AP. The dissociation service terminates an existing connection. The integration service connects the 802.11 WLAN to other LANs, including one or more wired LANs or 802.11 WLANs.

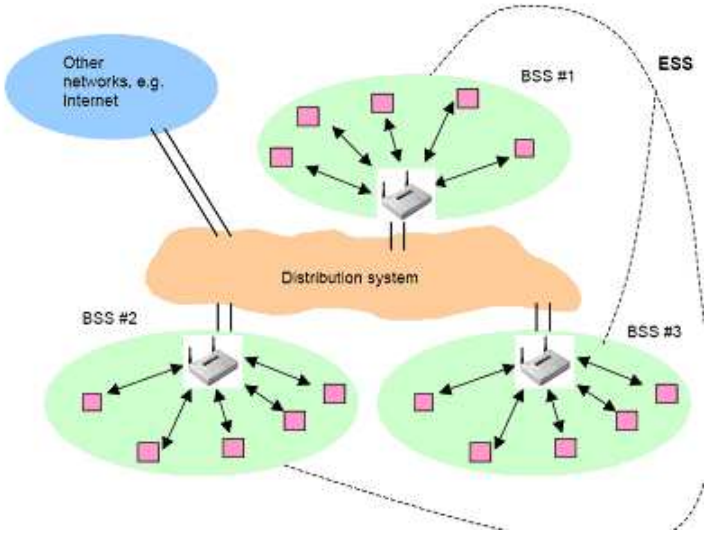

Figure 2.2: Extended Service Set

Parameter	*Value*
Sampling rate	20 MHz
Useful symbol duration	3.2 secs
Guard interval	0.8 secs
Total symbol duration	4 secs
Number of data sub-carriers	48
Number of pilot sub-carriers	4
FFT Size	64

Table 2.1: 802.11a Phy OFDM Parameters

2.2 802.11a Physical layer

The 802.11a standard [11] is intended for the 5 GHz license-free UNII band and provides data rates up to 54 Mbps. The 5 GHz band has an advantage of large bandwidth allocated for the unlicensed operations. The 802.11a is based on Orthogonal Frequency Division Multiplexing (OFDM) modulation, which allows achieving higher data rates within about the same channel bandwidth as 802.11b. OFDM is a multicarrier transmission technique. The OFDM signal consists of multiple subcarriers, each one being modulated by a low rate data stream. Low rate data streams are formed by demultiplexing one high data rate stream. Subcarriers are kept orthogonal, so data symbols modulated on these subcarriers can be recovered without mutual interference. Since the symbol rate on each subcarrier is slower than the original data rate, the OFDM technique is particularly efficient in time dispersive environments.

Data for transmission is supplied to the PHY layer in the form of an input protocol data unit train. This is then input to a scrambler that prevents long runs of 1's and 0's in the data from being input to the remainder of the modulation process. The scrambled data is input into a convolutional encoder. The encoder consists of a rate 1 2 mother code and subsequent puncturing to obtain higher rates such as 2 3 and 3 4. At this point, the coded data is divided into blocks of length 48, which is the number of data sub-carriers in an OFDM symbol. The coded data corresponding to one OFDM sym-bol is interleaved in order to prevent error bursts from being input to

the convolutional decoding process at the receiver. This interleaving across frequency bands provides diversity and better performance against frequency selective fading, in addition to the coding protection. The interleaved data is subsequently mapped to an OFDM symbol according to [11].

2.3 802.11a MAC Layer

The basic MAC mechanism specified in IEEE 802.11a is based on distributed coordination function (DCF) that allows for sharing of the wireless channel through the use of CSMA/CA. The DCF is implemented in every mobile terminal (MT) and access point (AP).

An MT with a pending data packet has to monitor the channel activity. If the channel is idle for a period of time equal to the DCF Inter Frame Space (DIFS), the MT transmits. Otherwise, if the channel is sensed as busy (either immediately or during DIFS), the MT continues to monitor the channel until the channel is sensed idle for a DIFS. At this point, the MT generates a random back-off time before transmitting, to minimize the probability of a collision with packets being transmitted by other MT's. The time is slotted and the random back-off time is an integer value that corresponds to a number of time slots.

2.4 802.11 And Virtualization

The 802.11 protocol allows virtualization of its physical layer by allowing networks, each with a different "ESSID", to use the same physical channel simultaneously. But the virtualization considered in this work is different as we use orthogonal physical channels for different wireless networks. But even though the physical channels are orthogonal, the task of virtualization is challenging as the radios interact with each other at various layers of protocol stack, thus making the two networks somewhat dependant on each other. The next chapter presents with the basic concepts of virtualization and explains different ways to achieve wireless network virtualization.

Chapter 3

Overview of Virtualization

Virtualization can be defined as "a technique for hiding the physical characteristics of computing resources from the way in which other systems, applications, or end users interact with those resources. This includes making a single physical resource (such as a server, an operating system, an application, or storage device) appear to function as multiple logical resources; or it can include making multiple physical resources (such as storage devices or servers) appear as a single logical resource" [3].

Virtualization in essence is creating a virtual version of any resource such as servers, operating system, storage device or network resources. It creates an external interface that hides the underlying physical hardware and resources. Virtualization can be divided into two broad categories:

1. Platform virtualization
2. Network Virtualization

3.1 Platform Virtualization

This type of virtualization involves simulation of virtual machines. On a given hardware, the "host" software (a control program), creates a simulated environment for the "guest" software. The "guest" software, which is often a complete operating system, runs as if it were the only operating system running on the underlying hardware platform. Generally, many such virtual machines can be run on one physical machine. Depending upon how the hardware virtualization is implemented, there are several types of platform virtualization:

- Emulation or Simulation: The virtual machine simulates the complete hardware,

which allows an unmodified "guest" OS to be run on a different CPU than that emulated by the guest OS. This type has been used to create software for new processors before they were physically available.

- Native virtualization: The virtual machine simulates hardware that is enough to allow an unmodified "guest" OS (one designed for the same CPU) to be run in isolation. Typically, many instances of the virtual machine can be run at the same time.

- Partial virtualization: This type of virtualization emulates multiple instances of a few hardware resources, but cannot support multiple instances of "guest" OS. One example is emulation of multiple address spaces.

- Para virtualization: The virtual machine does not necessarily simulate hardware, but instead (or in addition) offers a special API that can only be used by modifying the "guest" OS. Thus it presents a software interface to virtual machines that is similar to that of the underlying hardware. The operating system runs on top of virtual machine monitor (VMM) or a hypervisor. E.g Xen [16], VM.

- Operating system-level virtualization - User level operating system: Virtualizing at the operating system level involves enabling multiple isolated and secure virtualized OS to run on a single OS. The "guest" OS runs as a user level process on the host system. Applications running in a given "guest" environment view it as a stand-alone system. E.g. UML. For this work we are virtualizing the ORBIT node using UML's.

3.2 Network Virtualization

The basic concept of platform virtualization has been extended to network virtualization. Network virtualization is a method of combining the available resources in a network by splitting up the available resources, each of which is independent from the others and can be assigned (or reassigned) to a particular device in real time. The idea is that virtualization disguises the true complexity of the network by separating it into

manageable parts.

"Wireless Virtualization" and "slicing" have been defined by the wireless subgroup of GENI in [3]. Slice has been defined as "process of allocating a coherent subset (a slice) of GENI's physical resources to a specific experiment". Thus a slice essentially would be a collection of resources so as to form a wireless network on top of the underlying resources. Many such orthogonal slices may be allotted to different experiments at the same time, thus allowing multiple experiments to run at the same time.. Virtualization in context of slicing, would allow multiple instances of different resources to be implemented by a single node. This could be with same or different slices. Thus with our work, a single node can be shared within multiple experimenters, giving them an illusion of each having control over a separate node.

3.2.1 Challenges in Wireless Virtualization

Wireless virtualization is an interesting problem as making strictly isolated wireless networks ("slices") is very difficult. The radio resources interact with their neighbors at various points, thus isolating them completely for simultaneous usage is quite challenging.

In order to establish a wireless link, a transmitter and receiver have to be configured with appropriate channel parameters e.g. frequency, channel rate, essid, etc. Thus to support multiple experiments over a single network, multiple wireless links with different channel parameters are required at the same time. Ideally, activity over one link should not affect the performance of any other coexisting link. Thus in a nutshell, wireless network virtualization needs to address two key issues [3]:

1. Isolation: The virtualization scheme should ensure the isolation of wireless resources of experiments coexisting at the same time to ensure minimal interference among the experiments. For example two experiments should use two orthogonal 802.11 channels with different essids, so that there is negligible wireless interference between the experiments. A set of these resources would be the channel frequency, set of nodes, maximum transmit power etc. To ensure isolation, a

careful scheduling and allotment of network resources is required.

2. Monitoring: Even after a set of isolated or orthogonal resources is allotted to an experiment, there is a need for a monitor to ensure that the experiment is restricted to those resources only and not encroaching upon the resources of another experiment. For example, when a node in an experiment is transmitting at a particular frequency, no other experiment in its wireless range should have an activity on the same channel or a partially overlapping channel.

3.2.2 Techniques for wireless virtualization

Virtualization in wireless medium can be applied in a number of ways [1]:

1. Space division multiplexing (SDM): This is the simplest approach, where physical resources are partitioned in space. Every experiment is allotted a set of nodes that are far off from each other so that they have a minimal wireless impact on other experiments. This approach is possible as there is little impact of a wireless transmission beyond a certain range. Along with the node set, a restriction is required on the transmission power for each experiment as the wireless range of a transmitter depends heavily on the transmit power. Also techniques such as noise injection can be used to ensure wireless isolation between different set of nodes. A schematic of SDM is presented in Figure 3.1.

