POLITECNICO DI TORINO

III Facoltà di INGEGNERIA
Master of Science in Computer and Communication Networks
Engineering

Master Thesis

Multi-Stage Software Routers Implementation in Virtual Environment

Relatore:
Prof. Andrea Bianco

Candidato:
Nanfang Li

November 2009

Summary

Routers are the key components of modern packet networks and of the Internet in particular. The request for high-performance switching and transmission equipment keeps growing, due to the continuous increase in the diffusion of Information and Communications Technologies (ICT) and new bandwidth-hungry applications and services based on video and imaging. Routers have been able to support the performance growth by offering an ever increasing transmission and switching speed, mostly thanks to the technological advances of microelectronics. But from another point of view, nearly all of these high-end enterprise routers and core routers are based on proprietary architectures which may influence the compatibility in the total Internet frame work. Also the configuration of these routers and training cost to manage multi-vender routers with different architecture is very high. These shortcomings of encapsulation routers depress their growth but give researchers and networking developers the possibility to find new ways to substitute or at least to cooperate with proprietary routers. One possible solution is based on software routers exploiting personal PC architectures because of their low cost, open architecture and continuous evolution due to Moore's Law.

However, software routers suffer from intrinsic limitations of PC architectures: CPU speed, bus and memory access speed may easily become a bottleneck. A possible solution is to devise multistage software routers, i.e. architectures based on PCs interconnection. This solution aims at building up a large scale size router, i.e. a router with a large number of interfaces and huge routing capabilities. The thesis focus on a previously proposed multistage architecture, based on the interconnection of off-the-shelf PCs running open software such as Linux, Xorp, Click, Quagga. Multiple PCs in the front-end stage act as interfaces and load balancers, whereas multiple PCs in the back-end stage manage the routing functions. This multistage architecture overcomes several disadvantages that single PC suffers to when acting as routers, such as limited bus speed, Central Processing Unit (CPU) bandwidth, limited scalability in terms of number of network interface and lack of resilience to failures.

In recent years, virtualization techniques become reality, thanks to the quick

development of the PC architectures, which are now able to easily support several logical PCs running in parallel on the same hardware. Virtualization provides some advantages, among others hardware and software decoupling, encapsulation of virtual machine state, failure recovery and security. In the focus of the thesis, virtualization permits to build a multistage architecture exploiting logical and not physical devices, As a consequence, physical resources can be exploited in a more efficient way, enabling cost reduction (when planning the interconnection network) and energy saving features (switching on and off physical device when needed). Furthermore, logical resources could be rented on-demand instead of being directly owned, reducing Capex and Opex costs.

However, virtualization techniques are still difficult to deploy, and several challenges need to be faced when trying to integrate them into multistage router architectures. The main thesis goals are: to find out the feasibility of the virtualization approach, to implement the multistage software routers exploiting logical, i.e. virtualized, resources, and to analyze the performance of software routers in the virtualization domain.

Among the four types of virtualization techniques known as hardware emulation, full virtualization, para virtualization and operating system level virtualization, the full virtualization was considered, due to the minimal modifications required to the operating system and to its maximum flexibility. VMware is the Full virtualization software used, because it is a relatively new application with many improved functionalities, such as new virtual network interface drivers VMXNET and convenient user interfaces.

Two versions of VMware, namely ESXi, 3.5 in March 2009 and 4.0 in July 2009 were deployed, running as many virtual machines as possible. Each virtual machine was able to operate as a fully fledged router. The first test encompassed only one virtual machine acting as a software router, whereas the other virtual machines were active but left unused, to consume the hardware resources. The results show that the energy saving functions can be introduced into the virtual domain, since other VMs do not affect the running routers so much. Then multiple virtual machines acting as software routers were run in parallel. The aggregate throughput and the fairness tests show that the routing performance decrease as the number of virtual routers increase, due to the interrupts and context switches among the routers. If keeping a reasonable number of running VM routers inside each server, performance can be considered as satisfactory. If considering VMware 4.0, also the fairness test among flows was satisfactory.

To make a comparison between different virtualization technologies, the para virtualization software XEN was also tested. XEN required some additional effort

to install and some modifications on running virtual operating system. Furthermore, XEN showed much worse performance in terms of forwarding throughput and fairness as compared to VMware.

Since VMware 4.0 shows highest performance in routing throughput and fairness tests, close to those of a physical Linux machine, we choose VMware to build the multistage software router architecture exploiting VMs. XORP was run in back-end routers, to allow them to operate as routing machines and to execute the DIST protocol, needed to control the multistage architecture. In the front-end PCs, CLICK was exploited to provide the load balancing function and the MAC address translation. Unfortunately, the Click forwarding performance is much worse than those if Linux default forwarding path, especially for small packet size flow. This is due to some impairment from the VMware network stacks and PCI (PCIx) bus inability to some specific burst size traffic like in VMware ESXi 3.5. Overall, the multistage architecture run correctly when built exploiting VMs, but some performance reduction has to be paid with respect to the case when physical machines are used.

In summary, the master thesis work indicates that the proposed multistage architecture built in a virtual environment is a feasible solution for the near future in searching for scalable software routers, providing energy savings and resilience features, but with some degree of performance reduction compared to the case of multistage routers built exploiting physical devices. Thus, searching the trade-off between routing performance and improved flexibility is an interesting issue for future research.

Acknowledgements

This thesis has been done in Telecommunication Networks Group, a Dipartimento di Elettronica of Politecnico di Torino, Italy.

Firstly, I would like to express my sincere gratitude to my advisor, Associate Professor Andrea Bianco, for the profound advices and useful weekly meeting. The discussion with him shows me the way to proceed in my scientific research and teaches me the methods to overcome difficulties. I have benefited enormously from working with him over the past half year.

Secondly, I would like to thank Robert Brike and Luca Giraudo, two PHD students in this network group and offer me much help in detailed work. It saved me a lot of efforts.

Thirdly, I am also grateful to all my colleagues I have worked together in the networking group. The experience with them is enjoyable and valuable

Last but not least I want to thank my parents and friends for their long-term support during my study.

Contents

List of Figures

Chapter 1

Introduction

Routers are the key components of modern packet networks, and of the Internet in particular. The request for high-performance switching and routing transmission equipment keeps growing, due to the continuous increase in the diffusion of Information and Communications Technologies (ICT) and new bandwidth-hungry applications and services based on video and imaging streaming. So the demand for faster routing path calculation and higher aggregation interface transmission ability routers became imminence than ever before. Coming to the technology reality, the manufacture of new type of routers could supply nowadays requirement, but mostly under the proprietary architecture routers from several big company such as Cisco, 3com, HuaWei etc. It prevents the rapid developing of packet routers somehow, yet stimulates the emergence of open routers from another point of view.

1.1 Routers Challenging

Contrary to what happened for PCs, where standards were defined, allowing the development of an open, multi-vender market, the market of networking equipment in general, and of routers in particular, has always been characterized by the development of proprietary architecture. In this circumstance, although the routers have been able to support the performance demand by offering an ever increasing transmission and switching speed thanks to the technological advances of microelectronics, this sealed development concept results in incompatible equipments and architectures, especially in terms of configuration and management procedures, as well as the requirement to train network administrators to handle several proprietary architectures or to be limited to a single vendor. Thus, the final cost of equipment is high with respect to performance and equipment complexity.

As time pass on, new applications' popularization such as VoIP, IPTV and file sharing based on P2P, web server storage/downloading scheme, need more and more

bandwidth. Along with this, it is hard to implement new type of routers supporting huge amount of throughput and high forwarding speed in one single "box". Even it can be realized, the cost for this new generation routers will prevent its further evolution for some degree. Another point is this commercial dedicated equipment limit the level of flexibility. It is very difficult both to have access to internal details and to perform any knid of interventions that would require more complex operations than those involved by a configuration of parameters. In this case, the "closure" to external modifications is a clear attempt to protect the industrial investment.

But this challenging arouse network architecture designer, research group and college institute to find other practical solutions to substitute or at least to cooperate with proprietary routers. The "experimental" nature of Internet and its diffusion suggest a different approach. This type of need is more evident within the scientific community, which often finds many difficulties in realizing experiments, test-beds and trials for the evaluation of new functionalities, protocols and control mechanisms. But also the market frequently asks for a more open and flexible approach, like that suggested by the Open Source philosophy for software. This is especially true in those situations where the network functions must be inserted in products, whose main aim is not limited to realizing basic network operations.

1.2 Open Software Routers Novelty

Among all the attempts, some of them are outlined in [1] and [2], indicate that the recent technology advances give a good chance to do something really effective in the field of open Internet devices, sometimes called Open Routers (ORs) or Software Routers. This possibility comes, for what concerns the software, from the Open Source Operating Systems (OSs), like Linux and FreeBSD (which have sophisticated and complete networking capabilities), and for what concerns the hardware from the COTS/PC components (whose performance is always increasing, while their costs are decreasing). The attractiveness of the OR solution can be summarized in multi-vendor availability, low-cost and continuous update/evolution of the basic parts, as assumed by Moore's law.

Indeed, the PC world benefits from the de-facto standards defined for hardware components, which enabled the development of an open market with a wide availability of multi-vendor hardware, low costs given by the large PC market, wide information available on their architecture and the large availability of open-source software for networking applications, such as Click [3], Xorp [4] and Zebra/Quagga [5]. All these give us sufficient motivation and enough attraction to search a feasible solution on Open Routers. For what concerns the external measurements, [6] shows that a Linux PC configured as a router can obtain an interesting forwarding

throughput with relative low latencies by doing the measurement tests in RFC2544 compliant way. [7] gives us some hints about the internal measurements and a couple of optimal ways through tuning the network stack parameters in Linux kernel or using special patches, such as LC-tire routing table loop-up mechanism and novel routing table cache placement based on flows, to show that there are still lots of work can be done to improve the routing performance in Linux PC.

1.3 Multistage Switching

Although we can get interesting performance when configuring single PC as one Software Router as sketched in [6] and [7], only one off-the-shell PC still can not compare with commercial equipment when routing packets as everybody could imagine. Indeed, despite the limitations of bus bandwidth and central processing unit (CPU) and memory-access speed, current PC based routers have a traffic-switching capability in the range of a few gigabits per second, which is more than enough for a large number of applications. Moreover, performance limitations may be compensated by the natural PC architecture evolution, driven by Moore's law. However, high-end performance and large size devices cannot be obtained easily today with routers based on a single PC. In addition to performance limitations, several other objections can be raised to PC-based routers such as: software limitations, scalability problems, lack of advanced functionalities, inability to support a large number of network interfaces, as well as the inability to deal with resilience issues to match the performance of carrier-grade devices.

To overcome some of the limitations of software routers based on a single PC, using multiple PCs to route packets become an easy thinking but hard implementing solution. Actually, these multistage switching architectures were previously proposed in different research areas to overcome single-machine limitations [8]. Initially studied in the context of circuit oriented networks to build large-size telephone switches through simple elementary switching devices, multi-stage architectures were traditionally used in the design of parallel computer systems and, more recently, considered as a viable mean to build large packet-switching architectures. Indeed, the major router producers have proposed proprietary multi-stage architectures for their larger routers as reported in [9] and [10], and follow traditional multi-stage, telephony-derived switching architectures. In most cases, the routing functionality is distributed among cards installed in different interconnected racks. Such systems target high-end routers, with performance and costs that are not comparable with those of PC-based router architectures.

Since Open Software Routers became viable, researchers start to look for new ways to combine this multistage idea into Open Routers. Panama [11], a scalable

and extensible router architecture using general-purpose PCs with programmable network interfaces as router nodes and a Gigabit-Ethernet switch as the router backplane, is a project aimed to build large scale software routers with traditional PCs individually acting as independent standard routers. The IETF is also interested in distributed router architecture. The Forwarding and Control Element Separation (ForCES) Working Group [12] aims at proposing standards for the exchange of information between the control plane and the forwarding plane, when the control and forwarding elements are either in the range of a small number of hops or even in the same box. To the best of our knowledge, no proposals are available on the use of interconnected PC architectures that explicitly exploit traffic load-balancing to improve performance, scalability and robustness, as successfully done in web-server farms. Note that load-balancing at input stages has been shown to be beneficial for scheduling in high-end input-queued switches [13], thanks to the ability of transforming a non-uniform traffic pattern at input NICs into a uniform traffic pattern at the switching fabric ingress, a feature that we can exploit to overcome single-PC limitations by distributing the computational load among several back-end PCs. We will show this architecture in the next chapter thoroughly.

1.4 Virtualization Technology Revival

Since our multistage software routers can work properly in physical PCs now, we are seeking for new things to extend our Software Routers. One way is to implement complex controlling algorithms as those in commercial routers such as multicast capability, user restriction functions and security algorithms, the other possibility is to take the advantage of Open Routers flexibility in resource allocation to multiple users and energy saving. My colleagues are still working in physical PC domain, try to implement complex algorithms in controlling plan and enhance the switching capability in data plan, while I turn to a new aspect in virtual domain, which we think has the potential to add flexibility functionalities.

1.4.1 Virtualization Development Routine

Developed for nearly 40 years since the late 1960s, virtual machine monitor (VMM), the key component which stands for a software abstraction layer that partitions a hardware platform into one or more virtual machines, has gone through its favorable and fading period. As everybody knows, at the beginning of computer development, general-purpose computing was the domain of large, expensive mainframe hardware. It is not likely every single person can own one private computing machine, so users found that VMMs provided a compelling way to multiplex such a scarce resource among multiple applications. Thus, for a brief period, this technology flourished

both in industry and in academic research.

But the 1980s and 1990s, however, brought modern multitasking operating systems and a simultaneous drop in hardware cost, which eroded the value of VMMs. As mainframes gave way to minicomputers and then PCs, VMMs disappeared to the extent that computer architectures no longer provided the necessary hardware to implement them efficiently. By the late 1980s, neither academics nor industry practitioners viewed VMMs as much more than a historical curiosity. Fast forwarding to 2005, VMMs are again a hot topic in academia and industry: Venture capital firms are competing to fund startup companies touting their virtual-machine-based technologies. Intel, AMD, Sun Microsystems, and IBM are developing virtualization strategies that target markets with revenues in the billions and growing. In research labs and universities, researchers are developing approaches based on virtual machines to solve mobility, security and manageability problems.

Things will develop in the opposite direction when they become extreme, the development of multitasking operating systems and hardware cost dropping prevent the evolution of VMMs in 1990s, but now they become supporting the revival of virtual technology. More and more powerful PCs with low cost emerge, which give virtual domain the opportunity to build multiple virtual machines under single PC. Less expensive hardware had led to a proliferation of machines, which were often underused and incurred significant space and management overhead. And the increased functionality that had made operating systems more capable had also made them fragile and vulnerable. To reduce the effects of system crashes and breakings, system administrators again resorted to a computing model with one application running per machine. This in turn increased hardware requirements, imposing significant cost and management overhead. Moving applications that once ran on many physical machines into virtual machines and consolidating those virtual machines onto just a few physical platforms increased use efficiency and reduced space and management costs. Thus, the VMM's ability to serve as a means of multiplexing hardware-this time in the name of server consolidation and utility computing-again led it to prominence.

Moving forward, a VMM will be less a vehicle for multitasking, as it was originally, and more a solution for security and reliability. In many ways VMMs give operating systems developers another opportunity to develop functionality no longer practical in today's complex and ossified operating systems, where innovation moves at a geologic pace. Functions like migration and security that have proved difficult to achieve in modern operating systems seem much better suited to implementation at the VMM layer. In this context, VMMs provide a backward-capability path for deploying innovative operating system solutions, while providing the ability to safely pull along the existing software base.

1.4.2 Virtualization Technique Advantage

By concluding, the virtualization technology can offer:

- Decouple Software from the hardware by forming a level of indirection between the software running in the virtual machine (layer above the VMM) and the hardware, which can let the VMM exert tremendous control over how guest operating systems—-operating systems running inside a virtual machine—-use hardware resources.

- Provide a uniform view of underlying hardware, making machines from different vendors with different I/O subsystems can run on any available computer. Thus, instead of worrying about individual machines with tightly coupled hardware and software dependencies, administrators can view hardware simply as apool of resources that can run arbitrary services on demand.

- Supply a complete encapsulation of a virtual machine's software state, the VMM layer can map and remap virtual machines to available hardware resources at will and even migrate virtual machines across machines.

- Load balance the computing tasks or applications among a collection of machines, let multiple users work together at the same time to improve efficiency.

- Show robust for dealing with hardware failures or for scaling systems. Administrators can suspend virtual machines and resume them arbitrary times or checkpoint them and roll them back to a previous execution state. With this general-purpose undo capability, systems can easily recover from crashes or configuration errors.

- The VMM can provide total mediation of all interactions between the virtual machine and underlying hardware, thus allowing strong isolation between virtual machines running in a single hardware platform or multiple hardware platforms by using some new virtual software like VMware [14]. This isolation idea between virtual machines is very valuable for reliability and security. Crash of the application in one virtual machine or compromised by attackers in one virtual machine can not influence the other running systems.

- Moreover, virtual technical gives us the opportunity of mobility, since users can copy a suspended virtual machine over a network or store and transport it on removable media, which will be made use of in many scenarios.

Thus, virtual technique, or VMM in specific, is a tool for restructuring systems to enhance robustness and security—-without imposing the space or management overhead that would be required if applications executed on separate physical machines.

1.4.3 Virtual Routing Capability

Since a lot of applications can work properly in virtual domain, benefit from this rebirth technique, we want to expand this into a more general way, actually in routing for specific. Using just one physical off-the-shell PC to run multiple virtual machines and making them to route packets simultaneously is what we want to implement. If it's possible and the overhead introduced by the VMM is not server, we can further take advantage of this idea to achieve the migration of our multistage software routers, from physical hardware into virtual machines. By doing this work, we can add complex functionalities just by bring them from general virtual world into our architecture correctly, such as isolation, slicing, consolidating the framework, energy saving and so on.

1.5 Outline and Contributions

1.5.1 Contributions

The main contribution of this thesis work is to find the feasibility of routing ability in virtual machines, testing the throughput based on flows through different virtual machines running inside the same hardware. Compare with the bare Linux PC in physical environment, the aggregation throughput of multiple virtual machines can achieve satisfied results under tense tests by changing traffic load, packet size and number of virtual machines running inside one physical PC.

After getting the basic idea of routing ability in virtual machines, this thesis tried to implement the multistage software routers inside virtual server. With some tests on the data plan and the correctness behavior of the multistage software routers, we show that it is possible to run this architecture in one virtual server (physical PC) but with some performance penalty.

1.5.2 Outline

The contents of each chapter are as follows:

- Chapter 1 gives a general idea of this thesis background and work.
- Chapter 2 shows our physical multistage software routers' architecture.
- Chapter 3 describes some virtualization techniques, our choices of the virtual software and topologies during all the tests. At last some results of routing ability in virtual environment are presented.