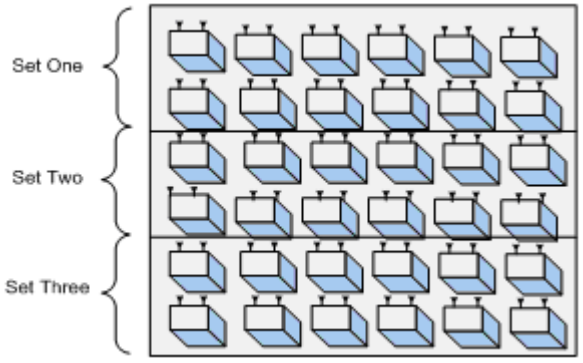

Figure 3.1: SDMA based Virtualization

2. Frequency Division Multiplexing: In this approach, the experiments are partitioned in the frequency domain. Every node in the ORBIT GRID is equipped

with multiple radio cards. Multiple virtual machines can be hosted by a single node with each virtual machine having access to one physical radio card. Thus, each experiment can be allotted one virtual machine on each node. Figure 3.2 explains FDM. Interference between the different experiments can be avoided by ensuring that the experiments are allotted orthogonal channels. The number of simultaneous experiments supported in standalone FDM scheme would be equal to the number of physical wireless cards present in each node. Also the type of experiments that can be supported would have to be investigated, since there would be a strong coupling between the virtual machines, as they share the same underlying hardware. This thesis work basically delves into this approach of wireless virtualization and addresses some of the issues involved with this approach.

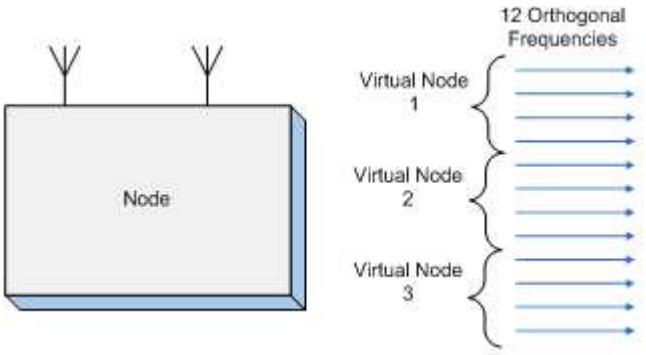

Figure 3.2: FDM based Virtualization

Another approach to FDM is that the virtual machines use the same physical wireless card, but use a different channel. A single physical 802.11a node can support up to 12 virtual nodes (since it consists of 12 orthogonal channels). Since a channel switch is required for each virtual machine they will transmit in a round robin fashion. For example an 802.11a node can be portioned into 3 virtual nodes with four frequencies per node.

3. Time Division Multiple Access: A node can also be virtualized in time domain. Multiple experiments time share the use of a particular frequency i.e. they transmit in their allotted time slots. As the number of virtual nodes using different time slots increase, their wait period between consecutive time slots also increases.

Figure 3.3 shows how multiple experiments can time share a network.

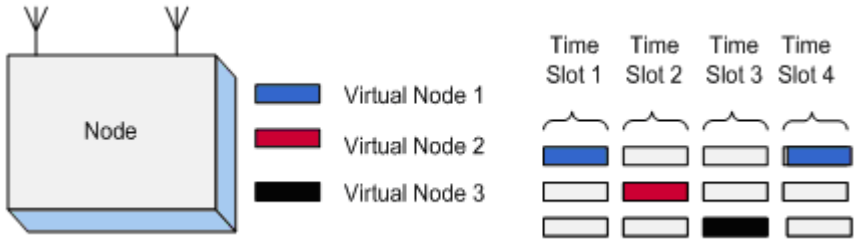

Figure 3.3: TDM based Virtualization

4. Code Division Multiple Access: CDMA refers to an approach that is similar to FDMA, except that different experiments would use different orthogonal codes for their communication link.

5. Hybrid Approaches: It is possible to visualize a number of virtualization schemes that combine one or more techniques among SDMA, FDM, TDM and CDMA.

   - Combined FDM and TDM (Frequency Hopping -FH): In this technique experiments hop through a unique sequence of frequencies (channels) over different time slots. Thus a slice for an experimenter would consist of a set of nodes that will use a fixed frequency hopping sequence (combination of frequency and time slots). Frequency Hopping mechanism is shown in Figure 3.4.

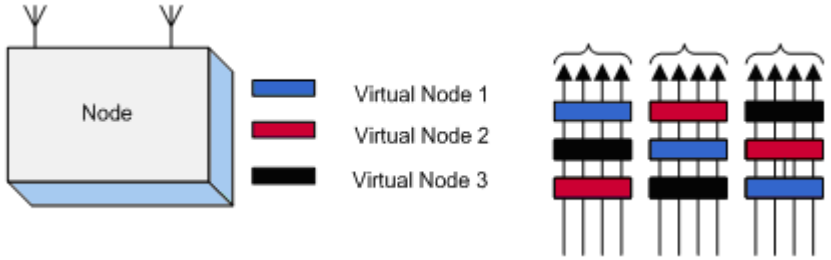

Figure 3.4: Combined FDM and TDM based Virtualization

   - Combined SDMA and FDMA: In this scheme the nodes can be partitioned along the spatial as well as frequency domain as shown in Figure 3.5. In this scheme the nodes can be partitioned even if they lie within each other's the

wireless range. Interference is avoided by making sure that every experiment is allotted a different orthogonal frequency.

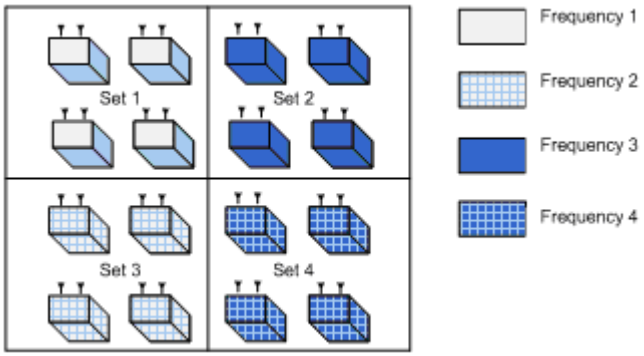

Figure 3.5: Combined SDM and FDM based Virtualization

3.2.3 Possible approaches for hardware virtualization of ORBIT nodes

A large number of machine virtualization techniques are known today each with different properties. Examples include VMWare [12], Denali [13], Xen [12], Linux Vservers [7], and User Mode Linux (UML) [4]. The Vserver approach is used in PlanetLab's current implementation [6]. In the Vserver approach, multiple virtual 'slices' share the same kernel. This technique is not suitable for ORBIT as it does not provide with sufficient isolation for experimenters (they can save their own kernel image and modify the applications and software on their kernel as suited to their experimentations). Xen 3.0 offers pure-hypervisor virtualization and allows a guest operating system to run without modification. However the wireless drivers with Xen are known to be unacceptably unstable when running inside the guest VM and they cause system-wide crashes [14]. In our work, we have considered the suitability of the UML platform to meet our needs.

PL-VINI, an implementation of VINI [9] on PlanetLab uses UML to run multiple experiments on the same PlanetLab nodes. Successful implementation of UML on Planet Lab platform has also encouraged us to use it for virtualization of ORBIT. The software details and working of UML is explained in the next chapter.

Chapter 4

Overview of UML

The UML operating system runs as a regular user process or "guest" on an operating system. We call the operating system that accommodates UML the 'host' operating system. Since each guest is an application process in user space, this approach provides supports running multiple virtual 'host' machines on a single piece of hardware, offering excellent security and safety without affecting the host environment's configuration or stability. UML architecture [4] is as shown in Figure 4.1. The UML memory, network devices and their configurations can be easily specified through command line options. The support for I/O devices such as network devices is provided through corresponding virtual devices. In our work we have set up networking for UML by creating a virtual interface in the guest connected to a TUN/TAP interface in host kernel.

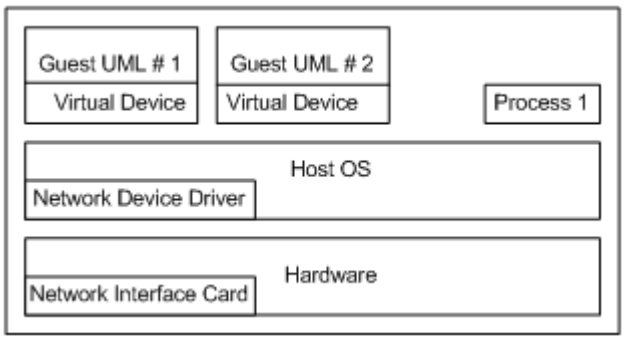

Figure 4.1: User Mode HOST operating system architecture

Full HOST API is supported in UML through the UML core. UML core uses system calls to the host HOST kernel to carry out privileged kernel functions. UML is limited by the size of the process address space for the architecture of host machine. Also resources such as CPU and memory are allocated according to the host OS' sharing and scheduling policy.

Really interesting thing about UML is that in most aspects it is identical to any normal HOST distribution. A person who is working inside a UML instance will not be aware that he is in fact working in a virtual machine rather than within the host HOST OS. One can do anything and everything in UML that he can accomplish in a normal HOST distribution. This includes tasks such as creating partitions, adding swap space, networking and so on.

4.1 Setting up a UML

Running a UML [5] requires a UML built kernel and a root file system to boot it on. Separate tools are required to set up networking in the UML. UML guest kernel is compiled especially so that it can be run from the command line. UML is limited by the size of the process address space for the architecture of host machine. For x86 it is 3 GB. "ubda" in the command line is used to give UML kernel name of file to create the virtual machine's /dev/ubda virtual block device, which will be its root file system (Analogous to /dev/hda in a HOST).