- Chapter 4 tries to migrate the physical multistage software router into virtual PCs thoroughly.
- Chapter 5 is a brief conclusion of what we have obtained in this thesis and future works we are interested in.
- Chapter 6 is references used during this thesis.

Chapter 2

Physical Multistage Green Software Routers

As mentioned in the first chapter, the final goal of the thesis is to run our multistage software routers inside virtual machines. By doing this, we can add new functionality such as slicing, security, energy saving etc. Nevertheless, the multistage software open routers' architecture has never been showed in detail. In this chapter, we will follow the idea from [15] and [16] and present the whole architecture of our multistage routers and some internal communication mechanisms which can maintain these routing elements as one router from the external administrator's point of view.

2.1 Architectural Design

Although single PC can work as software router with the help from open software such as Linux, Xorp and Click, the performance can not be compared with commercial equipment due to the limitations from bus bandwidth, CPU and memory-access speed. Besides performance reductions, router based on single PC has other criticisms against them, like scalability problems, lack of advanced functionalities, inability to support a large number of interfaces, as well as some resilience issues.

The exposed limitations of a single PC drove the design of a multistage architecture. We wished to exploit the optimal cost/performance features of standard PCs to design packet-switching devices beyond the limits of a single PC. Given the low cost per switched bit, we used a non minimal number of ports and switching elements in the multistage setup, while still being competitive on the overall cost. Several proposals were studied in the literature to implement multistage switching architectures. However, most exploit unidirectional transmission, for example, allowing to transfer information from input to output ports, and synchronous behavior, for example, assuming fixed packet size. Both assumptions fail in our case, since line

cards have physically bidirectional links and packets are of variable size. Moreover, according to the PC-based software router idea outlined in the Introduction, we wish to use only off-the-shelf networking and relatively cheap PC hardware.

Under this circumstance we propose a multi-stage software routers architecture shown as figure 1, to overcome the limitations of single PC acting as router and supply some advanced functionality contained in commercial routers. Our novel multi-stage architecture exploits classical PCs as elementary switching elements to build large routers, and it must appear to other routers and to network administrators as a single, large router. It composes three stages as sketched in figure 2.1. The front-NICs of load balancers, on the leftmost part in the figure, act as router I/O cards. The architecture encompasses a first-stage of load-balancing switches (L2-balancers), and a back-end stage of IP routers (L3-routers), interconnected by means of standard Ethernet switches. Both L2-balancers and L3-routers are standard PCs equipped with several NICs.

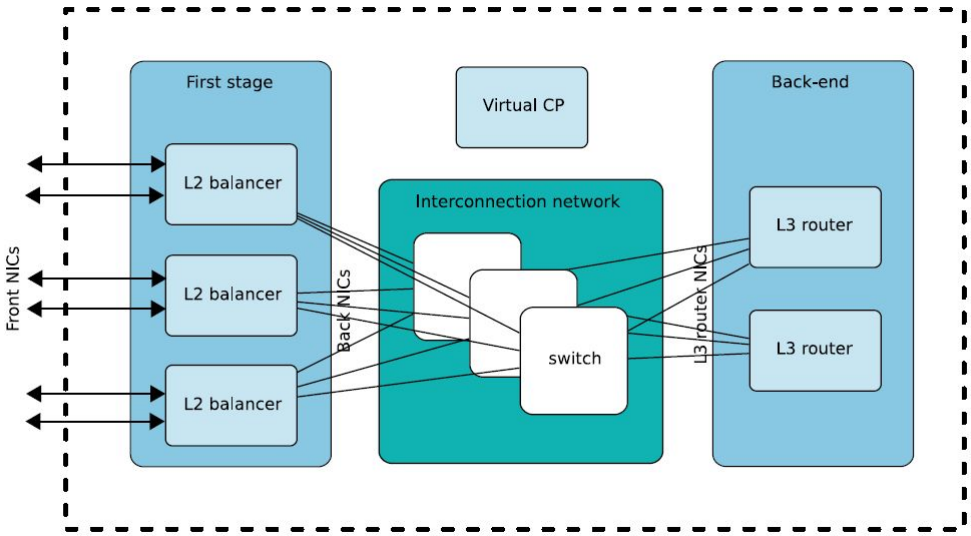

Figure 2.1: Multi-stage Software Routers Architecture

Packets arriving at the router input ports are:

1. Received by a L2-balancer front-NIC, processed by the balancer CPU to perform simple and fast load balancing among back-end routers, and then transmitted by the L2- balancer back-NIC toward the interconnection network;
2. Switched by the interconnection network to the appropriate L3-router NIC;
3. Received by the L3-router and processed by its CPU to perform the required packet operations, then transmitted toward the interconnection network;

4. Switched by the interconnection network to the right L2- balancer back-NIC;

5. Received by the L2-balancer back-NIC, processed by balancer CPU to switch the packet toward the appropriate front-NIC, then transmitted toward the next-hop node.

2.2 Interoperability

Combining multiple PCs to make them working together as routers is not rare. Some research groups already show the possibility such as [11] and [12] mentioned before. But most of the results are like multi-tasking operating system or distribution work load to different PCs, which is totally different in our proposal. In our three stages architecture, every single element can not work as router on its own. Each element performs a specific task to maintain the overall architecture works as a single router to outside world. In the following part we will show the details of each category's functionality to sustain this novel design.

2.2.1 Frond-end Side Load Balancer

When packets enter the multistage router from outside, the load balancer adapts the packets' layer-2 framing formats when they are forwarded from the front-NIC to the back-NIC. Several algorithms can be implemented, from a simple round-robin scheme to more complex algorithms that, for example, guarantee in sequence routing of packets [17], or balance packets to a particular L3-router based on QOS parameters. Load balancing is obtained simply by setting the destination MAC address of the Ethernet frame, so that the correct L3-router Ethernet NIC is addressed.

When packets return from the inside network, they are received by L2-balancer back-NIC and must be switched according to the destination MAC addresses to the corresponding front-NIC. The back-NIC driver has a static forwarding table, storing next-hop MAC addresses of network devices connected to the front-NIC. When a packet is received by the back-NIC driver, a look-up in the forwarding table is performed to call the correct front-NIC transmission function, causing therefore a direct transmission of the (unmodified) frame toward the next-hop. An additional entry is used to deal with broadcast/multicast messages, to correctly copy them to all front-NICs.

2.2.2 Interconnection-Network

This is a standard Ethernet network, eventually with multiple paths between the load balancers and back-end forwarding engines (FE) to support fault recovery and

to upgrade the switching capability of the router. Indeed, the load-balancing among L3-routers is achieved by addressing the corresponding L3-router input-NIC MAC address. According to the backward learning algorithm, the interconnection network is a standard switching behavior network. There is no need to change the normal operation of Ethernet switches, and therefore reduced costs by using just common devices.

2.2.3 Back-end Side Routing Array

Operations involved in the back-end routers are standard IP routing and packet manipulation, no changes are required compared to the standard feature set a single-box router implements. All layer-3 routers must be correctly configured, IP routing tables, firewalls, ACL rules etc, must be correctly set up.

But there is one thing we need to specify, the upgrade of routing table through inter-communication between all the routing elements. Since we configure the multi-stage architecture as a single router appeared to the outside network, there should be just a single routing table, and not allow to be changed by every router in the back-end stage. In order to achieve this, we add some tiny communication traffic between them, maintaining the integrity of the architecture and unifying the routing table. This traffic is send from the virtual control processor (CP) shown in the above of the figure, received and processed by all the routing elements.

2.2.4 Virtual Control Processor

There is a single control processor in every router entity, managing the routing table update, tuning the router parameters and adjusting the admissible control policies, this control processor (CP) is also appeared in our multistage routers but give a name as virtual CP. Basically, the functionality is the same as physical one, but it should be manage all the router elements inside the layer-3 stage. When system administrators define any rules or our router learned any new routing entries, the information must be distributed to all the L3-routers identically, making them totally the same when routing packets accordingly.

As mentioned before, there is a new task for virtual CP—-maintaining the unity of our multistage software routers. We implement a new protocol named DIST to achieve this [16]. Both the simulation results and theoretical analysis show that with negligible overhead introduced by DIST protocol, we can success to make all the L3 back-end routers work as a single router.

In order to enhance the resilience ability against failures or attacks from outside, we do not use a specific equipment to perform the virtual CP functions, but by

configuring the back-end routers to let them supplying the features of the CP. Since there are multiple L3 routers, we need to choose only one acting as virtual CP. From the stochastic point of view, each of the L3 routers has the same probability to be corrupted, so we just choose a master router, which is a random choice, to perform all the CP functions and tell the other slaves what to do through internal traffic. If the slave down, nothing changed but can be viewed as some performance decrease, if the master down, a new election phase occurs and switches one slave to master with minimal overhead compared with the normal traffic a router holds. In summery, we integrate the functionalities of the CP into the L3 back-end routers with minor modifications to improve the multistage software routers failure resistance.

2.3 Peculiar Features

The most special feature in our architecture is that all the elements can not work as single router as in most design philosophy, but only act as a gear in a big machine. This framework can result in lots of advantages such as:

- Overcome performance limitations of a single-PC-based router by offering multiple, parallel data paths to packets;
- Upgrade router performance by incrementally adding more switching elements, or incrementally upgrading each switching element;
- Scale the total number of interfaces the node can host, and, as a consequence, the router capacity;
- Automatically recover from faults, for example, reconfiguration can occur in case of any PC/element failure even in the virtual CP as explained just now;
- Not like high-end routers synchronous behavior, our solution supports a fully asynchronous behavior. Therefore there is no need for the global clock distribution, a complex task that requires large power consumption.
- No need to restrict the router working on fixed-size data units, which implies segmentation of the variable-sizes IP packet at inputs and re-assemble them at the outputs, as the high performance commercial routers always do.
- Provide functional distribution, to overcome single-PC CPU limitations, for example, allowing the offloading of CPU intensive tasks such as filtering, cryptography to dedicated PCs.

Object has its two sides, there are also some drawbacks in the architecture. The most obvious one is that each of the elements has no ability to achieve the routing

functions. Lot of efforts are needed to coordinate the single switching elements on the data, control and management planes, so as to make the interconnection of PCs behave as a single large router. However, in our architecture no chip re-design efforts are needed, since the building blocks are off-the-shelf components, whose performance will increase independently thanks to PC market evolution.

Another one is by increasing the number of the PCs in the first and third stages, along with the increase of the interfaces and interconnection switches, or use Field Programmable Gate Array (FPGA) to design specific interfaces, substituting the original ones in order to improve the performance, the overall cost is not cheap compare with traditional stand along PC working as router. But compared with commercial solutions based on the development of Application-Specific Integrated Circuits (ASICs), the expenditure is reduced effectively.

By concluding, our novel multistage software routers can sustain a large routing capability by increasing elements as will but keep a low overall cost compare with commercial equipments. With minor modifications in the Linux, Xorp, Quagga or CLICK thanks to the open environment, we can success configuring the three stages architecture working as a single router, implementing basic functionalities currently. This thesis following parts will try to search the routing capability of virtual machine and eventually migrate the physical multistage software routers into virtual domain, increasing new potential functions to the final product.

Chapter 3

Virtual Machine Routing Capability Seeking

Before implementing the multistage software routers virtually, the primary thing we need to solve is that whether or not the virtual machines can act as router. Then comparing with the physical PC, we want to know the performance penalty, if it exists, introduced by the VMM running between the hardware and virtual machines. Also we hope to get the details about multiple virtual machines running in one physical PC, aggregation throughput, fairness behave and performance impairment due to different number of guest operating system. All of these will be uncovered in this chapter.

3.1 Virtualization Techniques

To virtualize means to take something of one form and make it appears to be another form. Virtualizing a computer means to make it appear to be multiple computers or a different computer entirely. Virtualization is not a new topic since it has nearly four decades history [18]. IBM Inc, Massachusetts Institute of Technology (MIT) and Manchester University are the pioneers of this technique. Up to now, more and more companies and research institutes such as Intel, AMD, Sun Microsystems realize the importance of virtualization, found new projects to design improve own virtual ability.

Among all the virtual aspects, hardware virtualization, including memory reuse, interface multiplexing, I/O equipment and bus sharing, processor virtualization, which stands for masquerade one system to another like running Java Virtual Machine in Microsoft Windows operating system, and the Instruction set virtualization or binary translation, a new model of translating virtual instruction set to the physical instruction set of the underlying hardware dynamically to accelerate the system

operations, are the most common ones under study. By combining these novel aspects of virtualization, academia and industry delimits some more clear ideas and methods in virtualization domain as described in the following.

3.1.1 Types of Virtualization

When it comes to virtualization, there's not just one way to do it. In fact, there are several ways that achieve the same result through different levels of abstraction. This section will introduce four of the most common methods of virtualization in Linux and identifies their relative strengths and weaknesses. The industry sometimes uses different terms to describe the same virtualization method. The most common term is used here, with references to the other terms for consistency.

Hardware emulation

Arguably the most complex of the virtualizations is provided by hardware emulation. In this method, a hardware VM is created on a host system to emulate the hardware of interest, as shown in Figure 3.1.

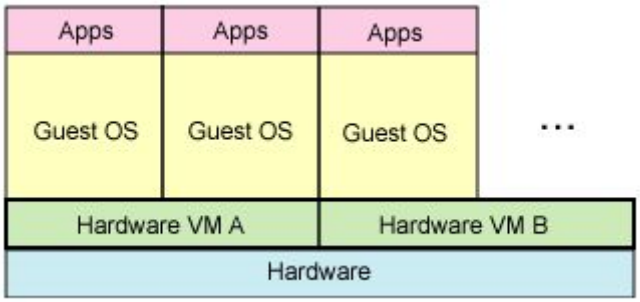

Figure 3.1: Hardware Emulation uses a VM to simulate the required hardware

As you can probably guess, the main problem with hardware emulation is that it can be excruciatingly slow. Because every instruction must be simulated on the underlying hardware, a 100 times slowdown is not uncommon. For high-fidelity emulations that include cycle accuracy, simulated CPU pipelines, and caching behaviors, the actual speed difference can be on the order of 1000 times slower.

Hardware emulation does have its advantages. For example, using hardware emulation, you can run an unmodified operating system intended for a PowerPC? on an ARM processor host. You can even run multiple virtual machines, each simulating a different processor.

One of the most interesting uses of hardware emulation is in co-development of firmware and hardware. Rather than wait until the real hardware is available,

firmware developers can use target hardware VM to validate many aspects of their actual code in simulation.

Full virtualization

Full virtualization, otherwise known as native virtualization, is another interesting method of virtualization. This model uses a virtual machine that mediates between the guest operating systems and the native hardware (see Figure 3.2). "Mediate" is the key word here because the VMM mediates between the guest operating systems and the bare hardware. Certain protected instructions must be trapped and handled within the hypervisor because the underlying hardware isn't owned by an operating system but is instead shared by it through the hypervisor.

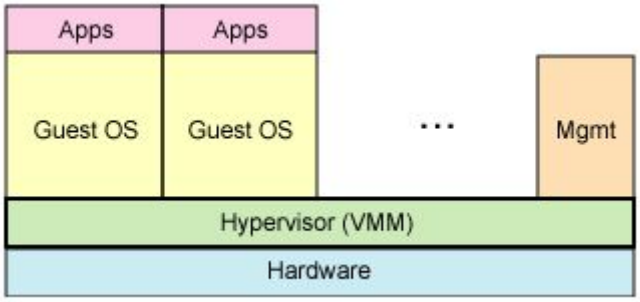

Figure 3.2: Full virtualization uses a hypervisor to share the underlying hardware

Full virtualization is faster than hardware emulation, but performance is less than bare hardware because of the hypervisor mediation. The biggest advantage of full virtualization is that an operating system can run unmodified. The only constraint is that the operating system must support the underlying hardware (for example, PowerPC). Some older hardware, such as x86, create problems for the full method of virtualization. For example, certain sensitive instructions that need to be handled by the VMM do not trap. Therefore, hypervisors must dynamically scan and trap privileged-mode code to handle this problem.

Paravirtualization

Paravirtualization is another popular technique that has some similarities to full virtualization. This method uses a hypervisor for shared access to the underlying hardware but integrates virtualization-aware code into the operating system itself (see Figure 3.3). This approach obviates the need for any recompilation or trapping because the operating systems themselves cooperate in the virtualization process.

As mentioned before, paravirtualization requires the guest operating systems to be modified for the hypervisor, which is a disadvantage. But paravirtualization offers performance near that of an un-virtualized system. Like full virtualization, multiple different operating systems can be supported concurrently.

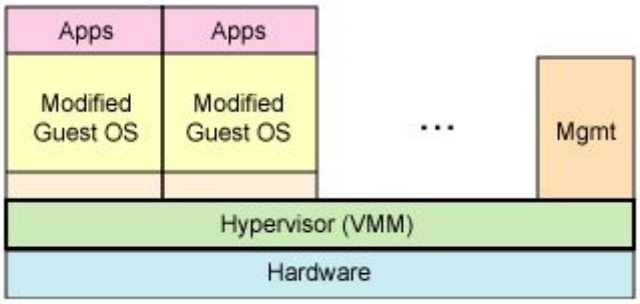

Figure 3.3: Paravirtualization shares the process with the guest operating system

Operating system-level virtualization

The final technique we'll explore, operating system-level virtualization, uses a different technique than those covered so far. This technique virtualizes servers on top of the operating system itself. This method supports a single operating system and simply isolates the independent servers from one another (see Figure 3.4).

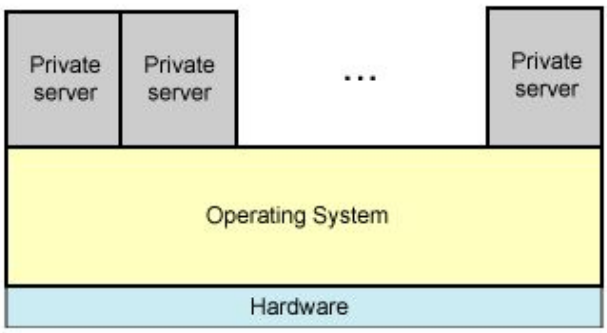

Figure 3.4: Operating system-level virtualization isolates servers

Operating system-level virtualization requires changes to the operating system kernel, and some of the features such as I/O guest code will be frozen in user mode but handled by the kernel mode. The advantage is also native performance compare to physical server.