4.1.1 Networking in UML

Enabling networking in UML requires configuration of a TUN/TAP device which forms an interface between the UML and the host machine. Enabling networking also requires setting up ip forwarding and routing in the host machine. Networking in UML can be viewed in Figure 4.2.

4.1.2 TUN/TAP Device Operation

TUN/TAP provides packet reception and transmission for user space programs. It can be viewed as a simple point to point or Ethernet device, which instead of receiving packets from a physical media, receives them from user space program and instead of sending them via physical media writes them to the user space program.

When a program opens /dev/net/tun, the driver creates and registers a corresponding net device tapX. Whenever kernel sends any IPX packet to tap0, it is passed on to

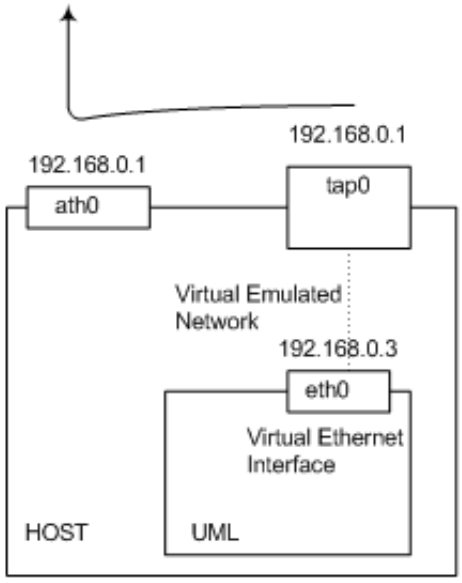

Figure 4.2: Networking in UML

the application (UML in this case). On the other side when an application writes the packet to the TAP device, the host kernel handles the packet like it came from a real physical device.

4.2 Packet Processing in UML

The UML packet processing can be divided into five layers [10] as shown in Figure 4.3:

1. The UML core boundary crossing: An application sending a packet invokes a sendto( ) system call and the control is transferred to UML kernel.

2. Packet Processing in UML core: The UML kernel executes the normal packet processing steps such as routing and forwarding decisions, network queue processing etc.

3. Virtual network device: The UML core sends the packet to the virtual network device which passes the packet to the host kernel. At this point UML has played it part and the control is transferred to the host kernel.

4. Packet processing in host kernel: The host kernel forwards the packets to a physical network device (in this case the wireless device).

5. Physical Network Device: The physical device then sends out the packet through the network interface card.

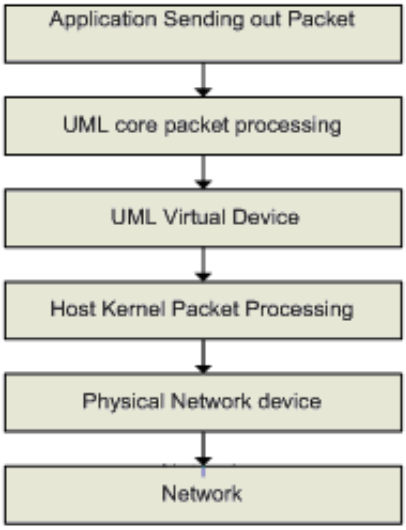

Figure 4.3: Packet Processing in UML

4.3 UML - Advantages and Disadvantages

Advantages:

1. Strong Isolation and Reliability: UML being a user level process presents strong isolation between the guests UML's by the host HOST process encapsulation [4]. When a guest UML is faulted or compromised, neither the host HOST nor other guest UML's are affected thus providing with strong reliability.

2. Easy Installation: VMM approach requires the installation of VMM on the bare hardware. While installing a UML only requires its installation on the host kernel. Thus the hardware installation issues due to the underlying hardware configuration are alleviated.

3. Easy System Diagnoses: To diagnose a UML, normal tools such as gdb can be used. No special VMM support or kernel patches are required [4].

4. Easy Maintenance: A UML can be easily paused, migrated and recovered using the traditional process recovery and maintenance techniques. The UML is easily scalable as running more virtual machines just requires running more user level processes.

5. Easy Configuration: UML parameters can be easily configured as command line options. The amount of memory allotted, the virtual devices and their configurations, all be specified as command line options while starting the UML.

Disadvantages:

1. Based on the UML Networking discussion above, the main sources of UML overheads can be categorized as follows:

   (a) UML provides kernel services by invoking host kernel.

   (b) Extra memory copies have to be executed to move data between the UML and guest kernel.

   (c) Additional software instructions need to be executed due to the UML layer.

   The performance with UML is impacted due to the overheads stated above. For instance, context switches within the virtual UML take up to a factor 100 times longer than on a standard HOST system.

2. The second drawback is that UML only offers a virtual Ethernet interface and not a virtual WLAN wireless interface, such as e.g. a 802.11a/b/g interface. Thus the wireless parameters can only be configured on the host machine. The UML interface acts a wired Ethernet interface that forwards packets on to the host physical wireless card.

In our work, we have considered the suitability of the UML platform to meet our needs based on its advantages of strong isolation and easy configurability as the ORBIT test bed requires the OS to be user configurable. The lack of wireless drivers in the UML though needs to be further investigated and it's a part of future work.

Chapter 5

Experimental Set Up and Results

5.1 Overview of Experimental Set Up

5.1.1 Overview of ORBIT Facility and System Details

The frontline of the test bed are 400 ORBIT nodes designed as custom made personal computer platforms, each equipped with two wireless 802.11a/b/g interfaces. In addition, each node has four Ethernet interfaces and a serial port with the purpose of remote access, supervision, and control. The nodes are placed in the two-dimensional rectangular grid separated by 1-meter distance. Details of the test bed hardware purpose, structure, and functionality are presented in [1].

We run two instances of UML on each ORBIT node. Since each node is equipped with two physical wireless cards, we configure each UML is to have exclusive access to one card. The virtual networking is set up using TUN/TAP devices as explained in Chapter 4. A schematic of a virtualized node is present in Figure 5.2.

5.1.2 Performance Metrics and Experimental Tools

In order to characterize and test the performance of the virtualized wireless network using UML we consider the following four metrics that are most popular and have the most impact on wireless conditions.

1. Throughput: We compare the average throughput for different experiments using UML to those obtained with native linux. Throughput has been chosen as the primary performance metric as it gives the direct indication of the quality of a network link. The experiments have been designed using varying channel

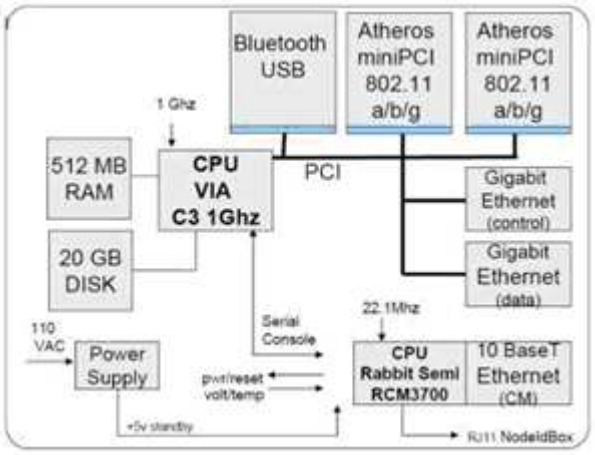

Figure 5.1: Details of ORBIT Node

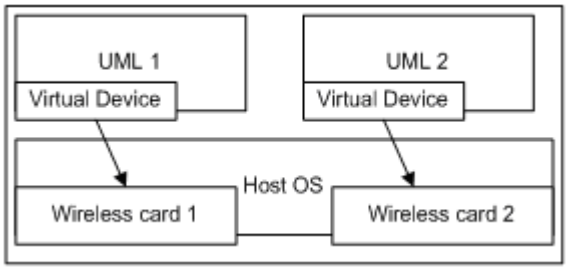

Figure 5.2: A Virtualized ORBIT Node

parameters (channel rate) and input traffic conditions (offered load and packet size).

2. Delay: Limiting the network delay is crucial for many applications. Increase in latencies are expected with the use of UML as it involves overheads of sending the packet to the host/UML kernel through virtual network device and packet processing at both the host and UML core as explained in Chapter 4. This work tries to find the variations in latency in case of virtualization and thus defines the range of latency bounded experiments that can be carried out with virtualized wireless network.

3. Jitter: Jitter is an important factor when measuring the performance of algorithms dealing with real time audio traffic. We expect an increase in jitter with the use of UML as the packets are buffered and thus have to wait for random period of time due to the context switching by the host CPU scheduler

The experiments in this work have been designed using TCP, UDP and RTP transport protocols. IPERF [15] is a widely used bandwidth measurement tool which gives flexibility in terms of sending UDP/TCP traffic with various command line options. We have used IPERF to measure UDP/TCP throughput.

RUDE [16] stands for Real-time UDP Data Emitter and CRUDE for Collector for RUDE. RUDE is a small and flexible program that generates traffic to the network, which can be received and logged on the other side of the network with the CRUDE. RUDE/CRUDE is used for this work for delay and jitter measurements. For the purpose of video experiments, we have used VLC Media Player [17] as the streaming server and client. The video packet data is analyzed using Ethereal.

5.2 Detailed Experimental Set Up and Results

5.2.1 Virtual Network Performane Analysis (One-One) using UDP

In this study we compare the performance of a Virtualized Wireless Link to that of a non virtualized link using UDP as transport protocol. The experiment set up is shown

Parameter	*Value*
Channel Rate	36 Mbps
Frequency	5.18 (Experiment 1) and 5.32 GHz (Experiment 2)
Packet Size	1470 Bytes
Input Traffic Load	50 Mbps
Protocol	UDP

Table 5.1: Experiment Parameters for UDP Point to Point Characterization

in Figure 5.3.