From this simple introduction of different types of virtualization techniques, we can see that there is a trade-off between the complexity and performance. Actually, the central design goals for virtualization technique or VMM (supervisor) which is the key in virtualizing, are compatibility, performance and simplicity. Compatibility is clearly important, since the VMM's chief benefit is its ability to run legacy software. The goal of performance, a measure of virtualization overhead, is to run the virtual machine at the same speed as the software would run on the real machine. Simplicity is particularly important because a VMM failure is likely to cause all the virtual machines running on the computer to fail. In particular, providing secure isolation requires that the VMM be free of bugs that attackers could use to subvert the system. So selecting one suitable technique to implement our multistage software is the base to the further works. We will introduce two well-known virtualization software/projects related to Linux, which is the open source system that can let anyone modify based on their own purpose, and it's the platform of our multistage software routers.

3.1.2 Linux-Related Virtualization Projects

Our initial aim of this project is to build open-source, easy-modified infrastructure to cooperate with commercial equipment for routing packets. From the hardware chosen like off-the-shell PCs and the routing software deployed such as Xorp and Click, we intend to use free of charge, easy modified open source software to accomplish this architecture. Things develop to the virtual domain, we want to keep this initial idea, choose some free vitalization software with source public. Linux-related virtualization software just gives us the opportunity to achieve this. Developing for nearly forty years, there are lots of projects funded to virtualizing, some of them are shown in table 3.1.

Project	Type	License
Bochs	Emulation	LGPL
QEMU	Emulation	LGPL/GPL
VMware	Full virtualization	Proprietary
z/VM	Full virtualization	Proprietary
Xen	Paravirtualization	GPL
UML	Paravirtualization	GPL
Linux-VServer	Operating system-level virtualization	GPL
OpenVZ	Operating system-level virtualization	GPL

Table 3.1: Linux-related virtualization projects

The advantages and shortcomings of different virtualization techniques have already been discussed. Emulation is mostly for testing new software code based on the under-developing system. Operating system level virtualization needs to modify the master system itself, which may introduce some incompatibility problems. Full virtualization is the perfect choice for our case, since it do not need any modifications to the guest operating systems, that's make the migration convenient to accomplish, and the performance is roughly the same as physical Linux. Here the performance means a general idea, it stands for the I/O reading, processes implementing or switching context in the Linux operating system. But when task comes to the routing ability, everything is agnostic. We need to carry out lots of experiments to find out this capability.

Since VMware is a commercial solution for full virtualization with free version such as VMware ESXi, VMware vSphere Client [20], we intend to use it as our choice. In VMware a hypervisor sits between the guest operating systems and the bare hardware as an abstraction layer. This abstraction layer allows any operating system to run on the hardware without knowledge of any other guest operating system. VMware also virtualizes the available I/O hardware and places drivers for high-performance devices into the hypervisor. The entire virtualized environment is kept as a file, meaning that a full system (including guest operating system, VM, and virtual hardware) can be easily and quickly migrated to a new host for load balancing. That makes perfectly for our case when upgrading the routing elements in the multistage architecture.

VMware is a new solution with enhance ability such as new drivers and good graphic user interface. We think these can give us better performance and management when running it. In order to compare with the routing performance, we also choose one classical virtualization software XEN [21] in Paravirtualization for comparing. Recall that in paravirtualization the hypervisor and the operating system collaborate on the virtualization, requiring operating system changes but resulting in near native performance. As Xen requires collaboration (modifications to the guest operating system), only those operating systems that are patched can be virtualized over Xen. From the perspective of Linux, which is itself open source, this is a reasonable compromise because the result is better performance than full virtualization. But from the perspective of wide support (such as supporting other non-open source operating systems), it's a clear disadvantage. But as pointed out, everything can be different in routing packets, and from our huge amount of tests, the routing ability in VMware is better than XEN with minimal modifications in the operating system (No patches for enhancing networking functions, no parameters tuning in the Linux IP routing stack and so on, in both VMware and XEN). These will be shown in the following part, along with the explanations.

3.2 Virtual Environment Implementation

In this part, the experimental conditions will be explained thoroughly, such as the hardware preferences, especially the Ethernet Card used and CPU in the mother board, software installation's hints, concluding the important things we need to deal with, and the topologies when testing the routing abilities of the virtual software routers.

3.2.1 Hardware Descriptions

The routing capability of virtual machine installed with Linux operating system is the final aim in this chapter. Basically, the things need to be cared of in virtual environment are similar with a physical PC when routing packets. Normal PC comprises three main building blocks: the CPU, random access memory (RAM), and peripherals, glued together by the chipset, which provides advanced interconnection and control functions. As sketched in Figure 3.5, the CPU communicates with the chipset through the front-side bus (FSB). The RAM (memory) provides temporary data storage for the CPU, and can be accessed by the memory controller integrated on the chipset through the memory bus (MB). Interfaces are connected to the chipset by the Peripheral Computer Interconnect (PCI) shared bus, or by a PCI-Express (PCIe) dedicated link.

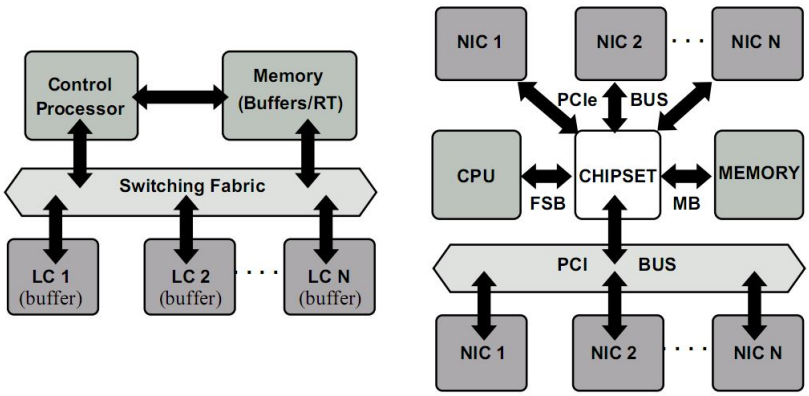

Figure 3.5: Schematic description of a packet router (left) and of a PC architecture (right)

Today's state-of-the-art CPUs run at frequencies 3.0 GHz with four or eight cores usually. The front-side bus is 64-bit wide, allowing for a peak transfer rate ranging from 3.2 Gbyte/s to 42.66 Gbyte/s. The memory bus is usually 64-bit wide and provides a peak transfer rate of up to 5.33 Gbyte/s. In high-end PCs, the memory

bandwidth can be further doubled or quadruplicated, bringing the bus width to 128 or 256 bits, by installing memory banks in pairs.

The PCI protocol is designed to efficiently transfer the contents of large blocks of contiguous memory locations between the peripherals and the RAM, using Direct Memory Access (DMA), without requiring any CPU intervention. Depending on the PCI protocol version implemented on the chipset and the number of electrical paths connecting the components, the bandwidth available on the bus ranges from about 125 Mbyte/s for PCI 1.0, which operates at 33MHz with 32-bit parallelism, to 4 Gbyte/s for PCI-X 266, when transferring 64 bits on a double-pumped 133 MHz clock. Similarly, when a PCIe channel is used, a dedicated serial link between the peripheral and the chipset is used, called lane. PCIe transfers data at 250 Mbyte/s per lane. With a maximum of 32 lanes, PCIe allows for a total combined transfer rate of 8 Gbyte/s.

When connected by means of either a PCI or PCIe bus, (Network Interface Cards) NICs allow a PC to receive and transmit packets, acting as router LCs. Therefore, by comparing the router and the PC architecture, it can be easily noticed that common PC hardware enables to easily implement a shared bus, shared-memory router. NICs receive and transfer packets directly to the RAM using DMA. The CPU performs packet forwarding by implementing in software the longest-prefix matching algorithm, and then routes packets to the right buffer in RAM, from which NICs fetch packets for transmission over the wire.

The parameters need to be refined in a PC architecture router are CPU operation speed, network interface cards transmission speed with PCI or PCIe considered and memory size. From a general point of view the higher these parameters are, the better a software router performs. In my experiment there are two PCs acting as the virtual servers. Besides these, a commercial router tester or traffic generator is used for testing the performance of the software routers for precision. And a controlling laptop is used in VMware as described below. All of these devices are connected with a standard layer 2 switch in our lab.

The first server is Supermicro C2SBX with Intel? Core? 2 Extreme, Quad, Duo processors E6750 @ 2.66 GHz, 4096 KB cache size. In the flags of the CPU, it supports pae for paravirtualization and vmx for full virtualization both. The RAM size is 2GB*4, totally 8 GB. The huge memory size is for multiple virtual machines running inside this server. Network interface is Intel (R) pro 1000 dual port card working in 1000 base T, full duplex always. The slot in the mother board for the NIC is PCI-E X16 Gen 2. Second one is a new server purchased not long ago. In order to keep the consistent of the test performed, we try our best to keep the parameters same. The server is Dell Inc Power Edge T100, with Intel (R) Xeno (R) E3110 CPU running at 3.0 GHz, and the cache size is increased to 6144 KB. These

upgrades of CPU frequency and cache size will not influence the performance of the routing ability too much if keeping the other key components the same. Also in this new server it supports pae and vmx for virtualization technologies. The memory size and NIC are exactly the same as the first server. The 2 servers are installed with virtualization software and running tests as shown in figure 3.6.

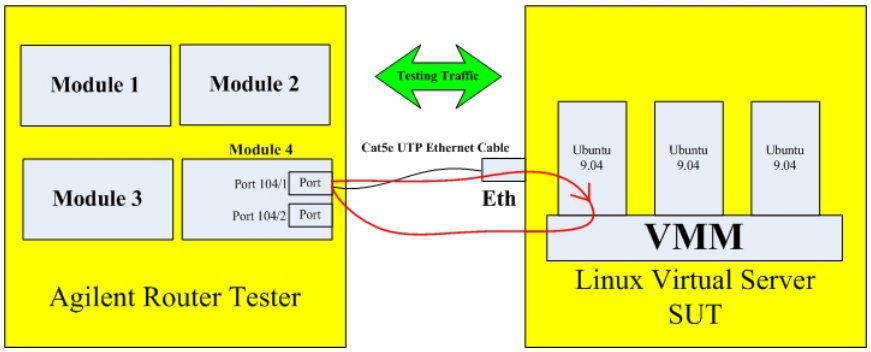

Figure 3.6: Basic structure of the routing ability test

This is just a general idea of the routing test. Packets are generated from commercial equipment Agilent Router Tester, going through the SUT (System Under Test) and come back to the Router Tester. Information is collected directly from the traffic generator for accuracy. The Agilent N2X Router Tester equipped with 4 modules, 2 ports on each of them and therefore 8 Gigabit-Ethernet ports totally. It can transmit and receive Ethernet frames of any size at full rate. Tests running with it are surely trust worth compare to software traffic generator.

In some cases there are multiple PCs involving for the tests. Then a high performance switch is needed. We use an Optronics 24 ports Gigabit switch to eliminate the bottle neck in the middle equipment for sure.

3.2.2 Software Deployment

As sketched before, the virtual routing tests are performed in XEN and VMware for comparing the routing ability in 2 different virtual technologies. Since the VMware ESXi is very new and the concepts of management/controlling inside are totally different from the original designing map, it is worth to show some details about how to install and use it. Xen is classic, but the command line only management method make it a little harder to use than VMware. From the perspective of recurrence all the tests in easily, the software deployment will be reviewed shortly.

VMware

VMware is the recognized leader in virtualization technology solutions proven to increase the utilization of existing hardware and reduce capital and operational costs throughout the organization. It can help consolidate servers, optimize software development, improve enterprise desktop management and provide affordable business continuity. Among the products aiming to different user groups, we choose VMware ESXi, which is free but keep evolution as time pass on. Actually, during the thesis work, there come 2 versions of ESXi, 3.5 released in 31/03/2009 and 4.0 in 22/06/2009. Since in version 3.5 there are some strange behaviors when routing packets in 256 byte and 512 byte, we keep the CVS active and install 4.0 when it has been released. The 2 versions are more or less the same to install, but with some changes to the hardware limitations that must be pointed out.

In VMware ESXi 3.5, for enhancing and extending the server usage, it can not be installed in a normal IDE (ATA) hard disk but only SATA (Serial ATA) hard driver. This can be understood, since IDE driver reach the limit of data transfer capability in 133MB/s. The data loss and electromagnetic interference prompted the SATA and improve the transferring speed up to 600 MB/s. The most beneficial of virtualization is run multiple operating systems on a single server. This purpose needs strict performance index for the hard disk. So the SATA become the ideal choice of the VMware installation driver. Besides this, in the BIOS settings, SATA RAID Enable needs to be active. This is very important since lots of users can not install ESXi because of this. If missing this option in the BIOS setting, no hard driver can be recognized or unstable VMM will be installed base on other abnormal conditions. After all of these have been done, there should be no difficult to install VMware ESXi 3.5 to the server.

In VMware ESXi 4.0, things are a little different. Since there is no SATA RAID Enable option in our new DELL server, 3.5 can not be installed on it for a long time. But when version 4.0 has been released, it can work properly in the new server, only with cautions of SATA hard disk chosen. VMware developing team consolidates the compatibility of ESXi for sure.

As mentioned before, the controlling method is a little novel but convenient in VMware. After the installation of the software in the server, it can not be manipulated so much directly. Configuring some Vlan or IP addresses in the server console are all the things we can do except shut down/restart it. In order to use the server deeply, another software named Vi Client for 3.5 or vSphere Client for 4.0 is need to be installed in other PC, acting as the monitor to the virtual server. There are two versions for windows and Linux operating system. During my thesis windows version has been chosen for a better graphic user interface. In the monitor PC lots of functions can be chosen such as creating/destroying the virtual machines,

monitoring the status of the virtual server and virtual guest systems, changing the network topology as will and so on.

XEN

The XEN hypervisor, the powerful open source industry standard for virtualization, offers a powerful, efficient, and secure feature set for virtualization of x86, x86_64, IA64, ARM, and other CPU architectures. It supports a wide range of guest operating systems including Windows, Linux, Solaris, and various versions of the BSD operating systems. XEN installation is harder than VMware, but it can be monitored directly from the XEN server, without the help from external controlling PC. Actually, XEN must be installed onto a basic Linux Operating System. Since all my tests are performed in Ubuntu 9.04, with kernel version 2.6.28-11, least when this thesis has been done, XEN is also installed inside Ubuntu 9.04.

In XEN the VMM is always a standard PC in Linux operating system called dom0 and the guest systems are called domU, which should be booted form dom0 only. As dom0 is a standard Linux operating system with slight modifications, it can be created based on the running Linux system using synthetic packet manager (binary packet) or can be compiled form source codes. During my tests, both solutions have been tested, they work exactly the same. But compiling the codes, creating a new kernel use menuconfig and patches the kernel is a little harder to operate. By modifying the grub in Ubuntu, the new XEN kernel can run finally.

After the XEN dom0 has been configured correctly, when booting the system, there will not be the standard opening screen but lots of parameters verification. As soon as the system starts, everything should be the same like normal Linux operating system, with the ability to run multiple domU guest systems on it. The domUs are virtual machines in XEN. Therefore they have separate user space (hard disk) to store there own stuffs. It is necessary to use dd command to create the image file and swap file for the guest operating systems. Then mkfs .ext3 and mkswap are needed to format the new virtual partitions. Install the target system into the new partition is just a standard operation, nothing new include. When all of these have been done, there is a domU system template, copy it for more systems or create new system image file are both fine for other domU.

Booting domU in XEN is based on the configuration file, which is like the following one more or less. From this file, lots of options can be played with, but in our case, the most interesting part are lied on routing packets, so IP address and MAC are the parameters we need to pay attention to. Others can be used with the default. More detail information can be found in [21]. There is a very important tip. If the operating system is Ubuntu, the disk option should be tap:aio as the file shows, but if the system is others like Debian or FreeBSD, the disk option should

be file which substitute tap:aio in the file.

Configuration file in XEN when start a domU:

```
kernel = '/boot/vmlinuz-2.6.24-24-xen'
ramdisk = '/boot/initrd.img-2.6.24-24-xen'
memory = '1024'
root = '/dev/hda2 ro'
disk = [
        'tap:aio:/home/xen/domains/xen1.example.com/swap.img,hda1,w',
        'tap:aio:/home/xen/domains/xen1.example.com/disk.img,hda2,w',
        ]
name = 'xen1.example.com'
vif  =  [
        'ip=192.168.85.236,mac=00:16:3E:62:DA:BB,bridge=eth0',
        'ip=192.1.1.5,mac=00:16:3E:62:DA:B1,bridge=eth3',
        'ip=192.2.1.5,mac=00:16:3E:62:DA:B2,bridge=eth3'
        ]
on_poweroff = 'destroy'
on_reboot = 'restart'
on_crash = 'restart'
extra = '2 console=xvc0'
```

There are lots of commands in dom0 to control the virtual machine such as xm create -c to start the virtual machine, xm shutdown to close VM, xm list to show the running guest system, xm vcpu-set to change the virtual CPU numbers of the VM and so on. But remember, all the commands need to be run after the xend process active in dom0, which can be treated as the parent of the domU. Another important thing need to be considered is the bridge configuration in XEN. That is the default connection within dom0 and domU. In our case, most of the time we just bind every interface in domU with the same physical interface card in dom0, for comparing the performance with VMware and with physical server with that specific NIC.

Other software used are common ones such as remote desktop connection, ultra-VNC Viewer in windows operating system or vncviewer, rdesktop in Linux system to entering the server of Router Tester. No more else involved during this thesis work.

3.2.3 Experimental Description

This chapter reveals the performance of the virtual machines' routing ability from several aspects as listed below:

- The aggregation throughput of multiple VMs running inside one physical server, with different packet size considered obeying the restriction of Ethernet standard.
- The effects of multiple VMs with normal use when one virtual machine is routing packets in one physical server, with different packet size considered obeying the restriction of Ethernet standard.
- Fairness tests between different VMs routing ability inside one physical server, with 64 byte and 1500 byte packet size considered.
- Dom0 and physical server comparison when routing packets in XEN, with different packet size considered obeying the restriction of Ethernet standard.
- Routing ability evolution seeking in VMware ESXi 3.5 and 4.0.

Since VMware and XEN are totally different from the management or manipulation point of view, there should be 2 topologies for each of them. Figure 3.7 shows the basic setting of all the experiments performed in VMware, while figure 3.8 offers the topology in XEN.

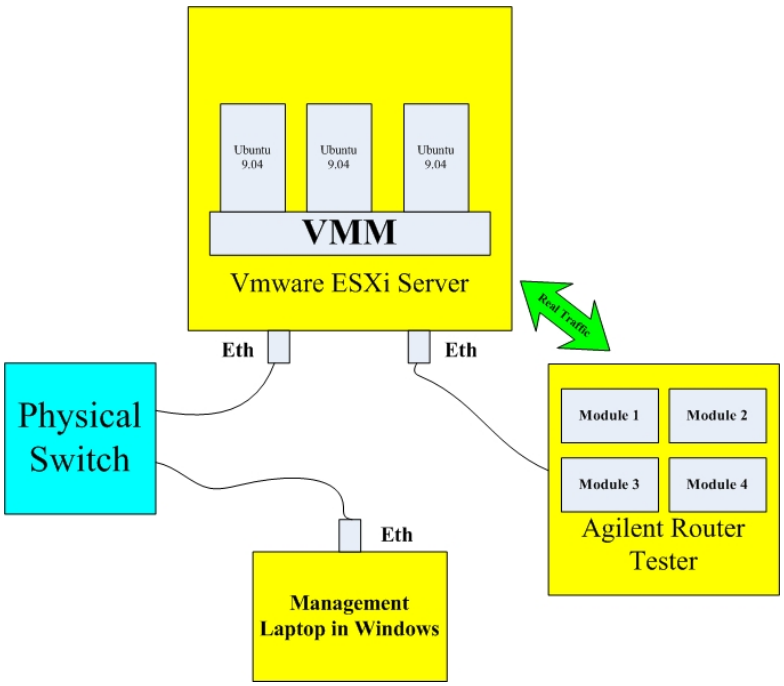

Figure 3.7: VMware Experimental Architecture

In VMware, there is no direct control of the server from itself, a management PC should be introduced. In my experiment the laptop was used to fulfill this task.

The server is separated from the user. As long as the password and placement of the server are safe, nearly no interference can occur on the server. This design philosophy enhances the security of the virtual architecture, gives us the opportunity to manage multiple virtual servers by using one controlling PC. But this topology gives us one drawback—-one extra NIC is needed in the server to communicate with the management laptop—-because our tests performed are intensive especially for the NIC. By separating the controlling port and data port can give us better performance. Since the packets between the virtual serve and the laptop are not so many compare to the data or "real traffic" in the picture, the Ethernet card choice is not critical. The physical switch here can be used for further upgrade such as increasing the servers and add sniffers. Agilent Router Tester and the server data port are connected directly using cat5e UTP cable directly, for minimize the undesired traffic loss or latency during the experiment. Only one physical data NIC is used in VMware tests, XEN tests and physical server routing ability tests for comparing. But in the virtual server, there will be several guest operating systems running at the same time, with multiple interfaces active each, which are mapped to the same physical interface by a bridging setting.

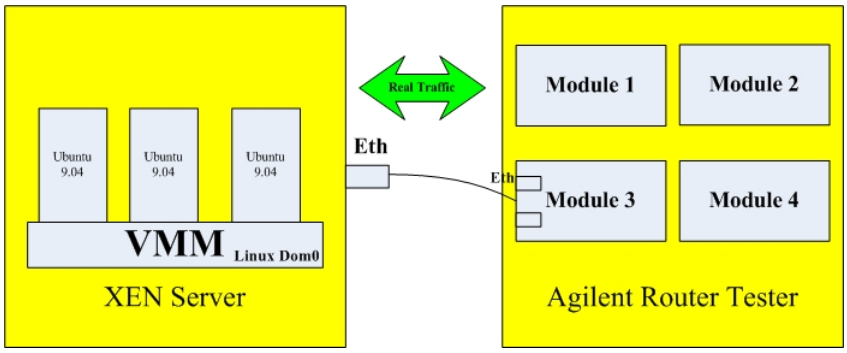

Figure 3.8: XEN Experimental Architecture

XEN is not like VMware since all the management of the virtual machines can be performed directly inside the server. The virtual server is running a dom0 Linux operating system all the time, the virtual server can be operated as a normal Linux, so there will be only one NIC to perform the data port in the server. When running several virtual operating systems, all the virtual NICs are mapped to the data port by using the bridge as VMware. The upgrade in XEN is not so easy like VMware does. Besides physical switch is needed, there should be some configurations manually to make them working together. In order not to introduce any interference of the traffic between virtual server and router tester, there is no switch in this topology.

Besides these physical topology and virtual technique difference, the experiments are the same in both environments. From this moment on, the physical topology will not be considered anymore. Running virtual machines will be treated as normal Linux operating system just like physical one.

3.3 Topology Setup and Numerical Results

Searching the routing ability in virtual domain is a new topic but with some interesting results already, such as [22] shows some performance of different forwarding options in XEN like bridged, routed configuration, [23] come up with some optimal parameters tuning for improving the performance when routing packets. But nearly all of these results are in XEN, a paraviratualization technique, and the idea of the tests is experimenting different topologies/settings for searching a better routing configuration in virtual machine. No tests have been done in VMware, a full virtualization technique and no tests are emphasized on the influence between the virtual machines inside the physical server, using the same interface to routing packets at the same time. These are what have been done in this chapter.

3.3.1 Single Virtual Machine Performed as a Router

The first category of the tests is intended to search the influence of the virtual machines running inside one physical server when only one VM route packets. All the VMs are deployed with the Ubuntu 9.04 Linux operating system, but only one has been active to route packets from one subnet to another, by changing the /proc/sys/net/ipv4/ip_forward value to 1 in the Linux. Then this VM has the ability to route packet as long as the routing table has been set correctly.

Since at the beginning of the virtual machines' routing ability seeking, we do not want to introduce too many overheads about looking up the routing table or learning new entries by using routing protocol such as OSPF or BGP, only static routing table is used in the tests. And for simplicity, 2 subnets are used when only one VM is routing packets. The topology of the tests is shown as figure 3.9.

In the Agilent Router Tester, only one module one port has been used to connect with the data port on the server, through a standard cat5e UTP cable. The VM (R) has been configured to routing packets in the figure, using 2 virtual interfaces inside it. The virtual interfaces and the physical interface have been connected through a virtual bridge both in VMware and in XEN. All the traffics will go through the data port in the server. By shutting down redirection ability in the Linux system, we can ensure the traffic make a loop from the router tester and the VM router correctly. Ifconfig and add route commands in the Linux system are used to set the

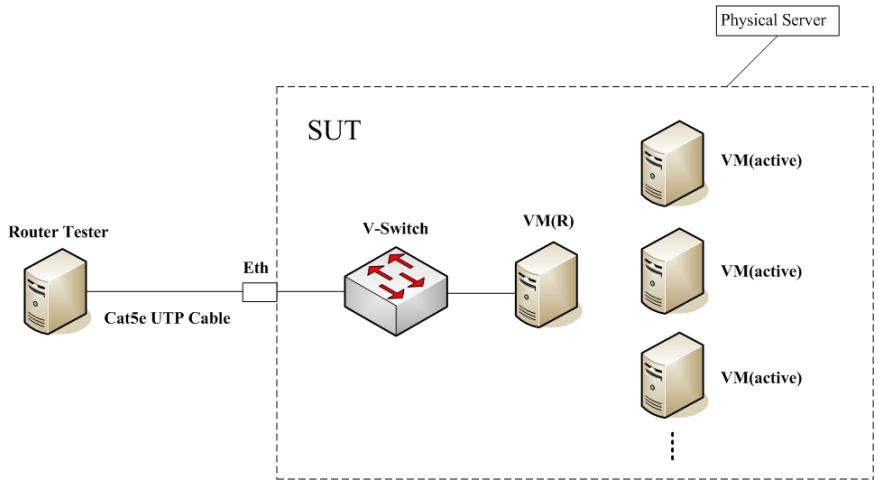

Figure 3.9: One VM acts as router with multiple VMs just open

IP addresses and routing table. Besides the configuration in VM router, the Router Tester should be set correctly, especially the next-hop MAC address is important.

In this experiment, all the other VMs are open to consume the server's resource such as CPU, context switch and internal communication traffic. We want to find out how deeply the running VMs influence the routing ability. This is very important since if adding the energy saving functions in our multistage architecture, the impact of opening machines but with low work load is just like the experiments carried out below.

The parameters used are reported as below when doing the testes:

- Packet Size : 64, 128, 256, 512, 1024, 1500 Bytes.
- Traffic Load : 50, 100, 200, 300, 400, 500, 600, 700, 800, 900, 990 Mbps.
- Physical Link Speed : 1Gbps.
- Number of Agilent Modules/ports : 1 Module with 1 port.
- Routing Table : 2 subnets with 2 entries.
- Time : 30 seconds for each packet size with each load.

VMware ESXi version 3.5 and 4.0 have been installed into 2 different servers as mentioned before, due to the compatibility issues from the VMware HCL (Help Compatibility on-Line). But in order to keep the experiments in the same conditions, we try our best to keep the hardware the same, only the CPU frequency is a little high in the new server, the other are exactly the same. XEN has been installed in

the old Supermicro server, on the new IDE hard driver. Experiments have been done in the three software environment.

- VMware 3.5/4.0: 1 VM in VMware has been set as a layer 3 router, by using the kernel forwarding path as default in Linux Operating System. Unbuntu 9.04 has been used as the operating system because of the highly updating speed. By setting the 2 virtual interfaces as gateways of the subnets, through ifconifg, we make a basic configuration of 2 nets communicating through one software router. The routing table has only 2 entries to make the 2 subnets connected. 2 IP addresses have been assigned to the virtual interfaces, belonging to different subnets. In the Agilent Router Tester, packets with 253 IP addresses in the specific subnet (from 192.1.1.2/24∼192.1.1.254 for example) has been randomly generated and transmitted to the router. Then we open other 1, 2, 3 Linux operating systems to take up the physical resource in the server respectively.

- XEN: XEN has the concept of dom0. To make a fairness compare in the virtual machines, we set 1 VM in domU in XEN to a layer 3 router, by using the kernel forwarding path as default in Linux Operating System. Unbuntu 9.04 has been used as the operating system because of the highly updating speed. The experimental conditions are exactly the same like in VMware as described. Only thing we need to concern is all the VMs are in domU. All the routing functions have been done virtually. Dom0 just acts as a monitor. This idea is different in some papers because they suggest all the routing functions should work in dom0. But in our case, running everything in virtual environment is the aim. So everything concluding forwarding, controlling and managing functions are implemented in domU.

In order to set up a bench of mark in the routing ability evaluation, a physical Linux Operating System in Ubuntu 9.04 has been deployed in the old server also. The first test is a simple one that reveals the dom0 and the physical Linux has the same routing ability more or less, shown in figure 3.10. From this we can see that only in 64 bytes packet length there is a little degradation when routing packets in the same condition, due to the impact of the patch and processes from XEN like xend. But this decrease can be ignored when running VMs on it.

The results of one VM routing ability when other VMs just opening are reported as shown in figure 3.11∼3.14.

From the results in VMware as shown in figure 3.11 and 3.13, it is very clear that when the number of the VMs opening with nothing else to do increases, the performance of the other VM acting as the software router becomes lower. This is obvious since the running VMs take up the physical server resource even though

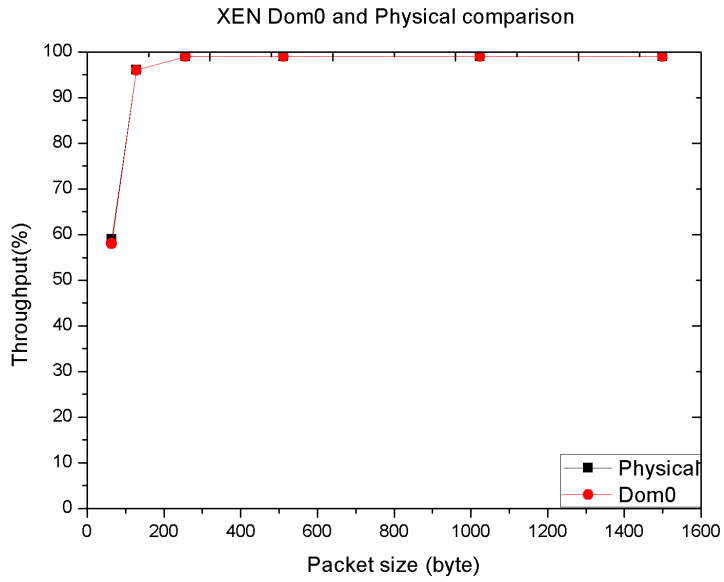

Figure 3.10: Physical Linux and dom0 in XEN when routing packets

they do nothing. But the interesting thing is the reduction of the performance due to more VMs opening is not so much compare with the reduction of the performance due to virtual software router compare to the physical software router. So it gives us the opportunity to take the advantage of this, when implementing the energy saving function in our multistage software routers. Another interesting thing is that the small packet size flow is worse than large packet size flow when the VM route packets. This is because in our tests the traffic load is from 100Mbps to 1Gbps with 100 Mbps increasment every time. The packet number is higher when small size packets are generated in Router Tester when load fixed compare to larger ones. And our software router is a standard IP packet router. The time and computation load in packet router are in header processing, so the more packets a router get at the same duration of teh time bound, the more intense the router works. In 64 byte packets, the software router is saturated when processing the huge amount of packets' header, while in 1500 byte packets, the software router shows very good performance because of a few packets' header to deal with.

Besides the common trends in the three types of experiments, which can be summarized as the performance of the virtual software router has lower routing ability when the number of VMs increase, but this reduction is not so large compare to the reduction between a physical software router and virtual software router, and the router can work better in large packet size as most of the cases do. There are

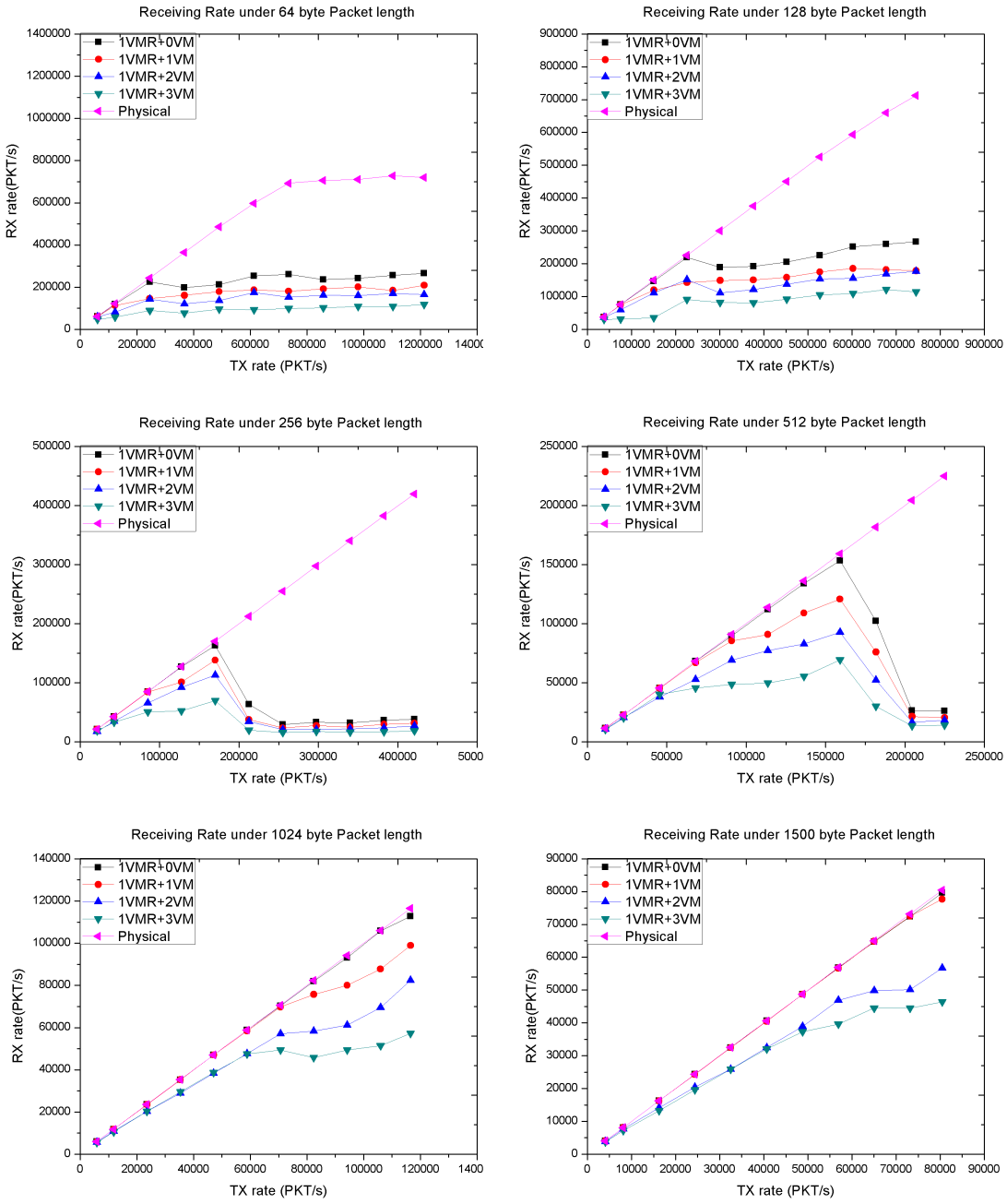

Figure 3.11: VMware 3.5: 1 VM act as software router when $0 \sim 3$ VMs just open

also some other interesting phenomenon in XEN and VMware 3.5.