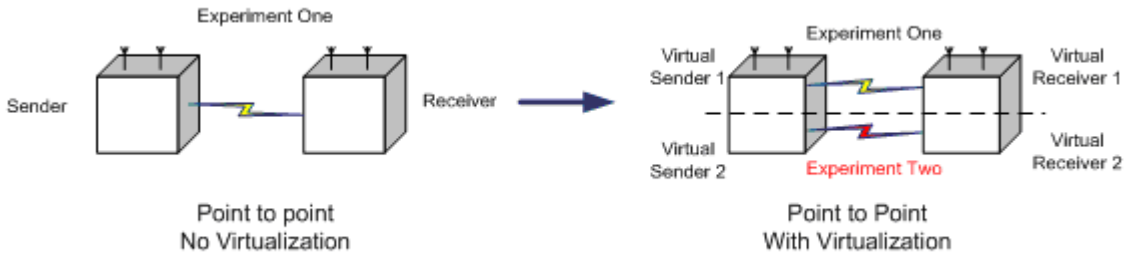

Figure 5.3: Experiment Set up for point to point experiments using UDP

We run an experiment on the native orbit nodes and then repeat the same experiment on a virtualized wireless network i.e. in effect running two experiments at the same time. These experiments have been designed with varying traffic input, channel rate and payload sizes. The objective of the experiments is to analyze the performance deviations introduced by the use of Virtual Machines in case of UDP transport protocol. The general experiment parameters for UDP analysis are shown in Table 5.1. For each experiment, one of the parameter is varied while keeping the others constant.

The results reported for the network with Virtualization are average of the values obtained from the two experiments as shown in Figure 5.3.

Throughput vs. Offered Load

In these experiments, throughput is measured for varying input traffic load for each wireless link. Figures 5.4 and 5.5 plot throughput and its time variance respectively as

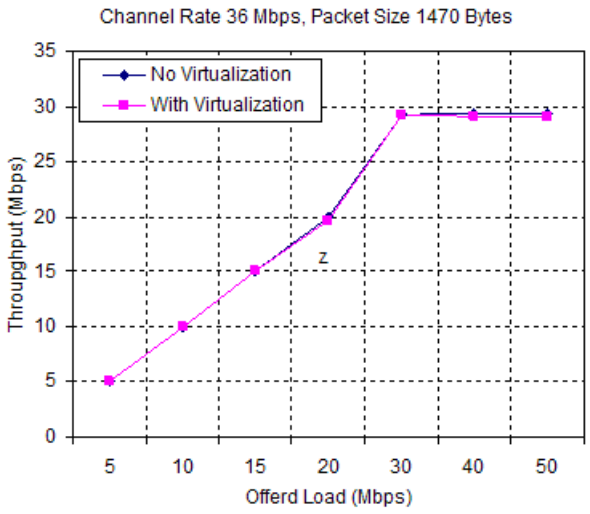

Figure 5.4: Throughput vs. Offered Load (Point to Point UDP)

a function of offered load for virtualized and non virtualized cases.

We observe that for offered loads of up to 50Mbps, the drop in throughput for virtualized network is less than 0.25 Mbps. These results have been obtained by modifying the madwifi driver buffer sizes. Without modifications, throughput difference between the two scenarios at saturation for the same parameters is about 9.2 Mbps. The lost packets are traced to be dropped by the wireless interface.

The interface can handle a burst of packets upto a certain length as specified by the TAIL-DROP-COUNT (default set to 50) variable in the madwifi driver source code. Also the number of transmit buffers that the wireless interface has is defined by the variable TX-BUF (default set to 40). The throughput results in case of virtual network directly correspond to these two fields. If we use the default values of these two fields, the throughput begins to get worse after 10 Mbps offered load and at saturation it drops by 9.2 Mbps.

Since these variables define the number of transmission buffers and their length, they have a strong impact in case of UML. As we run two simultaneous experiments using UML, the two virtual machines buffer the packets and send them in bursts when they get scheduled on the CPU. Thus the respective wireless interfaces also receive the packets in short bursts. In case of absence of buffer or short queue lengths, the driver

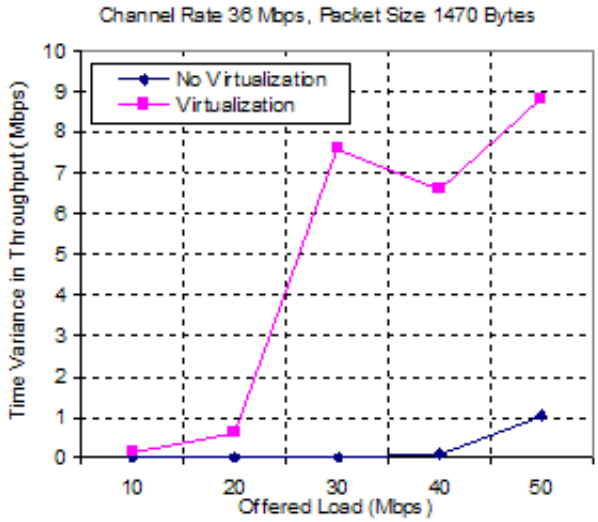

Figure 5.5: Throughput Variance vs. Offered Load (Point to Point UDP)

drops the packets and thus worsening the network performance. These results have been obtained by setting the TAIL-DROP-COUNT and TX-BUF to 1000.

Though the average throughput is not much affected due to the bursts, there is an increase in time variance of throughput as evident from figure. Thus virtual machines can be used to support experiments that require average throughput values and do not care about its time variance.

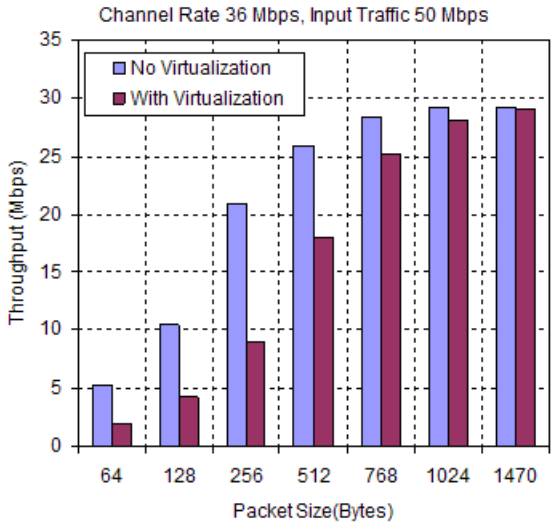

Figure 5.6: Throughput vs. Payload Size (Point to Point UDP)

Throughput vs. Packet Size

Figure 5.6 reports the bar plot for throughput as a function of packet size (64,128, 256, 512, 768, 1024 and 1470 bytes) for virtual and non-virtual wireless network. Packet size has a characteristic effect on the throughput. Shorter packet lengths provide lesser throughput. This is attributed to the fact that as the packet size decreases the lower layer processing overhead increases and the percentage of the IP headers over the payload increases. For packet size 1470 bytes the drop in throughput as reported by UML is 0.25 % which increases to 34 % at packet size of 64 bytes. These values have been obtained by driving the channel to saturation. Thus the values are the maximum throughput that can be obtained at that particular packet size.

The throughput values decrease rapidly with decreasing packet sizes. Figures 5.7 and 5.8 plot the actual number of packets/sec received for the different packet sizes for virtualized and non virtualized network. We observe that the maximum number of packets received per second for network without virtualization is about 80,000 while for virtualized network the number is 32,000 packets/sec. The wireless driver can handle only up to a particular number of packets/sec. The reported number of packets/sec for the virtualized scenario is averaged over experiments one and two. Thus the actual number of packets/sec as processed by the host machine is 65000. Thus irrespective of the packet size, the driver is successfully able to send around 30000-35000 packets/second per UML. Thus we conclude that the applications that require less than 30000 packets/sec traffic would be able to use this virtualization without any deterioration in throughput.

Throughput vs. Channel Rate

Figure 5.9 plots the saturation throughput as a function of various 802.11a channel rates supported by the ORBIT nodes. At channel rates less than or equal to 36 M, the throughput difference between the virtual and non-virtual network experiments is less than 0.5 %. It drops to a maximum of 10.2% at channel rate of 54 M. Thus we conclude that for UDP, network virtualization can support experiments with a throughput drop

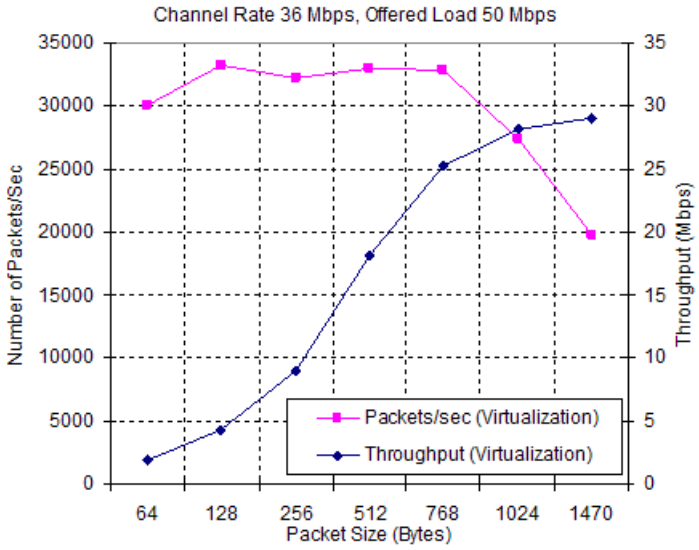

Figure 5.7: Packets/Sec vs. Payload Size (Point to Point UDP)