In VMware 3.5, the routing ability in 256 and 512 byte packet size is very poor

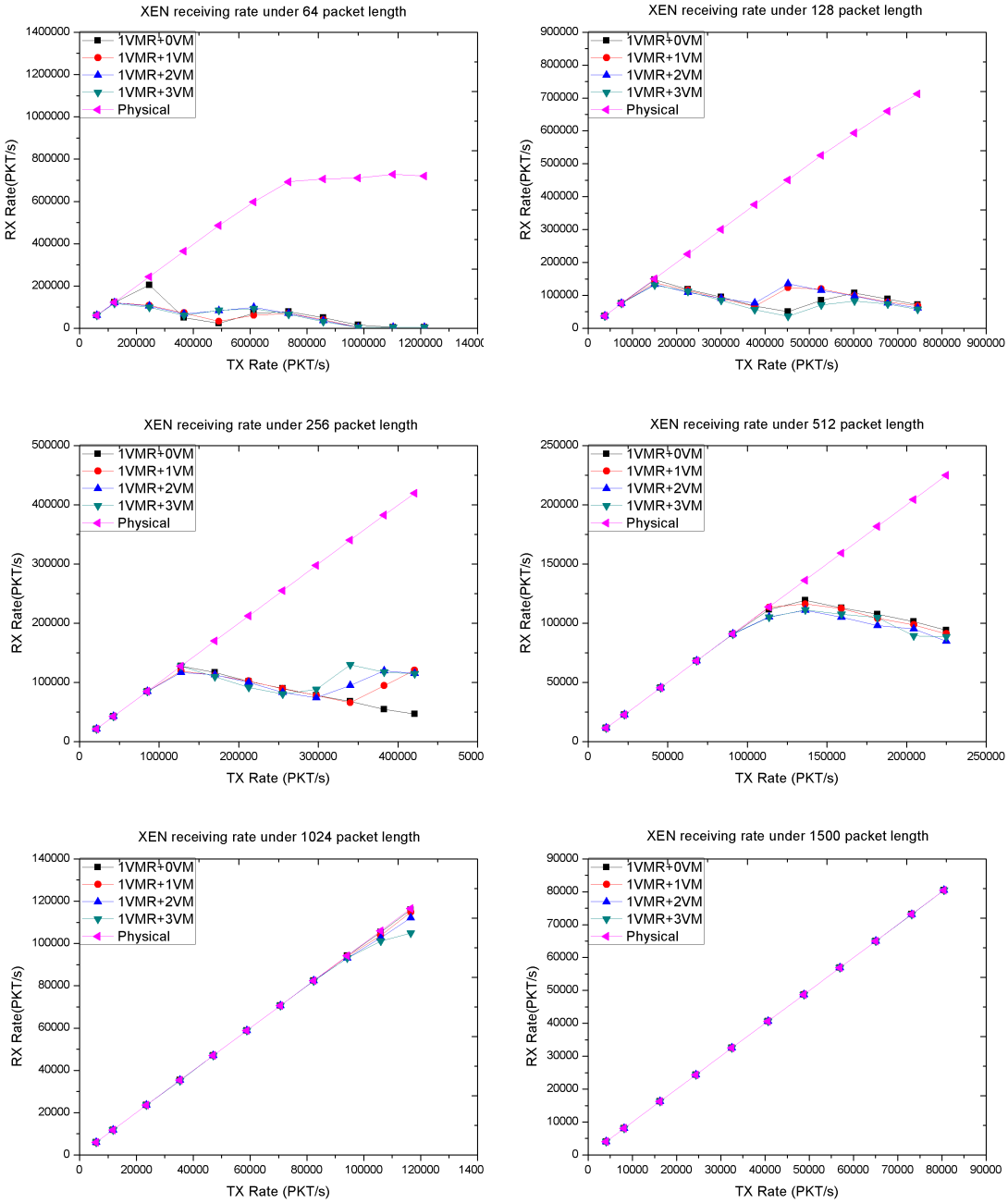

Figure 3.12: XEN: 1 VM act as software router when 0 $\sim$ 3 VMs just open

when the traffic load is high as shown in figure 3.11 and 3.14. As we decrease the traffic load injecting into the software router, this phenomenon has been mitigated

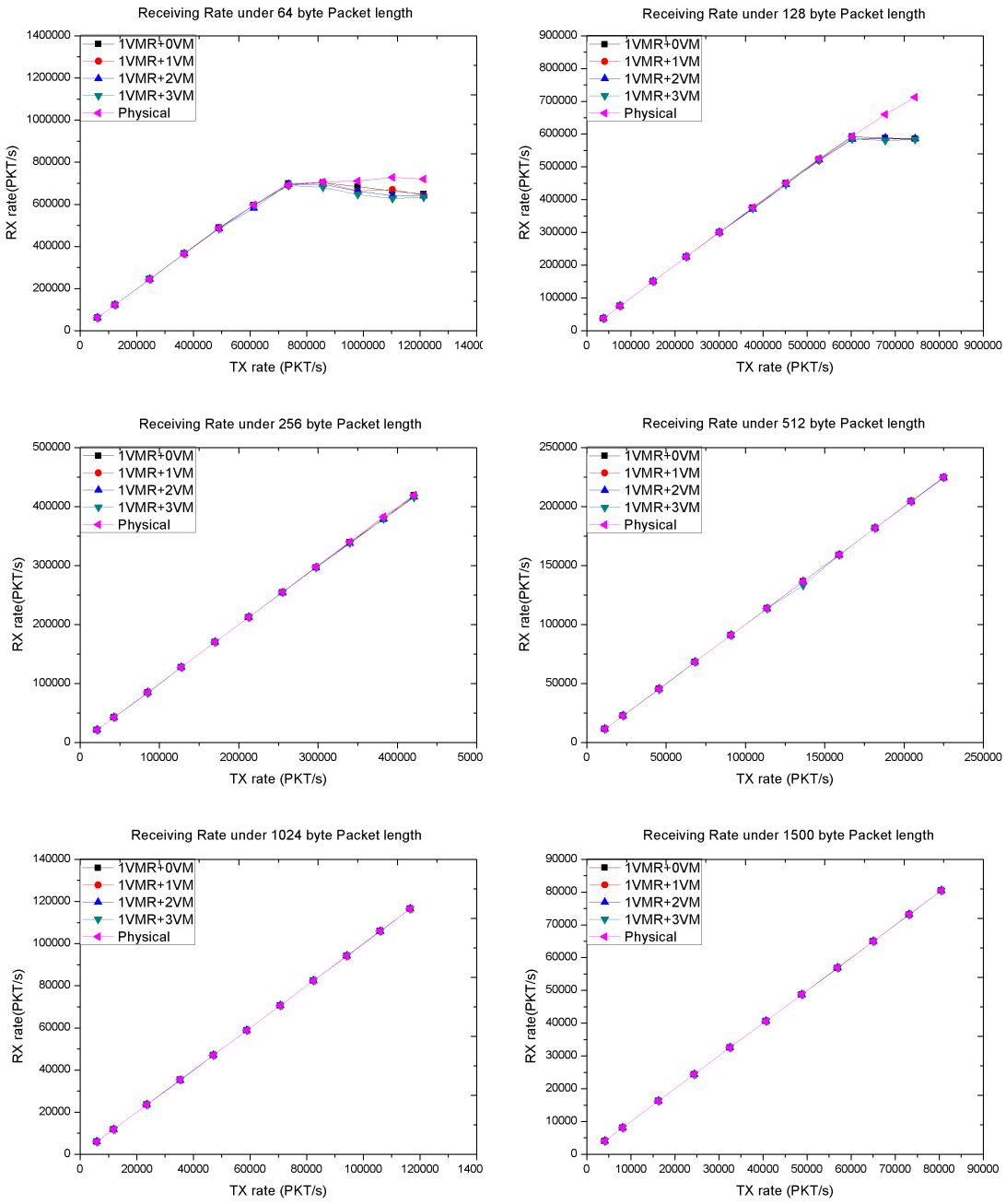

Figure 3.13: VMware 4.0: 1 VM act as software router when $0 \sim 3$ VMs just open

but still exists. Since the internal architecture is hard to reveal in a short time, the explanations about this lied on two possibilities. One is the VMware virtual interface

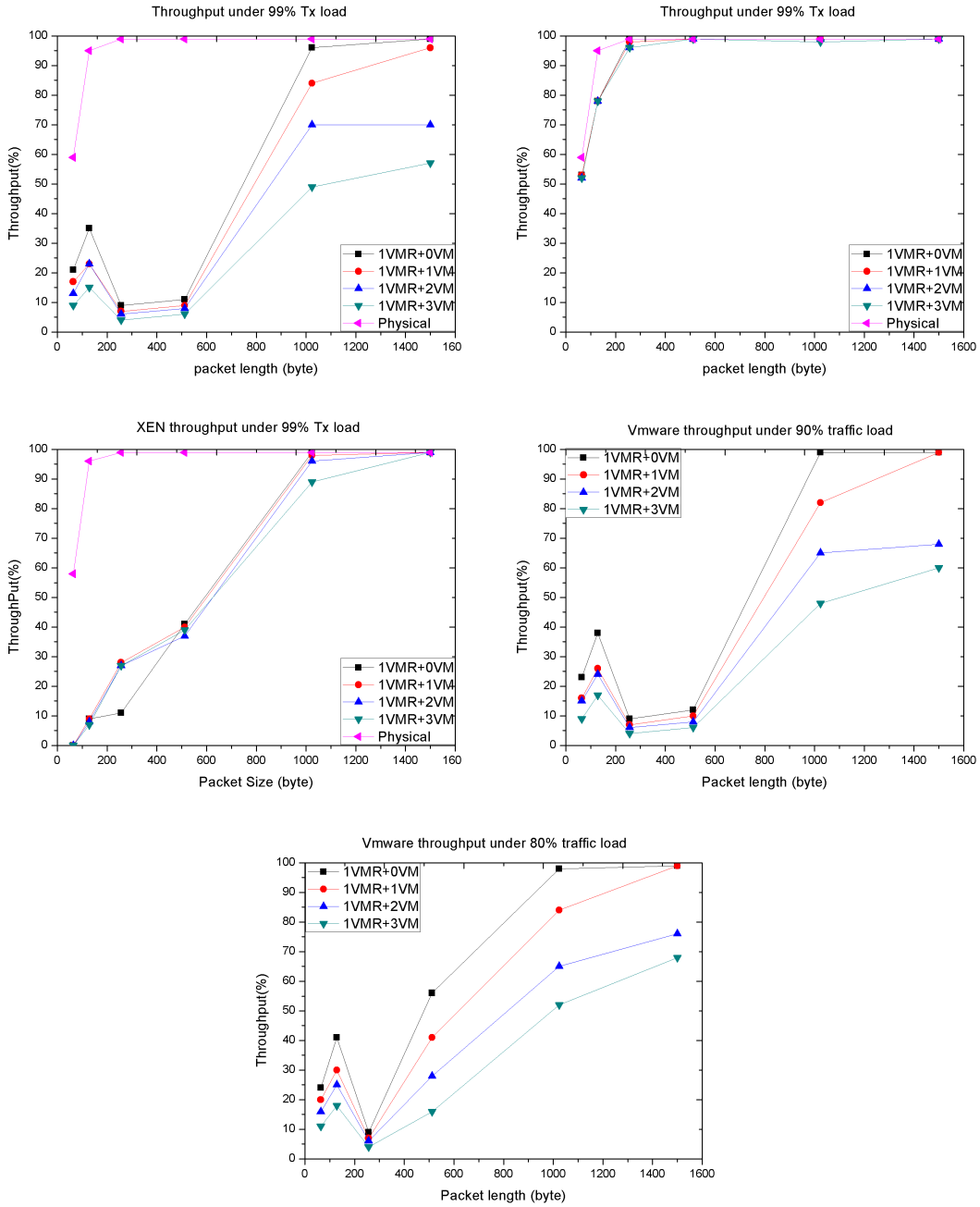

Figure 3.14: VMware 3.5(top-left), VMware 4.0(top-right) and XEN(middle-left) throughput under 99% traffic load, VMware 3.5 throughput under 90% traffic load(middle-right),VMware 3.5 throughput under 80% traffic load(bottom)

drivers, which are private of the VMware Inc. There are several options when we set the configuration, such as the traditional E1000 Intel 82545EM Gigabit Ethernet NIC, Vlance AMD 79c790 PCnet32 LANCE NIC, Flexible to let the system decide which driver to and VMXNET network adapter optimized for VMware server. Since in VMware ESXi 3.5 the best performance card is VMXNET 2 (Enhanced), so our tests are in this driver, the results can not be accepted for future use, but when things move to VMware ESXi 4.0, a new version comes up named as VMXNET 3, and from the performance shown in the figures, we can see that it is extremely good even compare to the physical server itself when routing packets. The other one is related to some PCI-X performance impairment. Indeed, in [27], the authors, using a PCI protocol analyzer, show that the bus efficiency for bursts of 256 byte or less is pretty low.

In XEN, the routing performance seems a little complex when packet size is small. There is always a wave in the curve. But this phenomenon is very normal actually. In order to explain this, a concept must be introduced first—-NAPI. The NAPI [24] was introduced in the 2.4.27 kernel version, and it has been explicitly created to increase the reception process scalability. It handles network interface requests with A interrupt moderation mechanism, which allows to adaptively switch from a classical interrupt management of the network interfaces to a polling one. In particular, this is done by inserting, during the HW IRQ routine, the identifier of the board generating the IRQ to a special list, called "poll list" and scheduling a reception SoftIRQ, and disabling the HW IRQs for that device. When the SoftIRQ is activated, the kernel polls all the devices, whose identifier is included in the poll list, and a maximum of quota packets are served per device. If the board buffer (RxRing) is emptied, then the identifier is removed from the poll list and its HW IRQs re-enabled, otherwise its HW IRQ are left disabled, the identifier kept in the poll list and a further SoftIRQ scheduled. While this mechanism behaves like a pure interrupt mechanism in presence of low ingress rate (i.e., we have more or less a HW IRQ per packet), when traffic raises, the probability to empty the RxRing, and so to re-enable HW IRQs, decreases more and more, and the NAPI starts working like a polling mechanism.

For each packet, received during the NAPI processing, a descriptor, called skbuff, is immediately allocated and used for all the layer 2 and 3 operations. A packet is elaborated in the same NET_RX SoftIRQ, till it is enqueued in an egress device buffer, called Qdisc. Each time a NET_TX SoftIRQ is activated or a new packet is enqueued, the Qdisc buffer is served. When a packet is dequeued from the Qdisc buffer, it is placed on the Tx Ring of the egress device. After the board transmits one or more packets successfully, it generates a HW IRQ, whose routine schedules a NET_TX SoftIRQ. During a NET_TX SoftIRQ, the Tx Ring is cleaned of all the descriptors of transmitted packets, that will be de-allocated, and refilled by the

packets coming from the Qdisc buffer.

When multiple VMs are running inside the server, the NAPI is active just like the physical ones. When there are too many interrupts generated, the routing performance become very low in small packet size, which suppose to have more packets' headers to process by the CPU. But after some threshold the NIC will no longer work in the interrupt mode, but change to a NAPI mode, then the throughput become higher than before. As the traffic load becomes higher, the traffic jam will block the interface, making the receiving rate decrease a little. So from the figures we can see some wave behavior. Take the 256 packet size for example, the 4 VMs case is the first one that using NAPI to process packets, and the last one is 1 VM. That's because the more VMs running in the server, the more tasks the server should take then of course it is easier to reach the interrupt threshold.

VMware 4.0 shows very good performance in the figures, the driver in VMXNET 3 has been tuned to optimal state. There is only minor decrease in 64 and 128 byte packet sizes, the others are very close to the physical server. So our multistage software router will be migrated into this scenario.

3.3.2 Multiple Virtual Machines Performed as Routers

In this part we will perform another type of tests intending to find out the aggregation of the routing throughput when multiple VMs route the packets at the same time. This time all the VMs opening inside the server will be set as software routers just like the first tests. All the VMs are deployed with the Ubuntu 9.04 Linux operating system and all of them have been active to route packets from one subnet to another, by changing the /proc/sys/net/ipv4/ip_forward value to 1 in the Linux. Then all the VMs have the ability to route packets as long as the routing table have been set correctly.

Since at the beginning of the virtual machines' routing ability seeking, we do not want to introduce too much overheads about looking up the routing table or learning new entries by using routing protocol such as OSPF or BGP, only static routing tables are used in the tests. And for simplicity, 2 subnets for each VM router are used. The topology of the tests is shown as figure 3.15.

In the Agilent Router Tester, only one module one port has been used to connect with the data port on the server, through a standard cat5e UTP cable. The VMs (R) have been configured to routing packets as shown in the figure 12, using 2 virtual interfaces inside each VM. All the virtual interfaces and the physical interface have been connected through a virtual bridge both in VMware and in XEN. All the traffics will go through the data port in the server. By shutting down redirection ability in the Linux system, we can ensure the traffic make a loop from the router tester and

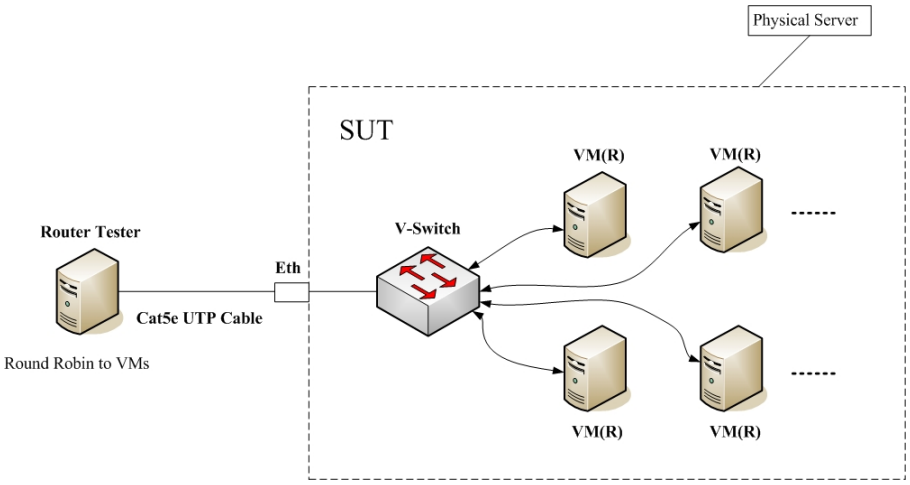

Figure 3.15: Multiple VMs act as routers

the VM routers correctly. Ifconfig and add route commands in the Linux system are used to set the IP addresses and routing tables. Besides the configuration in VM routers, the Router Tester should be set correctly, especially the next-hop MAC addresses is important.

In this experiment, all the VMs are used to route packets. We want to find out the aggregation throughput of the VM routers. This is very important since the multistage software routers will be implemented inside one physical server eventually. Multiple routing elements in the third stage and LBs in the first stage are just standard open Linux routers like in this experiment's VM routers.

The parameters used are reported as below when doing the testes:

- Packet Size : 64, 128, 256, 512, 1024, 1500 Bytes.
- Traffic Load : 50, 100, 200, 300, 400, 500, 600, 700, 800, 900, 990 Mbps using Round Robin schema to inject into different VM routers.
- Physical Link Speed : 1Gbps.
- Number of Agilent Modules/ports : 1 Module with 1 port.
- Routing Table : 2 subnets with 2 entries for each of the VM router, totally 8 entries with 8 subnets in this experiment.
- Time : 30 seconds for each packet size with each load.

VMware ESXi version 3.5 and 4.0 have been installed into 2 different servers as mentioned before, due to the compatibility issues from the VMware HCL (Help

Compatibility on-Line). But in order to keep the experiments in the same conditions, we try our best to keep the hardware the same, only the CPU frequency is a little high in the new server, the others are exactly the same. XEN has been installed in the old Supermicro server, on the new IDE hard driver. Experiments have been done in these three software environment.

- VMware 3.5/4.0: Totally 4 VMs in VMware have been set as layer 3 routers, by using the kernel forwarding path as default in Linux Operating System. Unbuntu 9.04 has been used as the operating system because of the highly updating speed. By setting the 2 virtual interfaces as gateways of the subnets in each VM router, through ifconifg, we make a basic configuration of 2 nets communicating through one software router each. The routing table has 2 entries to make the 2 subnets connected in each VM router. 2 IP addresses have been assigned to the virtual interfaces, belonging to different subnets in different VM routers. In the Agilent Router Tester, packets with 253*4 IP addresses in specific subnets have been randomly generated and transmitted to the routers, using a round robin schema. The running VM routers in the physical server have been configured incrementally from 1 VM router to 4 VM routers during the tests.