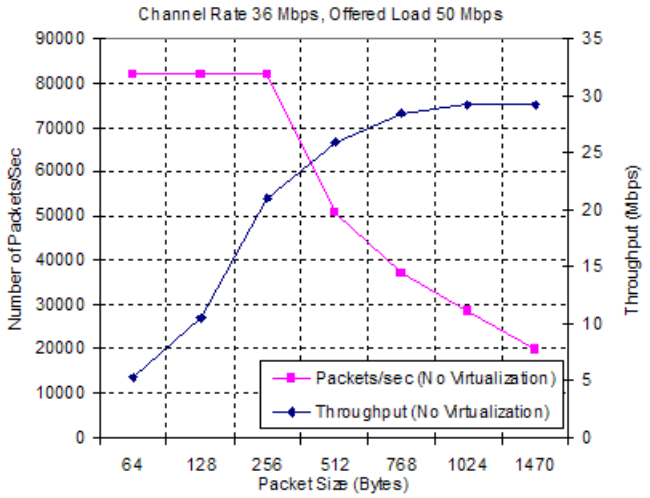

Figure 5.8: Packets/Sec vs. Payload Size (Point to Point UDP)

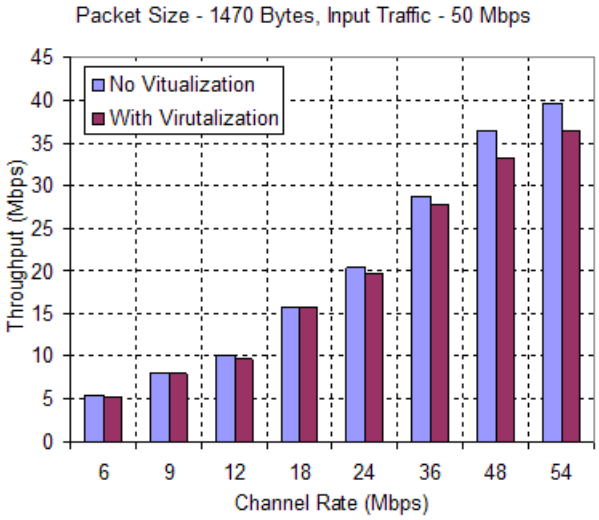

Figure 5.9: Throughput vs. Channel Rate (Point to Point UDP)

of less than 0.5% for channel rates up to 36M.

Delay and Jitter vs. Offered Load

Figure 5.10 compares delay as a function of offered load for UDP experiments with virtualization and no virtualization. As expected, there is an increase in packet transmission delay when UML is involved in the experiments. At 10 Mbps offered load the increase in delay is 4.7 msec which increases to 10.2 msec at 50 Mbps.

The increase is due to processing overheads in the UML core and in the TUN/TAP virtual devices. The UML writes the packet to the Tap character device, which is then received by the host through the Tap virtual interface. The reverse happens when when the UML receives the packet.

An increase in jitter from the native non virtualized experiments is also observed in experiments with UML as evident from figure 5.11. This increase can be attributed to the fact that when two UML are running on the same machine, the packets are buffered for a random amount of time (while the UML is context switched) before sending out.

.

The increase in jitter remains almost the same for all offered loads. An increase of 0.183 milliseconds is observed at 10 Mbps and 0.178 milliseconds at 50 Mbps. From

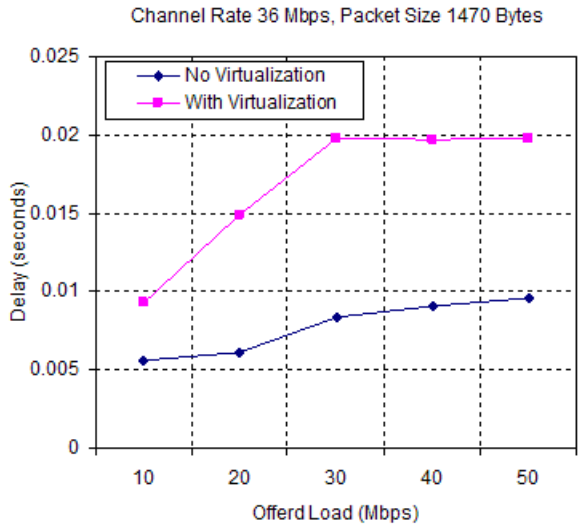

Figure 5.10: Delay vs. Offered Load (Point to Point UDP)

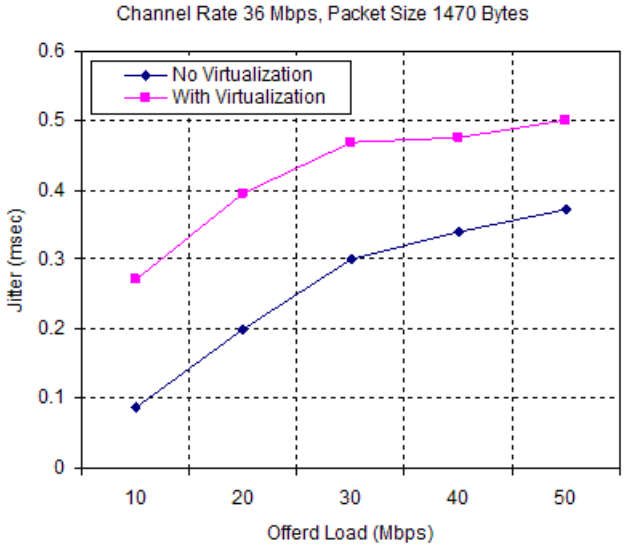

Figure 5.11: Jitter vs. Offered Load (Point to Point UDP)

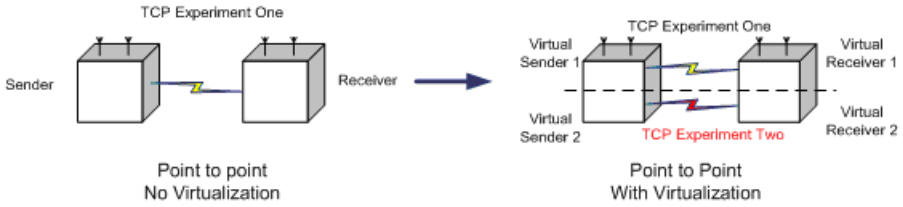

Figure 5.12: Experiment Set up for point to point experiments using TCP

Parameter	*Value*
Channel Rate	Variable (6 - 54 Mbps)
Frequency	5.18 (Experiment 1) and 5.32 GHz (Experiment 2)
Packet Size	1470 Bytes
Protocol	TCP

Table 5.2: Experiment Parameters for TCP Point to Point Characterization

the jitter values obtained we conclude the real time applications based on UDP that can run with an overall maximum jitter of 0.5 milliseconds can be supported by UML. These bounds are for single hop experiments only. Jitter over multiple hops can be aggregated such that it has a more substantial impact on real time applications

5.2.2 Virtual Network Performance Analysis (One-One) using TCP

In this study we compare the performance of a Virtualized Wireless Link to that of a non virtualized link using TCP as transport protocol. The experiment set up is shown in Figure 5.25. The general experiment parameters for TCP analysis are shown in Table 5.2. The results reported for the network with Virtualization are average of the values obtained from the two experiments as shown in Figure 5.25.

TCP bandwidth for wireless network with virtualization is compared to native wireless network for various channel rates in figure 5.13. We observe that the difference in throughput is less than 0.3 % for all the channel rates.

TCP throughput vs time is plotted in figures 5.14 and 5.15 for channel rate 36M and 6M respectively. We observe that there is negligible deviation in time variation

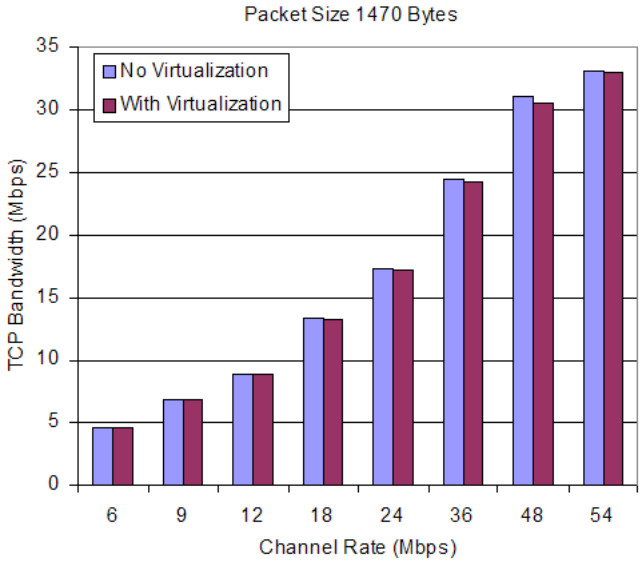

Figure 5.13: TCP Bandwidth Analysis (Varying Channel Rate - Point to Point)

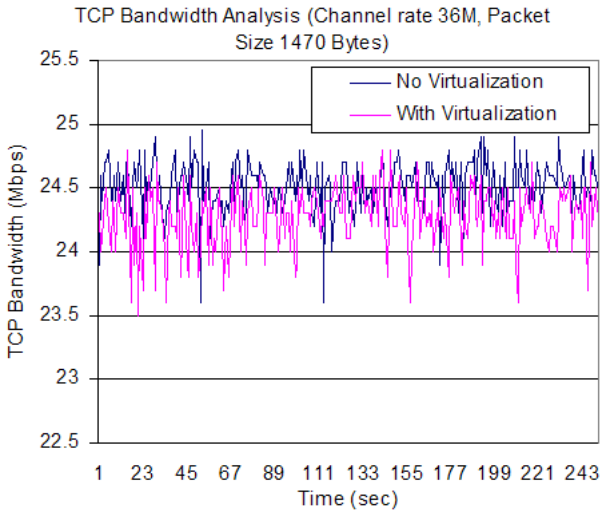

Figure 5.14: TCP Bandwidth vs. Time (Point to Point - Channel Rate 36M)

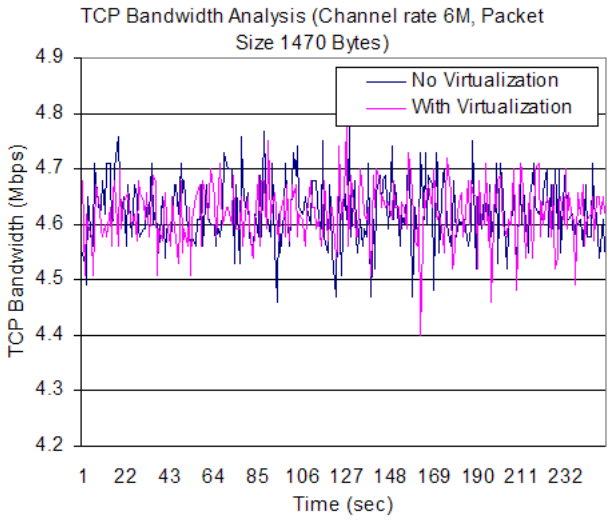

Figure 5.15: TCP Bandwidth vs. Time (Point to Point - Channel Rate 6M)

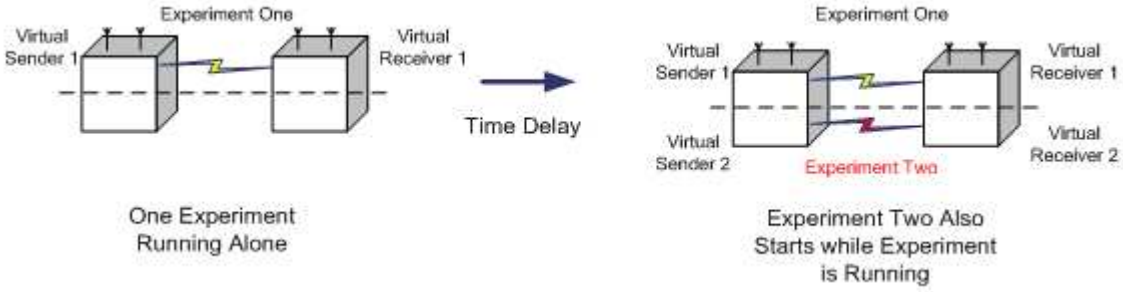

Figure 5.16: Set-Up to Study Transient behavior of Virtual Machines

of throughput in case of virtualization. The throughput lines in case of 6M channel rate run close for the two scenarios while for channel rate 36M, throughput in case of virtualization is slightly lower than the no virtualization case. Thus we observe that virtualization has negligible effect on the TCP performance of the wireless network and it can be safely used to conduct experiments involving TCP as the transport protocol.

5.2.3 Transient Behavior Study

In this section we analyze the transient behavior of the virtual machines in case of UDP as transport protocol. We will look at how the virtual machines interact with each other. The set-up for these experiments is as shown in Figure 5.16.

We start a UDP traffic flow between virtual sender and receiver #1 (Experiment

Parameter	*Value*
Channel Rate	36 Mbps
Frequency	5.18 (Experiment 1) and 5.32 GHz (Experiment 2)
Packet Size	1470 Bytes
Input Traffic Load on Experiment 1	10 and 25 Mbps
Input Traffic Load on Experiment 2	5-40 Mbps
Protocol	UDP

Table 5.3: Experiment Parameters for Transient Analysis

One) shown in figure and after some time delay start another UDP traffic flow between virtual sender and receiver #2 (Experiment Two). We analyze the effect of experiment two on the throughput of already running experiment one for various channel and experimental parameters. We also define a constant called coupling coefficient that is given as

$$C = \frac{(R - R')}{R} \tag{5.1}$$

Where C = coupling coefficient

R = Average throughput obtained at Virtual Receiver 1 when only experiment is running

R' = Average throughput obtained at Virtual Receiver 1 after experiment two has also started running.

Experiment parameters to calculate the coupling coefficient are as shown in Table 5.2.3.

Figure 5.17 shows the coupling effect between the two virtual machines with time. We run experiment one at an offered load of 20 Mbps. The experiment two however runs for short intervals for increasing offered loads of 5, 10, 20 and 30 Mbps. We observe that as soon as the second experiment starts, the throughput on experiment one drops to zero for an instant, but it again settles down around its original value. Also an in increase in variance in throughput is observed when both the experiments are running simultaneously.

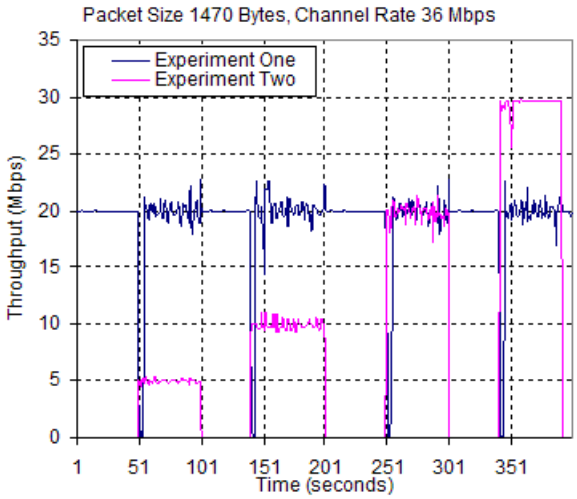

Figure 5.17: Transient Behavior with Time

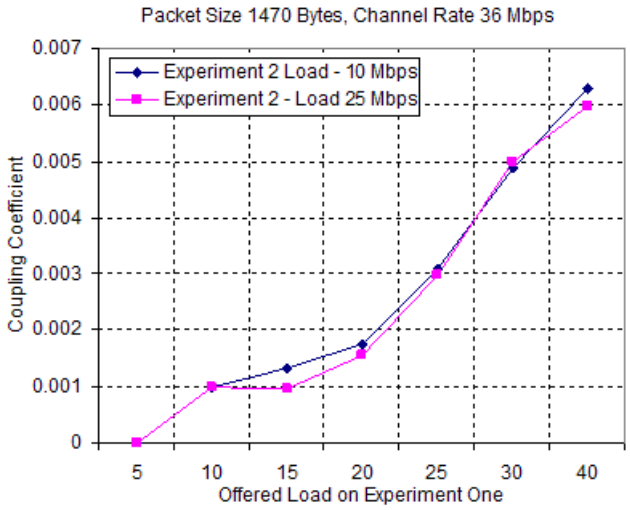

Figure 5.18: Coupling Coefficient vs. Offered Load on Experiment One

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
1764	root	25	0	67664	38m	38m	R	88.9	8.2	1:35.61	vmlinux
1769	root	15	0	1032	504	504	T	11.0	0.1	0:10.87	vmlinux
1804	root	16	0	2200	1096	864	R	0.3	0.2	0:00.88	top
1	root	16	0	1968	656	556	S	0.0	0.1	0:01.02	init
2	root	34	19	0	0	0	S	0.0	0.0	0:00.00	ksoftirqd/0
3	root	10	-5	0	0	0	S	0.0	0.0	0:00.00	events/0
4	root	10	-5	0	0	0	S	0.0	0.0	0:00.01	khelper
5	root	11	-5	0	0	0	S	0.0	0.0	0:00.00	kthread
7	root	10	-5	0	0	0	S	0.0	0.0	0:00.05	kblockd/0
8	root	14	-5	0	0	0	S	0.0	0.0	0:00.00	kacpid
117	root	15	0	0	0	0	S	0.0	0.0	0:04.06	pdflush
119	root	12	-5	0	0	0	S	0.0	0.0	0:00.00	aio/0
118	root	15	0	0	0	0	S	0.0	0.0	0:17.94	kswapd0
704	root	10	-5	0	0	0	S	0.0	0.0	0:00.00	kseriod
743	root	15	0	0	0	0	S	0.0	0.0	0:00.12	kjournald
965	root	15	0	0	0	0	S	0.0	0.0	0:24.83	kjournald

CPU Usage with one UML

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
1764	root	25	0	67664	38m	38m	R	46.5	8.2	0:42.43	vmlinux
1788	root	25	0	67660	34m	34m	R	43.6	7.3	0:22.21	vmlinux
1769	root	15	0	1032	504	504	T	5.3	0.1	0:05.09	vmlinux
1790	root	15	0	1032	504	504	T	4.7	0.1	0:02.58	vmlinux
1804	root	16	0	2200	1096	864	R	0.3	0.2	0:00.73	top
1	root	16	0	1968	656	556	S	0.0	0.1	0:01.02	init
2	root	34	19	0	0	0	S	0.0	0.0	0:00.00	ksoftirqd/0
3	root	10	-5	0	0	0	S	0.0	0.0	0:00.00	events/0
4	root	10	-5	0	0	0	S	0.0	0.0	0:00.01	khelper
5	root	11	-5	0	0	0	S	0.0	0.0	0:00.00	kthread
7	root	10	-5	0	0	0	S	0.0	0.0	0:00.05	kblockd/0
8	root	14	-5	0	0	0	S	0.0	0.0	0:00.00	kacpid
117	root	15	0	0	0	0	S	0.0	0.0	0:04.06	pdflush
119	root	12	-5	0	0	0	S	0.0	0.0	0:00.00	aio/0
118	root	15	0	0	0	0	S	0.0	0.0	0:17.94	kswapd0
704	root	10	-5	0	0	0	S	0.0	0.0	0:00.00	kseriod
743	root	15	0	0	0	0	S	0.0	0.0	0:00.11	kjournald

CPU usage with two UML

Figure 5.19: CPU Usage Snapshot with one and two instances of UML

UML is a heavy weight process. Each instance of UML creates its own process to perform work in parallel. Figure 5.19 compares the CPU usage with one and two active UML's. When only one UML is running, it takes upto 100% of CPU. With two active instances of UML, they divide the CPU share equally. The sudden drop in throughput when a second traffic flow is started, is due to virtual sender and receiver # 2 taking over their respective CPU immediately as they become active. After an instantaneous irregularity, both the virtual machines then share the CPU equally and thus the throughput settles down again around the original value. The increase in variance is again due to the buffering of packets by a virtual machine while another virtual machine is running.