- XEN: XEN has the concept of dom0. To make a fairness compare in the virtual machines, we set 4 VMs in domU to layer 3 routers, by using the kernel forwarding path as default in Linux Operating System. Unbuntu 9.04 has been used as the operating system because of the highly updating speed. The experimental conditions are exactly the same like in VMware as described. Only thing we need to concern is all the VMs are in domU. All the routing functions have been done virtually. Dom0 just acts as a monitor. This idea is different in some papers because they suggest all the routing functions should work in dom0. But in our case, running everything in virtual environment is the aim. So everything including forwarding, controlling and managing functions are implemented in domU.

In order to set up a bench of mark in the routing ability evaluation, a physical Linux Operating System in Ubuntu 9.04 has been deployed in the Supermicro server. Through the same configuration we make it acting as a standard open router. The results of this physical routing performance have been reported in every graphs as shown in 3.16 $\sim$ 3.18.

From these results it is clear that when increasing the VMs' number, the aggregation throughput is becoming worse. This phenomenon occurs severe in small packet size rather than large ones. This is due to small packet size traffic has more packets to be processed by the kernel than the large ones if traffic load maintains

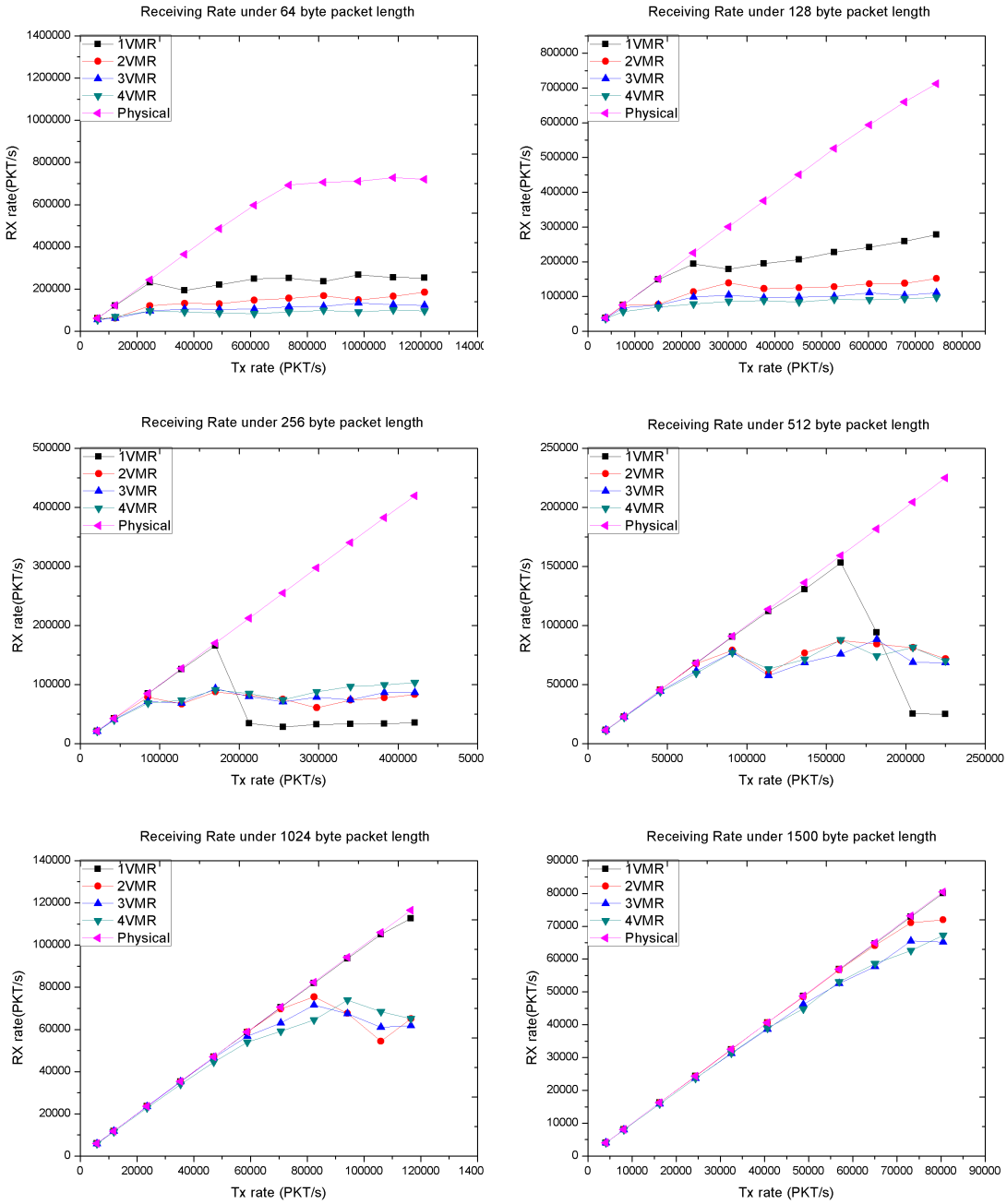

Figure 3.16: VMware 3.5: $1 \sim 4$ VMs act as software routers

the same. And more open VMs will consume more computational resource from the physical CPU and internal bus, so the performance will be decrease when increasing

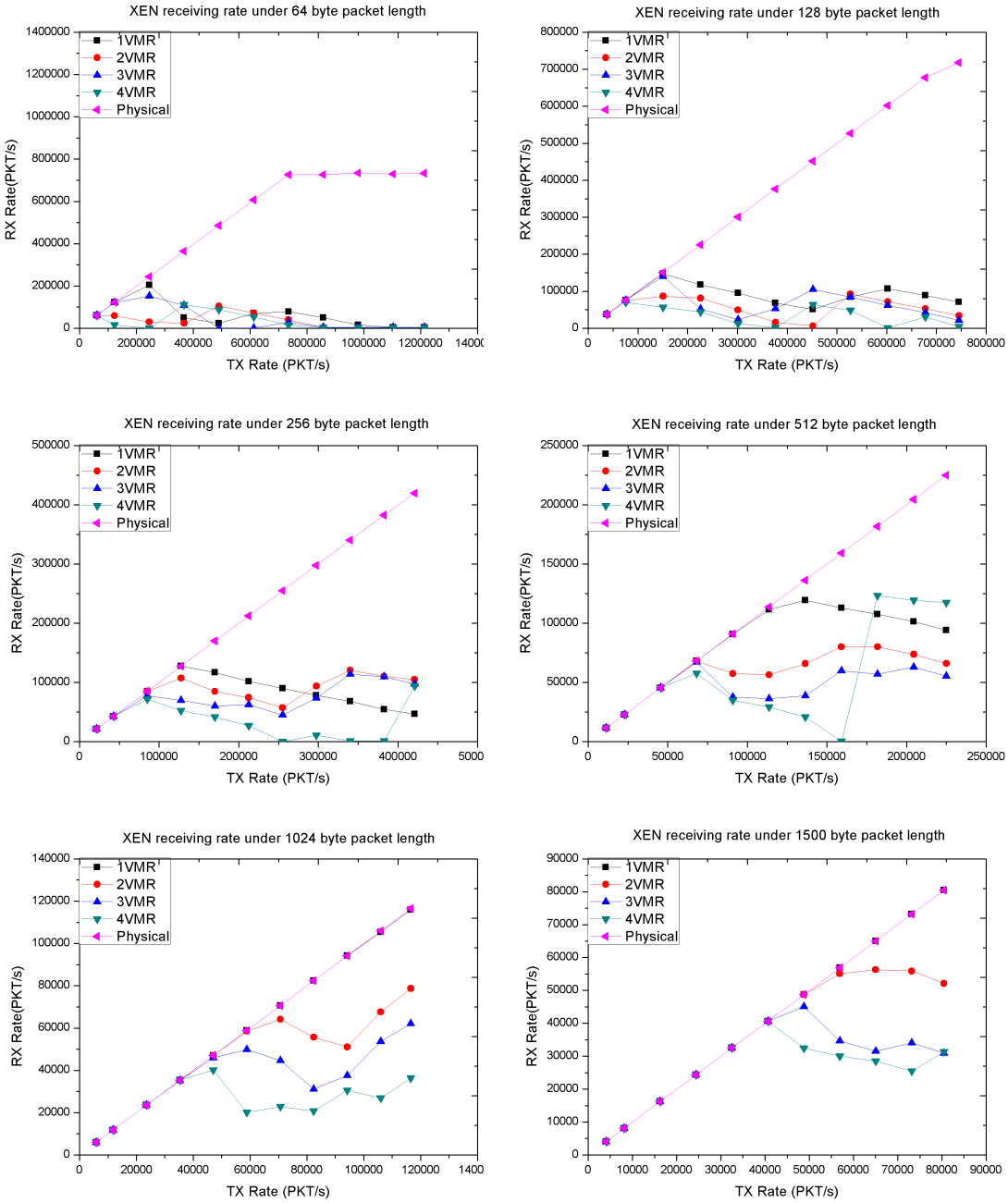

Figure 3.17: XEN: 1∼4 VMs act as software routers

the virtual systems.

If comparing the opening VMs acting as router (3.3.2) and just open some VMs

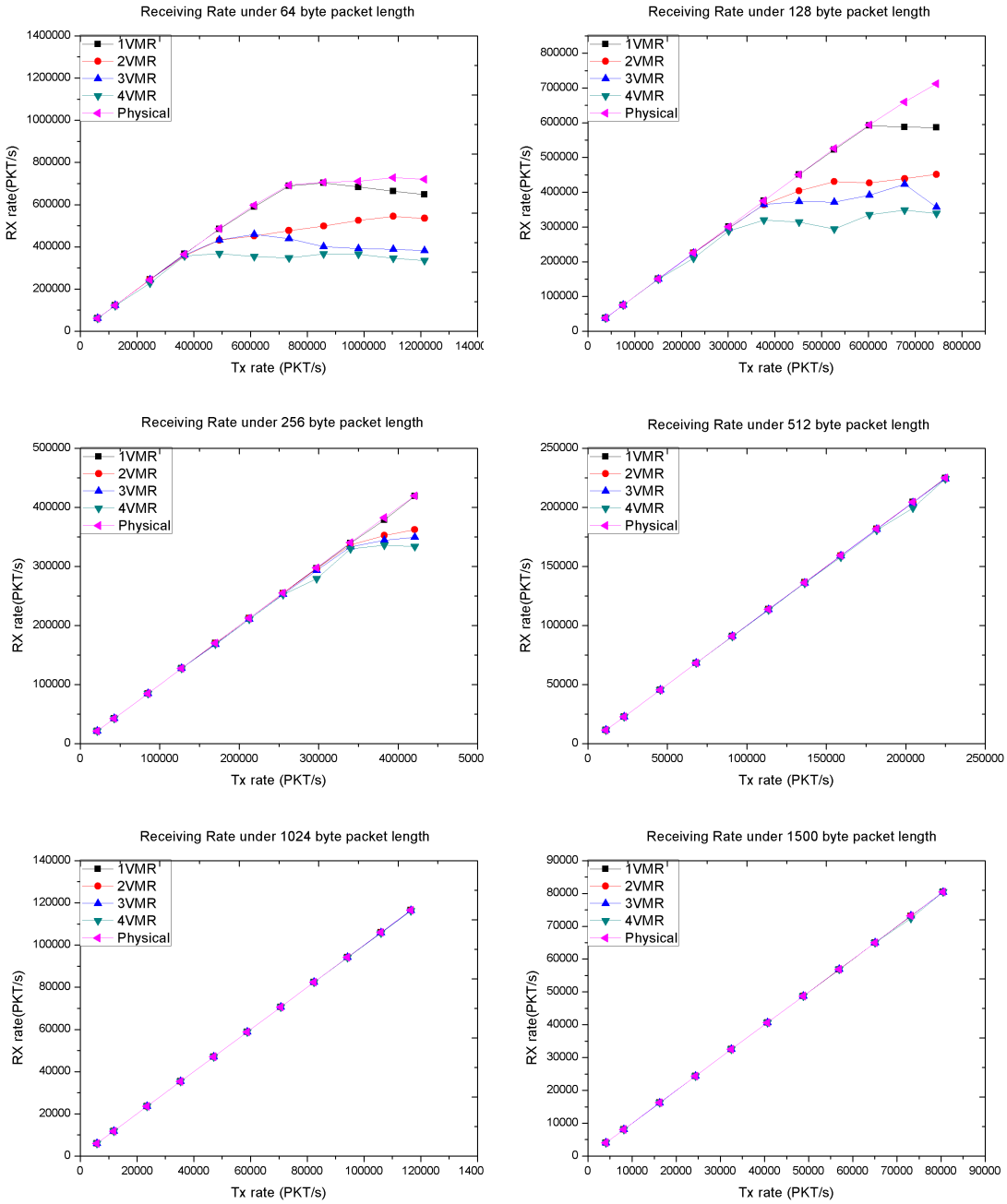

Figure 3.18: VMware 4.0: 1∼4 VMs act as software routers

but do nothing (3.3.1), like shown in the figures 3.19(right) (Here only the 256 byte packet size traffic has been plotted, the other are just the same and 256 can stand for

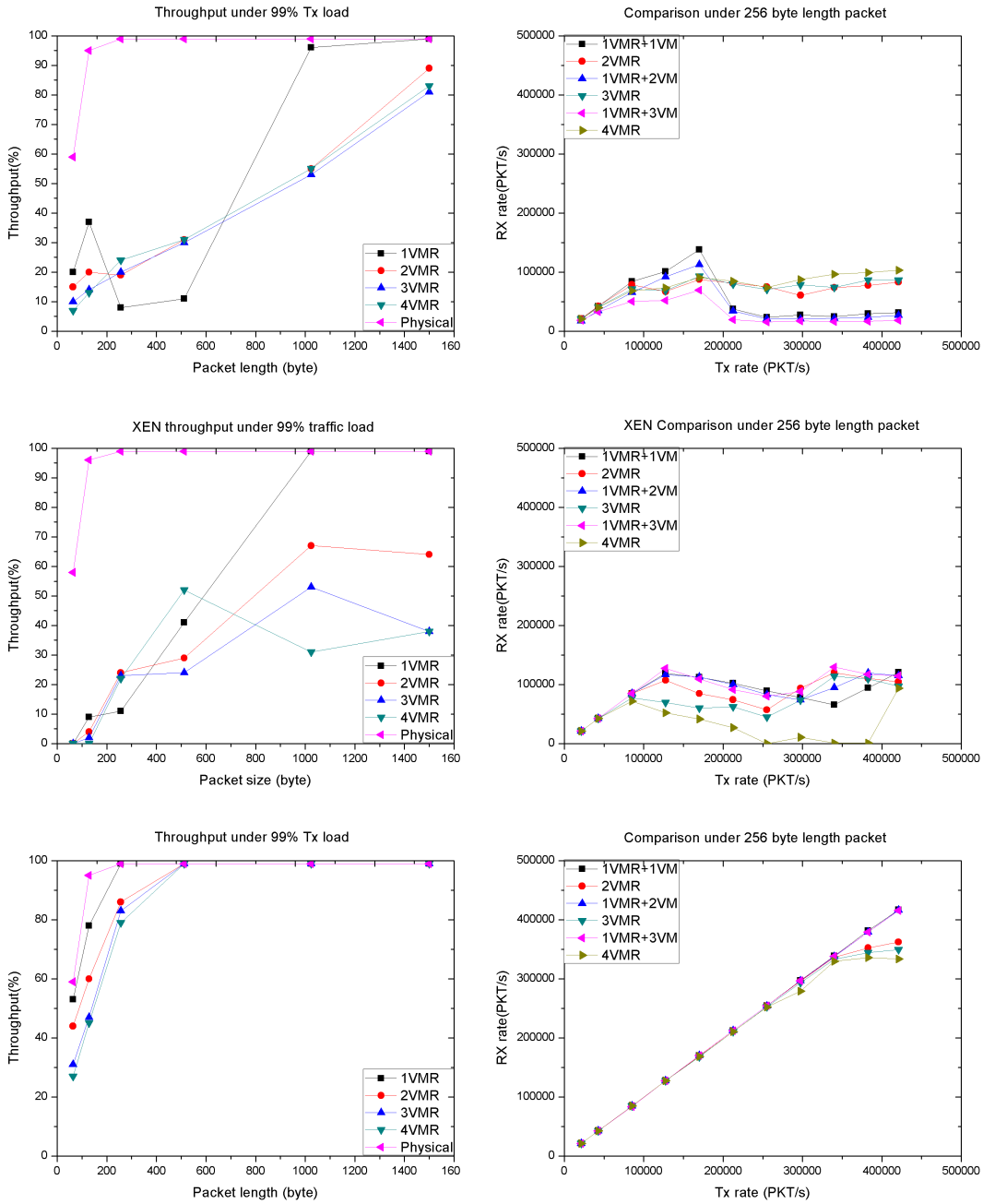

Figure 3.19: Comparison between xVMRs and 1VMRs and throughput under 99% traffic load in VMware 3.5(top), XEN(middle) and VMware 4.0(bottom)

small and large packet size), we can find out that in most the cases the aggregation throughput of multiple VM routers is worse than just active only one VM with the routing ability. This is interesting because people always think multiple VM routers perform better than single software router. That is true in physical software routers, but is opposite in virtual domain. In a physical server, the total amount of hardware resource is fixed. By configuring just one VM routing packets, all the resource will be allocated to that VM if the VMM has been designed well such as VMware 4.0, so the performance should be not so difference with physical routing ability. But as long as multiple virtual routers show up inside one physical server, the resource will be divided among them, then some context switch functions and interrupts/preemption in the CPU occurs. These are drag down the system performance definitely. But when there are only running VMs doing nothing, they need limited resource from the physical hardware and lower the communication between the VMM and VMs. These will not influence the routing ability of the software router in that server so much. From the figures, we can see exactly the same results. Multiple VM routers aggregation throughput is lower than only one VM routing packets in the same number of VM running inside the server.

Figure 3.19(left) shows the throughput when the packet size is switched from 64 bytes to 1500 bytes. Combining with the previous results, we can find out some interesting things in VMware 3.5 and XEN. In VMware 3.5, the routing performance sharply decreases in 128 and 256 bytes packets size as described in 3.3.1. But from these multiple VM routers results in VMware ESXi 3.5, we can not find the performance reduction. That's should be some coding bugs in VMM or in VMXNET 2. Most likely there is no NAPI enabled when only one VM acting as router. In XEN, it is easy to explain the reason why there are a lot of waves. When the physical server does not intensive for the interrupts, the NIC is working in a interrupt mode. But when the traffic load becomes high, too many interrupts have been generated. This will make the NIC working in the NAPI mode and improve the performance. The threshold of the interrupts is different from the VM routers. As many VM routers cause a lot of switch functions inside the server, this will make the server more critical and let the server's NIC enter the NAPI mode easily. While the few VM routers will ease the totally amount of interrupts and make the NIC slower to enter the NAPI mode. That is why we find some strange behavior in XEN. At the beginning, the more VM routers are, the lower performance of the aggregation throughput. In the middle of these figures, the 4 VM routers' curve is always decrease to the threshold and then become performing better than the others. After the traffic load increase more, all the curves will be working in the NAPI mode, and 4 VM routers' curve drops again to the lowest one. Also this phenomenon is vivid in small packet size. In the large packet size, since the total amount of packets are not so heavy, the NICs will not working in the NAPI mode. So the curves are more

consistent, especially in 1024 and 1500 bytes packet size.