Figure 5.18 plots the coupling coefficient 'C' as a function of offered loads on experiment one. The experiment one runs on an offered load from 5 to 40 Mbps, while the experiment two starts with the offered load of 10 and 25 Mbps. The coupling coefficient when experiment one is running 5 Mbps is seen to be 0 while the maximum of .0065 is observed when it's running 40 Mbps. This coupling coefficient values are almost same regardless of the load on experiment two. Thus we conclude that the transient behavior is more CPU driven that the load on experiment two.

We have compared the performance of virtualized network with the non virtualized one for basic point to point experiments using TCP and UDP and have defined some upper bounds on performance metrics of throughput, delay and jitter. We have also looked at the transient behavior in case of Virtualization. Now we will compare a few common experiments that are performed on ORBIT facility on the virtualized network and analyze the degradation/deviations in their performance.

5.2.4 Experimental Comparison of Virtual and Non-Virtualized Networks

Multiple Clients transmitting UDP traffic to a Sever

In this experiment we configure four UDP clients sending traffic to a single server on the same physical channel. The experiment set up for a virtualized network and non

Parameter	*Value*
Channel Rate	Variable (36 Mbps)
Frequency	5.18 (Experiment 1) and 5.32 GHz (Experiment 2)
Packet Size	1470 Bytes
Protocol	UDP

Table 5.4: Experiment Parameters for Client/Server Experiment

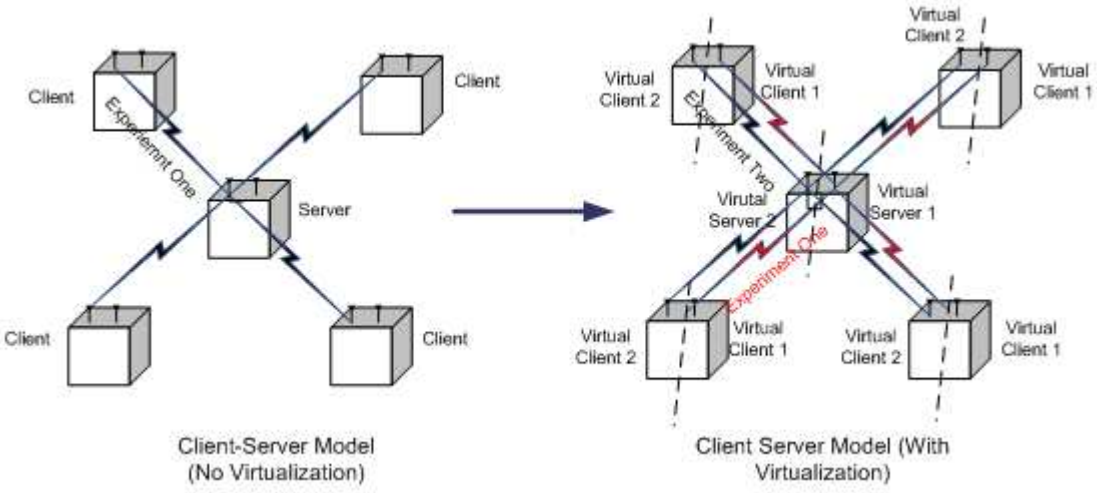

Figure 5.20: Experiment Set Up for Client Server Model using UDP Traffic

virtualized network is shown in figure 5.20. The objective of the experiment is to compare the network performance for both the cases by looking at throughput, delay and jitter.

Figure 5.21 plots the combined throughput obtained at the server for both the cases for various combined offered loads. We observe that for loads up to 30 Mbps the drop in throughput is less than 5% while the maximum drop of 10% is obtained at saturation. Thus the throughput difference between the two case remains in the bound of maximum 10%. The drop in throughput is combined over the 4 virtual machines, thus its greater than that observed for point to point experiments.

Delay and jitter for the two cases are compared in figure 5.22. The line graph depicts jitter while the bar plot depicts the delay. Significant increases in delay and jitter have been observed for the experiment performed on virtualized network. The

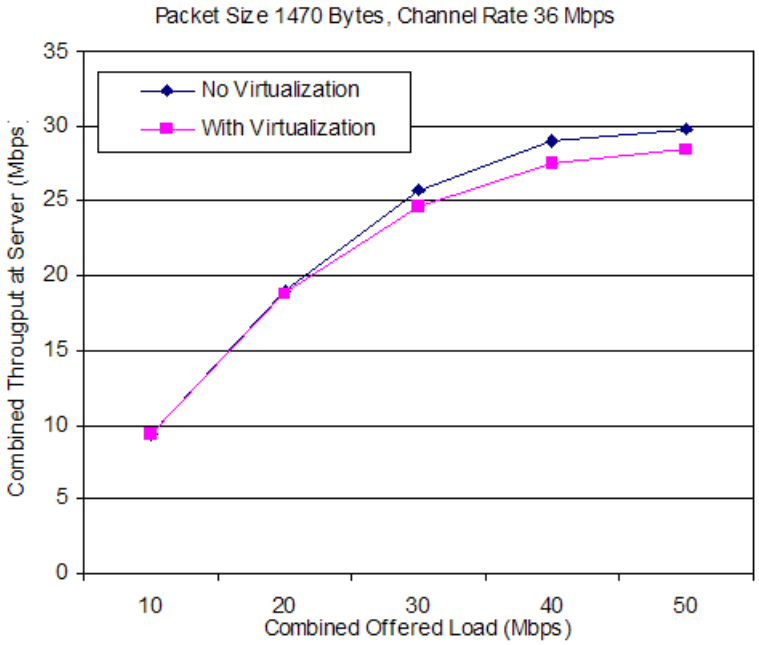

Figure 5.21: Combined Throughput vs. Offered Load (Client-Server Experiment)

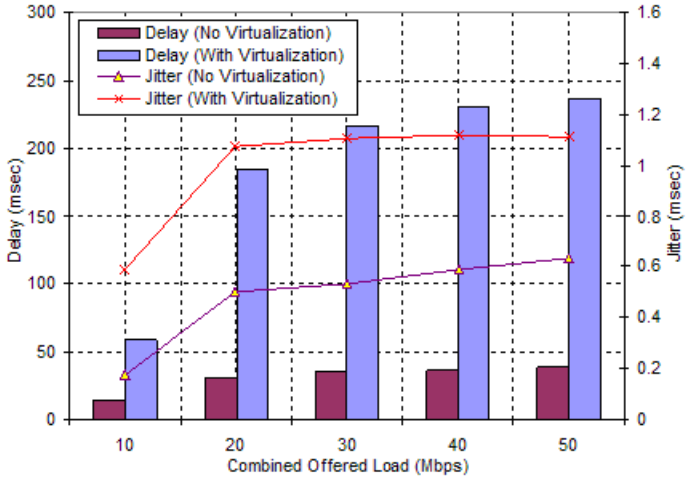

Figure 5.22: Delay and Jitter vs. Offered Load (Client-Server Experiment)

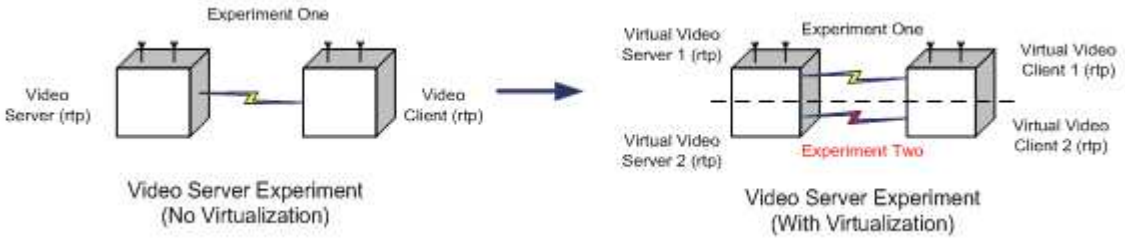

Figure 5.23: Experiment set up - Video Client Server Experiments

Parameter	*Value*
Channel Rate	6M
Frequency	5.18 (Experiment 1) and 5.32 GHz (Experiment 2)
Video Bit Rate	Variable
Protocol	RTP

Table 5.5: Experiment Parameters Video Experiment

increase in delay though increases with the offered load. It increases from a difference of 29 milliseconds at 10 Mbps offered load to a difference of 200 milliseconds at saturation. We observe an almost constant increase in jitter in the range of 0.5 - 0.7 milliseconds for all the offered loads.

Performance of the virtualized network corresponds with the earlier point to point UDP characterization. The loss in throughput at saturation for channel rate 36M was observed as 3.4%. In this experiment (that also uses channel rate 36M) the throughput for network with virtualization drops by 3.9% from the non virtualized case.

Video Server streaming Video Traffic to a Client using RTP

The experiment set up is shown in figure 5.23 and parameters in Table 5.5. We set up video server and client on two nodes using an open source media player VLC. We were not able to actually see the video quality as orbit nodes do not have a graphical console. We use RTP protocol to send the video traffic. We sniff the packets on the client using 'tcpdump' utility and analyze the 'tcpdump' log files using 'Ethereal' network analyzer.