This time VMware 4.0 shows better performance again due to the optimizing the NIC drivers in VMXNET 3. But we can see that when things are in virtual domain, multiple VMs routing packets at the same time will decrease the routing ability a lot even in the simple condition. When routing algorithms have been used, the computation time involved will make the performance worse as we can imagine. For now, VMware 4.0 shows the best performance, so our multistage software router will be migrated into this scenario in next chapter.

3.3.3 Fairness Test

In this part the fairness test will be done in VMware ESXi 3.5 and XEN. Since the better performance will not influence the fairness issues among VMs, we skip the tests in VMware 4.0. This time all the VMs opening inside the server will be set as software routers just like the previous tests. All the VMs are deployed with the Ubuntu 9.04 Linux operating system and all of them have been active to route packets from one subnet to another, by changing the /proc/sys/net/ipv4/ip_forward value to 1 in the Linux. Then all the VMs have the ability to route packets as long as the routing table have been set correctly.

Since at the beginning of the virtual machines' routing ability seeking, we do not want to introduce too many overheads about looking up the routing table or learning new entries by using routing protocol such as OSPF or BGP, only static routing tables are used in the tests. And for simplicity, 2 subnets for each VM router are used. The topology of the tests is the same with the aggregation throughput tests shown as figure 13.

In the Agilent Router Tester, only one module one port has been used to connect with the data port on the server, through a standard cat5e UTP cable. The VMs (R) have been configured to routing packets as shown in the figure 13, using 2 virtual interfaces inside each VM. All the virtual interfaces and the physical interface have been connected through a virtual bridge both in VMware and in XEN. All the traffics will go through the data port in the server. By shutting down redirection ability in the Linux system, we can ensure the traffic make a loop from the router tester and the VM routers correctly. Ifconfig and add route commands in the Linux system are used to set the IP addresses and routing tables. Besides the configuration in VM routers, the Router Tester should be set correctly, especially the next-hop MAC addresses is important. Since we need to capture the traffic flows from different VM router, in the Router Tester we need to generate multiple traffic flows manually and then inject them into different virtual routers correctly.

In this experiment, all the VMs are used to route packets. We want to find out

the fairness issue among the VM routers. This is very important since the multistage software routers will be implemented inside one physical server eventually. Multiple routing elements in the third stage and LBs in the first stage are just standard open Linux routers like in this experiment's VM routers. Fair or not will be very critical when we slicing our multistage software routers to different users.

The parameters used are reported as below when doing the testes:

- Packet Size : 64,1500 Bytes.
- Traffic Load : 50, 100, 200, 300, 400, 500, 600, 700, 800, 900, 990 Mbps but with different proportion among the VM routers as shown in the figures, using Round Robin schema to inject into different VM routers.
- Physical Link Speed : 1Gbps.
- Number of Agilent Modules/ports : 1 Module with 1 port.
- Routing Table : 2 subnets with 2 entries for each of the VM router, totally 8 entries with 8 subnets in this experiment
- Time : 30 seconds for each packet size with each load.

VMware ESXi version 3.5 has been installed into the Supermicro server on a SATA hard driver while XEN has been installed into the same server with a new IDE hard driver. Experiments have been done in these two software environment.

- VMware 3.5: Totally 4 VMs in VMware have been set as layer 3 routers, by using the kernel forwarding path as default in Linux Operating System. Unbuntu 9.04 has been used as the operating system because of the highly updating speed. By setting the 2 virtual interfaces as gateways of the subnets in each VM router, through ifconifg, we make a basic configuration of 2 nets communicating through one software router each. The routing table has 2 entries to make the 2 subnets connected in each VM router. 2 IP addresses have been assigned to the virtual interfaces, belonging to different subnets in different VM routers. In the Agilent Router Tester, packets with 253*4 IP addresses in specific subnets have been randomly generated and transmitted to the routers, using a round robin schema. The running VM routers in the physical server have been configured incrementally from 2 VM routers to 4 VM routers during the tests. The traffic has been captured in the Router Tester based on different flows to the VM routers.
- XEN: XEN has the concept of dom0. To make a fairness test in the virtual machines, we set 4 VMs in domU to layer 3 routers, by using the kernel

forwarding path as default in Linux Operating System. Unbuntu 9.04 has been used as the operating system because of the highly updating speed. The experimental conditions are exactly the same like in VMware as described. Only thing we need to concern is all the VMs are in domU. All the routing functions have been done virtually. Dom0 just acts as a monitor. This idea is different in some papers because they suggest all the routing functions should work in dom0. But in our case, running everything in virtual environment is the aim. So everything including forwarding, controlling and managing functions are implemented in domU.

When only 2 VM routers are running, the traffic between them has been injected in three different ways: half-half, 10%-90%, 30%-70% of the total amount of load. This can reveal the fairness issues based on the traffic fluctuation. While in 3 and 4 VM routers, there are only 1/3-1/3-1/3 and 1/4-1/4-1/4-1/4 traffic shape, a uniform input traffic matrix actually. This can reveal the fairness issues among more than 2 VM routers in the same server.

The fairness tests show different results in XEN and VMware. In VMware the flows among different VM routers are more or less the same when combing back to the Router Tester when uniform input traffic matrix is used. From the figures there is approximately 10% 15% difference in every case. This result can be considered as fair when multiple VM routers are running inside the same physical server under VMware. And when the traffic load is not fair between the VM routers like in figure 3.20, the routing performance is also unfair between them. Always the big flow VM router get more resource from the physical server and show better performance than small flow. That's because in VMware more intensive task VM get more calls from the higher level and preempt lot of hardware resource than small flow. Also another thing is that small packet size flow suffers from the un-fair phenomenon more compare to large packet size flow because of more packets' headers to process in the kernel.

In XEN the fairness tests result in some interesting behavior. There is always some "master" flow which gets nearly all the resource from the under layer hardware. The more intensive the server works at high input traffic load, the more severe this phenomenon comes up. And this is very stable as soon as the "master" VM router stands out from the others. After several tests aimed at the master choosing issue, we find that the master virtual router comes up with a simple rule, which can be concluded as the first VM running in the XEN will always get most of the routing computational resource from the hardware. The following VMs are just get the resource when there are spare ones. This is strange but appears in every fairness test with heavy traffic load, especially in small packet size. From figure 3.22 and 3.23 we can see that even in large packet size flow with high traffic load, this behavior

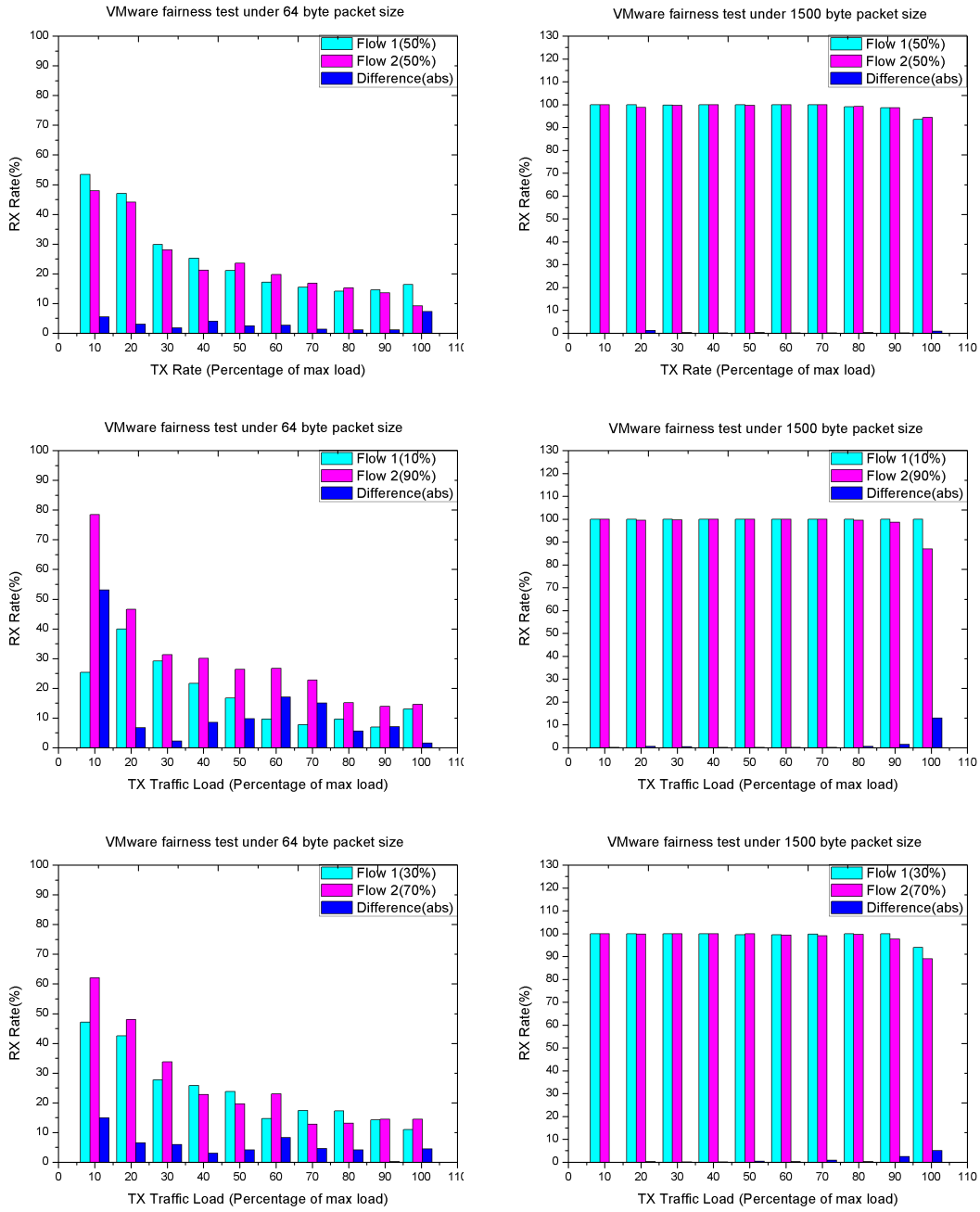

Figure 3.20: VMware fairness tests-1

shows up too. So from this fairness tests in XEN, we can see that the performance is worse than VMware and nearly no fairness among the flows when heavy traffic

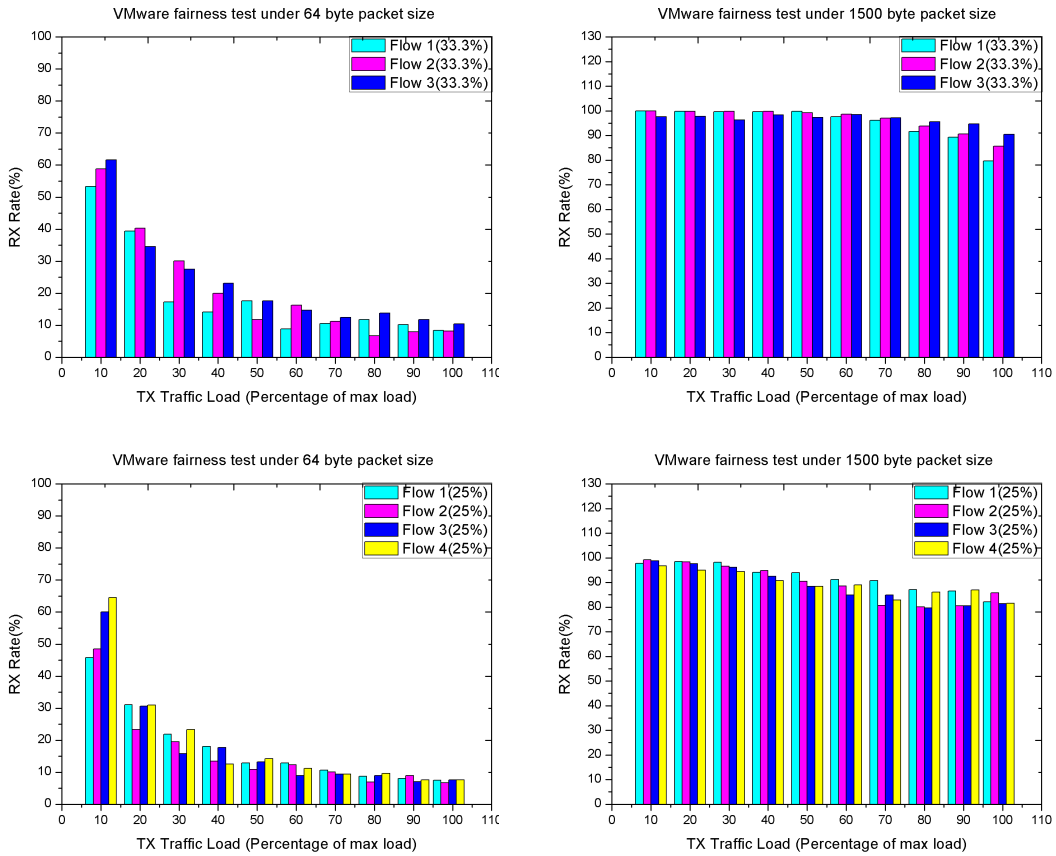

Figure 3.21: VMware fairness tests-2

load is used.

All of these routing performance tests show that in VMware ESXi 4.0, the aggregation throughput of multiple VMs in a single physical server is higher than VMware 3.5 and XEN, and the flows through different VM routers are fair enough to sustain further usage such as slicing. The influence from the opening VMs but consuming a few hardware resources is negligible. This will give us the opportunity of turning off some virtual routers during the night, which can be considered as low traffic period.

So from this moment on, all the work will be done in VMware 4.0 since the performance is better. Next chapter tries to implement the multistage software routers in the virtual server, along with some test results about every element in the architecture.

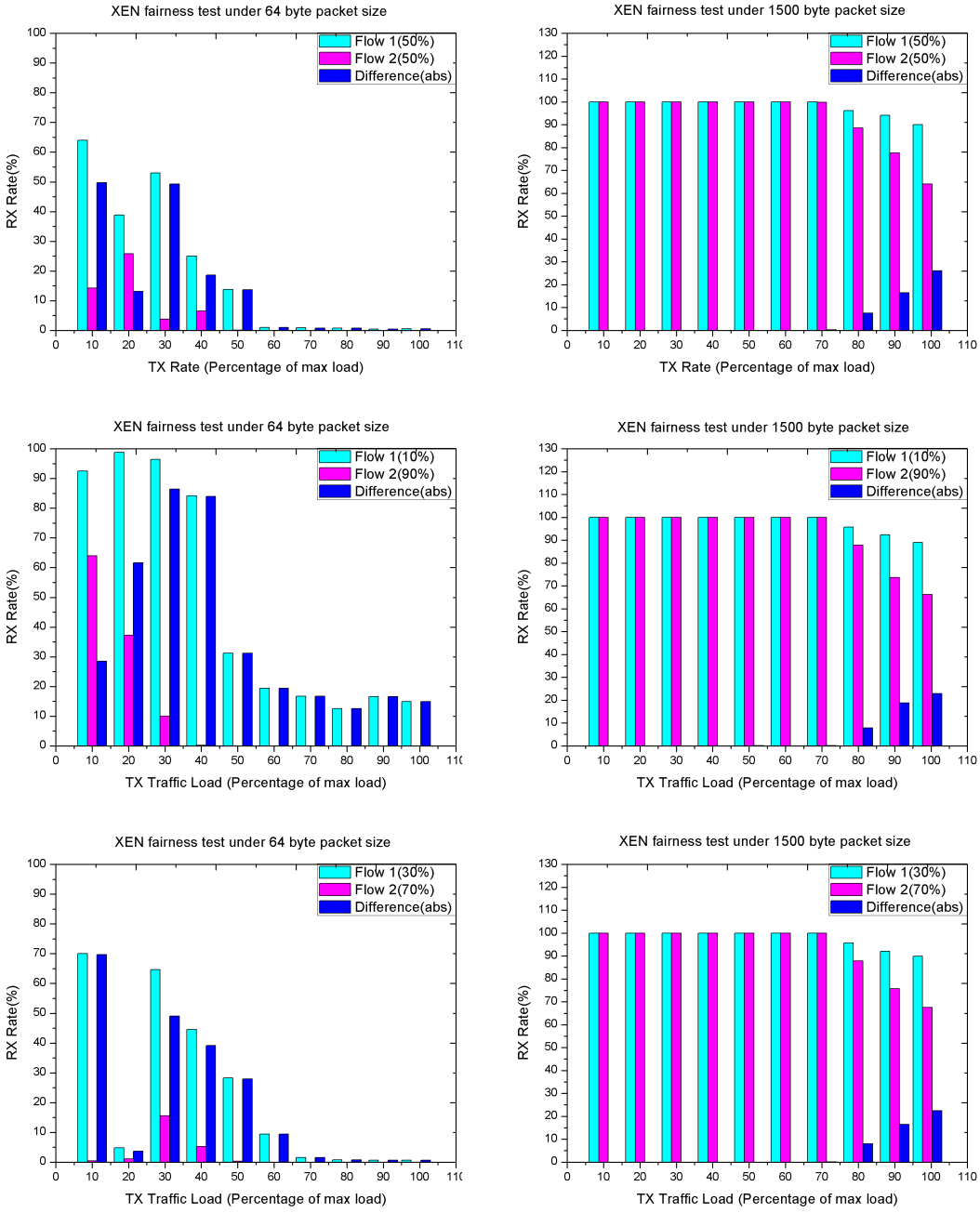

Figure 3.22: XEN fairness tests-1

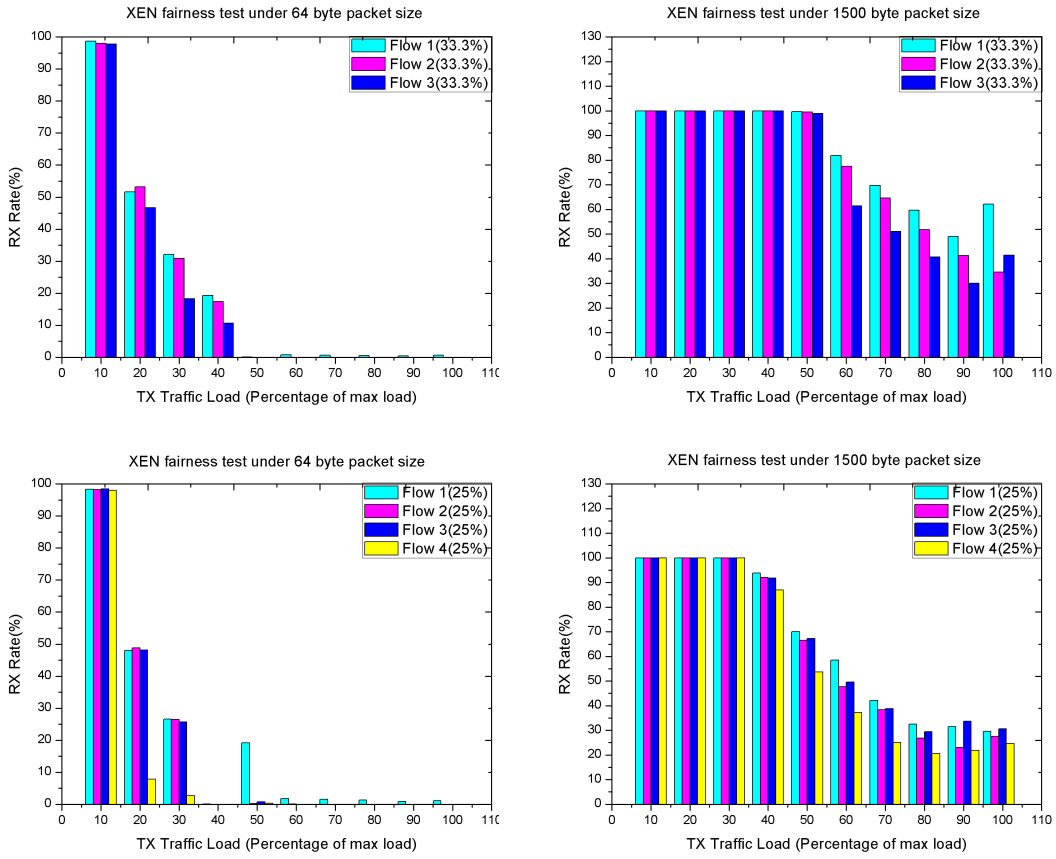

Figure 3.23: XEN fairness tests-2

Chapter 4

Multistage Software Routers Architecture Migrate

From the routing ability tests in chapter 3, we find a feasible solution in virtual environment when routing standard IP packets by using Linux operating system. VMware ESXi 4.0, a full virtualization technique, free of charge and with outstanding graphic user interface is the best choice for now. So in this chapter the physical multistage software routers will be implemented inside VMware ESXi 4.0.