The objective of this experiment is to analyze the performance of virtualized network

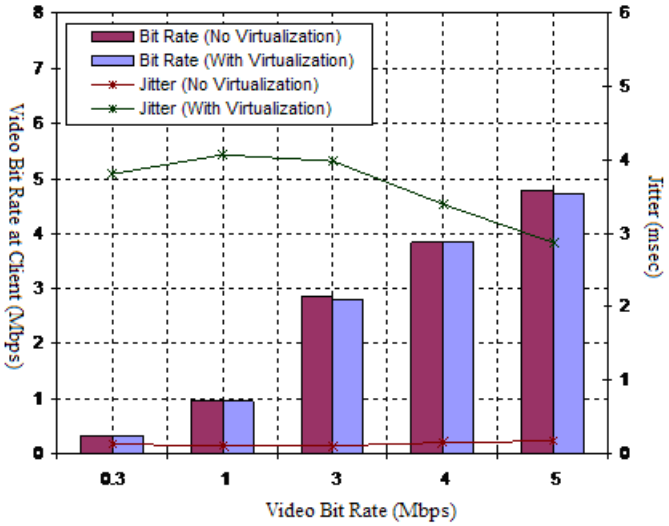

Figure 5.24: Jitter and Bit rate vs. Video bit rates

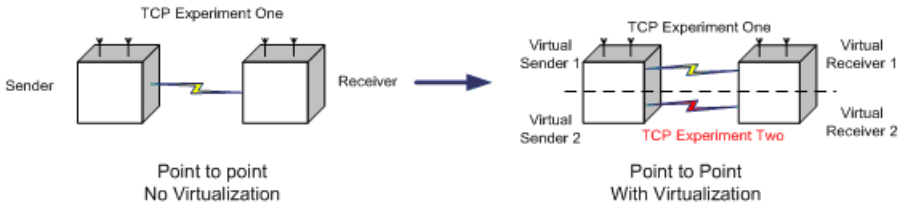

Figure 5.25: Experiment Set up File Transfer experiment using TCP

in case of a real time application (using RTP in this case). We stream videos coded with different bit rates and analyze and compare the video bit rates and jitter as observed by the client. In Figure 5.24 the bar plot compares the bit rates while line graph plots the jitter.

The video bit rates obtained though are similar for both the cases while there is a significant increase in jitter in case of network virtualization. With virtualization the jitter varies between 2.8 and 4 milliseconds while for non virtualized case it remains below 0.3 milliseconds. A real time audio/video application typically requires jitter less than around 50 milliseconds. Thus even though there is an increase in jitter with virtualization, it still remains well below the required limits and hence such experiments can be carried out on the virtual platform. But these results are true only for single hop experiments. With multihop real time experiments, the jitter might increase out of limits with the increase in number of hops.

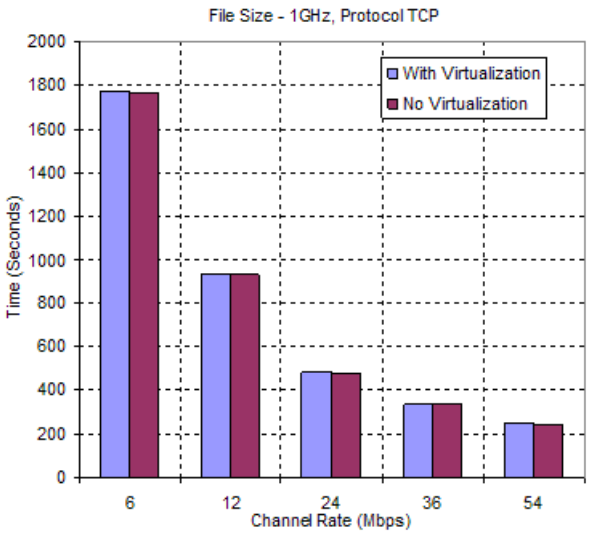

Figure 5.26: File Transfer Time using TCP vs Channel Rate

File Transfer (1 GHz) using TCP

In this experiment we calculate the total time taken for point to point transfer of a large file (size 1 GHz) using TCP protocol. Figure 5.26 plots the total time taken for the file transfer for both virtual and non virtual networks. We observe that there is an increase of about 5 seconds in the time taken for file transfer in case of virtualization for all the channel rates.

The increase in the total file transfer time may be attributed to the increase in TCP packet processing times due to the UML layer overhead. The delay in processing results in a smaller TCP bandwidth which becomes more obvious with large experiment durations. Short time point to point TCP experiments are not much affected as the actual deviation in bandwidth is very less.

Chapter 6

Conclusion and Future Work

6.1 Conclusion

In this work we have tried to characterize the behavior of Wireless virtualization and conclude whether it's a feasible option. We study and analyze the performance of UML under varying traffic parameters and channel conditions. From the study we conclude:

- For UDP traffic, it is safe to use UML with experiments investigating long term average throughputs. Short term throughput experiments might give erroneous results in case there is experiment going on the second virtual machine.
- For UDP traffic, the throughput losses depend upon the number of packets/sec that is being sent out on the channel as shown in figure 5.6. Thus we conclude that irrespective of the packet sizes, the experiments that require less than 30,000 packets per second can successfully be conducted on the virtualized platform.
- A wireless network virtualized using UML can be used for any kind of TCP experiments with all the channel rates as we observer negligible difference in throughputs and its time variation.
- From the real time experimental study we conclude that there is a considerable increase of jitter with the use of UML for time bounded protocols such as RTP. But the jitter still remains of the order of 4-6 milliseconds that is well below the maximum jitter levels (about 50 milliseconds) that the real time audio and video applications can withstand. Thus we conclude that our virtualized platform can support the standard real time applications.

While one is performing experiments on a virtualized network, it is important that a proper trace of packets is maintained. If there are any losses in traffic, the experimenter should make sure the cause of the discrepancy. The packet losses may be arising due to network virtualization and not entirely may be dependent on the wireless network.

6.2 Future Work

There is immense possibility of further research and implementation in the current work. They are stated as follows:

1. This work only investigates UML as an option for wireless network virtualization. One can investigate other OS virtualization options such as Xen and VMWare as mentioned in Chapter 3 and then conclude as to which option would be best suited for ORBIT virtualization.

2. The existing ORBIT software infrastructure consisting of the Node Handler, Node Agent and applications such as OTG/OTR and OML needs to be integrated with the UML running over the ORBIT nodes.

3. A virtualized wireless driver for UML - As mentioned in Chapter 5, UML does not have direct access to the wireless card. It instead has a virtual Ethernet connection with the host kernel. A virtualized wireless driver can provide the UML kernel with wireless configuration capabilities. An' iwconfig' command given to the UML virtualized driver can be passed on to the host kernel which in turn would implement the commands.

4. Designing a Virtualization infrastructure consisting of a facility like a grid manager that would oversee the scheduling of the virtualized network and would limit the privileges of an experimenter to the resources he has been allocated for a given time frame.

References

[1] D. Raychaudhuri, I. Seskar, M. Ott, S. Ganu, K. Ramachandran, H. Kremo, R. Siracusa, H. Liu, and M. Singh, "Overview of the ORBIT radio grid testbed for evaluation of next-generation wireless network protocols," IEEE Wireless Communications and Networking Conference, March 2005. [Online]. Available: http://www.orbit-lab.org/download/publications/Orbit_Overview.pdf

[2] "Us nsf - national science foundation." [Online]. Available: http://www.nsf.gov/

[3] W. W. Group-GENI, "Technical document on wireless virtualization," September 2006. [Online]. Available: www.geni.net/GDD/GDD-06-17.pdf

[4] J. Dike, "A user-mode port of the linux kernel," 5th Annual Linux Showcase and Conference, Oakland, California, 2001.

[5] "User mode linux kernel." [Online]. Available: http://user-mode-linux.sourceforge.net/

[6] A. Bevier, M. Bowman, B. Chun, D. Culler, S. Karlin, L. Peterson, T. Roscoe, T. Spalink, and M. Wawroniak, "Operating system support for planetary-scale network services," In proceedings of the NSDI., San Francisco, California, 2004. [Online]. Available: http://www.cs.princeton.edu/ acb/nsdi04/paper.pdf

[7] "Linux vserver project." [Online]. Available: http://linux-vserver.org/

[8] A. Bavier, T. Voigt, M. Wawrzoniak, and L. Peterson, "Silk: Scout paths in the linux kernel," *Technical Report- Uppasala University Sweden*, February 2002.

[9] A. Bavier, N. Feamster, M. Huang, L. Peterson, and J. Rexford, "In vini veritas: Realistic and controlled network experimentation," In ACM SIGCOMM, vol. 36, no. 4, San Francisco, California, October 2006, pp. 3–14. [Online]. Available: http://www.vini-veritas.net/papers

[10] Y. Koh, C. Pu, S. Bhatia, and C. Consel, "Efficient packet processing in user-level operating systems: A study of uml," *Proceedings of the 31st IEEE Conference on Local Computer Networks LCN* , 2006.

[11] *Supplement to IEEE Standard for Information technology Telecommunications and information exchange between systems Local and metropolitan area networks Specific requirements Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications: High-speed Physical Layer in the 5 GHZ Band*, IEEE-SA Standards Board IEEE Std 802.11a-1999 (Supplement to IEEE Std 802.11-1999), September 1999.

[12] "Vmware: Virtual infrastructure software." [Online]. Available: www.vmware.com

[13] A. Whitaker, M. Shaw, , and S. Gribble, "Scale and performance in the denali isolation kernel," *In Proc. of the OSDI.*, Boston, MA, December 2002 2003.

[14] S. Banerji, "Time based virtualization of an 802.11-based wireless facility." [Online]. Available: www.geni.net/docs/banerji.pdf

[15] "Iperf 1.7.0: The tcp/udp bandwidth measurement tool." [Online]. Available: http://dast.nlanr.net/Projects/Iperf/

[16] "Rude/crude - real-time udp data emitter/collector for rude." [Online]. Available: http://rude.sourceforge.net/

[17] "Videolan vlc media player." [Online]. Available: http://www.videolan.org/

[18] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, I. Pratt, A. Warfield, and R. Neugebauer, "Xen and the art of virtualization," *In Proc. of the ACM SOSP.*, October 2003.