The front end load balancer and the back end routing elements will be introduced detailedly, along with some tests to show the overhead of the other software used to accomplish the functions mentioned before. Then a prototype of the full architecture will be implemented with 2 load balancers and 2 routing elements in the back-end.

4.1 Front-end Load Balancer in Click

In the front-end PCs, load balancing functions have been done based on the MAC addresses modification. In the first version a simple Round Robin schema has been used to make a uniform distribution to the back-end router elements. In the future more sophysicated algorithm will be implemented. Since CLICK has been designed into modules that easy to be reuse for the MAC modification task, we deploy this software into our front-end PCs.

Click [25] is a new software architecture for building flexible and configurable routers. It was developed for Linux platform at MIT originally. Then a lot of organizations such as Mazu Networks, ICIR extended the functionality of Click, make it well-known to active router architecture developers, researchers and students.

A Click router is assembled from packet processing modules called elements. Individual elements implement simple router functions like packet classification, queuing, scheduling, and interfacing with network devices. Each element class corresponds to a subclass of the C++ class element. Users can import their own element class as needed. Our usage of the Click architecture is based on this idea. We implement our own scripts and new elements based on the load balancing functions, basically by changing the MAC addresses of the IP packets when it come into the multistage software routers. In the internal architecture, it is a layer 2 switch, so by tricking the MAC all the packets will go to the correct back-end routing elements. After the routing functions have been done, the packets come back to the LB and the MAC has been changed once more to hide the internal architecture.

There are currently two drivers that can run Click router configurations, a Linux in-kernel driver and a user-level driver that communicates with the network using Berkeley packet filters or a similar packet socket mechanism. The user-level driver is most useful for profiling and debugging, while the in-kernel driver is good for production work. In our work, the in-kernel level has been chosen to run Click with our configuration to improve the performance.

The Linux kernel module requires a patched Linux in 2.2, 2.4, or 2.6 system. The latest version of Click is 1.7.0rc1, which is released in 03/07/09. After extracting all the files, we discover the latest patch for Linux kernel is 2.6.24-7, so this one has been used. Although all the routing ability tests are proceeding in Ubuntu 9.04 with kernel 2.6.24-11, there should be no change in the routing codes in these two kernels.

By downloading the kernel from www.kernel.org, a patch should be installed throgh patch -p1 command. Then we need to install libncurses-dev package in order to run menuconfig properly. After installing the libncurses-dev I use make menuconfig in the LINUXSRCDIR to configure the new kernel. After some simple configurations we get a new .config file for this 2.6.24-7 Linux kernel. Here we can also use the existing configuration file from the running kernel. Just copy it and make little modification. The results should be the same.

After all the previous works have been done correctly, the following things are just compiling the booted files. We use make bzImage and make modules to get the kernel files and modules files of that kernel respectively. Then make install and make modules_install are needed to put all the files into the /boot directory and /lib/modules/2.6.24.7/ directory respectively. The config, system.map, vmlinuz will be on the /boot directory, they are not enough to boot a new kernel. We need to run mkinitramfs -o /boot/initrd.img-2.6.24.7 to get the initrd.img file needed to boot the new kernel. Here I met a strange problem, after writing the menu.lst of the boot file, I choose the new kernel but it can not work. The error is something like

the system can not mount the modules of the new kernel. So here all we need to do is update our initrd.img by typing the command update-initramfs -c -k 2.6.24.7, everything should be fine now. We can boot the new kernel and install the standard Click module.

Last thing is running ./configure -with-linux=LINUXDIR in the Click source directory and use make install to finish all the procedure of installing the Click module into our patched kernel. My college is developing the specific LB functions' script in CLICK. We can simply by using click-install ROUTERCONFIGFILE in the system-level CLICK operating system to active the load balancing functions in the front end PCs.

Click can be used only in 1 processor PC, with dual-core CPU, when the traffic load is high, it will crash. So in every virtual machines only 1 CPU has been used during the tests.

4.2 Back-end Routing Array in Xorp

About the back-end routing elements, either XORP or Quagga can be used since we have already been implemented the DIST protocol into both of them. In this thesis, the Xorp has been used along with DIST protocol.

XORP [26] is the abbreviation of the eXtensible Open Router Platform, which provides a full featured platform that implements IPv4 and IPv6 routing protocols and a unified platform to configure them. It is the only open source platform to offer integrated multicast capability. XORP's modular architecture allows rapid introduction of new protocols, features and functionality, including support for custom hardware and software forwarding. In our multistage software router project, we need to extend this platform to sustain our back end forwarding engines' communication, making them act as a "single" router core to the administrator. This new extended protocol is named as DIST.

DIST is a protocol designed and developed in our lab to implement the control and management plane requirements in the multi-stage router architecture. The protocol is managed by demons running on every FE, together with traditional routing software. Each demon cooperates strictly with the routing software working as a plugin. The current implementation supports both Xorp and Quagga. The main functionalities of the DIST comprise route distribution, lost packets detect, automatic configuration, fault recovery and some security features.

The DIST protocol is aimed to make the back-end routing elements work as a single routing element just like the commercial router. So keeping all the routing tables identical is the main task. In order to enhance the security ability, we do not

use a specific PC to do this, but by choose a master node among all the back-end routing elements and make the others the slaves. In Xorp, it process all the packets based on the routing table and learned new entries as the routing protocol works on. The packets are routed through the default Linux routing kernel path to transfer. This is very important since all the routing seeking tests are performed under the Linux kernel path. But CLICK changes to its owe routing path and from the results below we find it is poor.

Every process in XORP, as well as DIST, implements a set of interfaces. Interfaces are described into .xif file, meanwhile processes are described into .tgt files, since they are named as TARGETS and they may implement more than one interface. Interfaces contain the definition of methods that a process exposes into the internal communication bus and so the functionality that it offers to the other processes, like CORBA objects. We have defined a specific interface for DIST (into polidist.xif) and then a template for the target (polidist.tgt). We have to move these files into the source tree of XORP (under xrl/interfaces and xrl/targets) and then compile it using clnt-gen and tgt-gen (two scripts that are under ./xrl/scripts/). Then we have to copy the folder polidist into the main directory of XORP's sources. Makefiles are auto-generated, so we need to modify Makefile's templates that are used to generate Makefiles. That is Makefile.am. There is a Makefile.am in every sub-directory, but we only have to modify Makefile.am into xrl/interfaces, xrl/targets and into the main directory.

Main directory:

```
#Applications (directory ordering should not matter)
SUBDIRS += bgp fib2mrib mld6igmp ospf pim rip rtrmgr static_routes vrrp polidist
```

Interface directory:

```
#Implement demone of DIST in XORP
noinst_LTLIBRARIES += libpolidistxif.la
libpolidistxif_la_SOURCES = polidist_xif.hh polidist_xif.cc
```

Target directory:

```
tgt_files                 += polidist.tgt
noinst_LTLIBRARIES   += libpolidistbase.la
#implement demone DIST into XORP
libpolidistbase_la_SOURCES = polidist_base.hh polidist_base.cc
$(srcdir)/polidist_base.hh $(srcdir)/polidist_base.cc:          \
                $(INTERFACES_DIR)/common.xif                    \
                $(INTERFACES_DIR)/redist4.xif                   \
                $(INTERFACES_DIR)/socket4_user.xif              \
                $(INTERFACES_DIR)/rib_client.xif                \
                $(INTERFACES_DIR)/polidist.xif                  \
```

After all of these have been done we have to re-generate every source's directory by using the script "bootstrap.sh" from the main directory. Here we have a very important thing to notify, the auto-tools must be used the specific versions, not the Ubuntu 9.04 native one, it's too new and make some compatible problems. If everything is OK, we can use ./configure and make to build the program into the source directory. It's not important to install polidist into the system (at least during the development phase), since we can run it directly from that directory (if the main XORP process, xorp_rtrmgr is already active) without problems.

There are two scripts in XORP that are needed to run the polidist. One is xorp_rtrmgr to active the XORP functionality in Linux. In this one we need to write our own configuration files to set the IP addresses of the interfaces, fill the routing tables and choose the routing protocol. Another one is xorp_polidist to active the DIST protocol in the routing elements, no need to set anything, the internal communication traffic can make all the elements working together as one router.

4.3 Experimental Setup and Performance Evaluation

In this part, the tests will be carried out at three steps. The first one is to find out the front-end LB performance in CLICK. The second one aimed to reveal the overhead introduced by the routing software like XORP compare to bare Linux only. The last part is some final architectures of the multistage software router. In this chapter all the tests are done in VMware 4.0 only, since the better performance than the others.

4.3.1 Click in Virtual Linux

In this category of tests, totally 2 VMs are installed with CLICK in Ubuntu 9.04, which have been set as layer 3 packets routers by changing the /proc/sys/net/ipv4/ip_forward value to 1 in the Linux. Besides this configuration, we need to active the CLICK in the kernel, using different scripts. Then we can say that the VMs are layer 3 routers since the IP routing table should be set correctly as well as the addresses, in order to transmit packets based on IP headers. And there are also some layer 2 switches' concepts because all the MAC addresses needed to be indicated explicitly from interface to interface.

The intensive experiments have been done in the previous chapter, so in this part

we have only 2 LBs maximum for getting some idea of Click routing ability. In every LB VM, there are 2 interfaces active, belonging to 2 different subnets. Besides the routing table setup using add route in the Linux kernel, we also have to tell Click that interface 1 traffic goes into interface 2 and vice versa, by setting the correct MAC addresses of the interfaces. The topology is shown in figure 4.1.

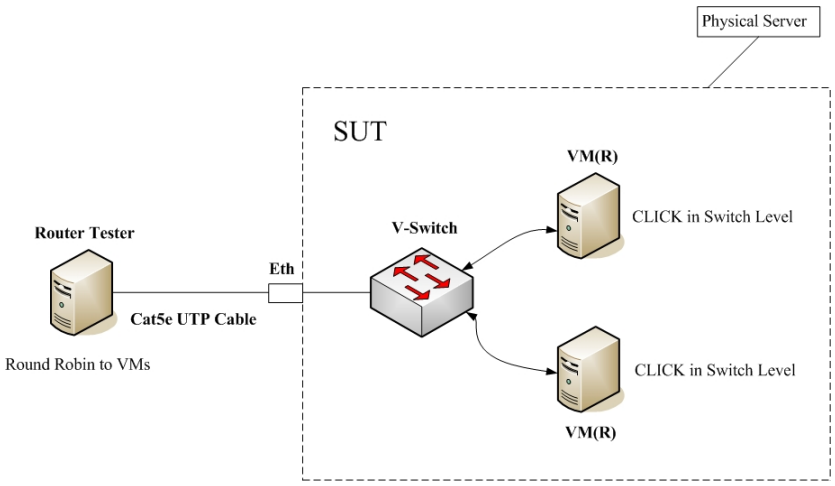

Figure 4.1: Click routing path performance test topology

In the Agilent Router Tester, only one module one port has been used to connect with the data port on the server, through a standard cat5e UTP cable. All the virtual interfaces inside the VM (R) and the physical interface have been connected through a virtual bridge in VMware. All the traffics will go through the data port in the server. By shutting down redirection ability in the Linux system, we can ensure the traffic make a loop from the router tester and the VM routers correctly. Besides the configuration in VM routers, the Router Tester should be set correctly, especially the next-hop MAC addresses is important.

In this experiment, all the VMs are used to route packets with Click routing path. We want to find out some basic idea of Click running inside virtual machines when routing standard IP packets.

The parameters used are reported as below when doing the testes:

- Packet Size : 64, 128, 256, 512, 1024, 1500 Bytes.
- Traffic Load : 50, 100, 200, 300, 400, 500, 600, 700, 800, 900, 990 Mbps , using Round Robin schema to inject into different VM routers.
- Physical Link Speed : 1Gbps.

- Number of Agilent Modules/ports : 1 Module with 1 port.
- Routing Table : 2 subnets with 2 entries for each of the VM router, totally 4 entries with 4 subnets in this experiment.
- Time : 30 seconds for each packet size with each load.

VMware ESXi version 4.0 has been installed into the Dell server on a SATA hard driver. Totally 2VMs in VMware have been set as layer 3 routers along with Click enabled, by using the Click forwarding path to route packets. Unbuntu 9.04 has been used as the operating system because of the highly updating speed. By setting the 2 virtual interfaces as gateways of the subnets in each VM router, through ifconifg, we make a basic configuration of 2 nets communicating through one software router each. The routing table has 2 entries to make the 2 subnets connected in each VM router. 2 IP addresses have been assigned to the virtual interfaces, belonging to different subnets in different VM routers. In the Agilent Router Tester, packets with 253*2 IP addresses in specific subnets have been randomly generated and transmitted to the routers, using a round robin schema. The running VM routers in the physical server have been configured incrementally from 1 VM router to 2 VM routers during the tests. The traffic has been captured in the Router Tester based on the aggregation throughput. Results have been shown in figure 4.2 and 4.3(The physical routing performance has also been reported in the graph to make a compare)

From the results we get, it is clearly to see that the Click routing path has a dramatic reduction in the routing aggregation throughput. Recall that in chapter 3 VMware 4.0, the virtual routing ability is nearly the same as physical ones, with some decrease but can be accepted. But from this test, the performance in small packet size is very poor. And by observing the graph carefully, we can find out that 2 VMs in Click's curve shows better performance when we concentrated on the aggregation throughput. This is a little strange and at this moment, we think the reason for explaining the phenomenon is lied on the internal coding about the NIC drivers in VMXNET 3, which has shown great when working in Linux kernel path. But as things go to the other forwarding path such as Click, it suffers great reduction when routing the packets. Remember in VMware 3.5, the routing ability is poor as long as only one VM is routing packets. This is just like in VMware 4.0 but using Click. So we hope in the future version of the VMware ESXi release, this will be solved.

Small packet size flow always suffers more as performance decreases because when the traffic load is the same (100 Mbps 1 Gbps), small packet size indicates more IP packets and more headers to process by the CPU. The physical interrupts and computational resource are higher also, making the physical server more intense

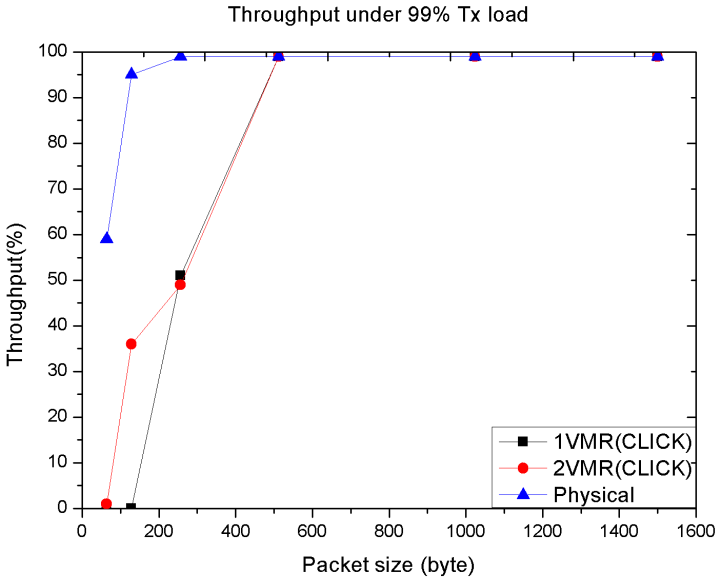

Figure 4.2: Click routing performance under 99% traffic load

and reduce the total routing abilities from the virtual machines running inside the server.

4.3.2 Xorp in Virtual Linux

In the second category of tests, totally 2 VMs are installed with Xorp as well as Dist protocol active in Ubuntu 9.04. They have been set as layer 3 packet routers by running a specific script in Xorp. In order to maintain the test condition as much the same as possible, we use IPv4 forwarding engine and static router module in Xorp script. After filling up the routing table and assigning the IP addresses from the script, we can say that the VMs are layer 3 routers since it can transmit packets based on IP headers processing, from one subnet to another (Multicast situation is not considered in our test for now).

The intensive experiments have been done in the previous chapter, so in this part we have only 2 back-end routing elements maximum for getting some idea of Xorp overhead in virtual environment. In every Xorp routing element, there are 2 interfaces active, belonging to 2 different subnets through the configuration from the script. And for ding a simple test of the Dist protocol overhead in VMware, we run the tests with and without Dist protocol enabled. The topology is shown in figure 4.4:

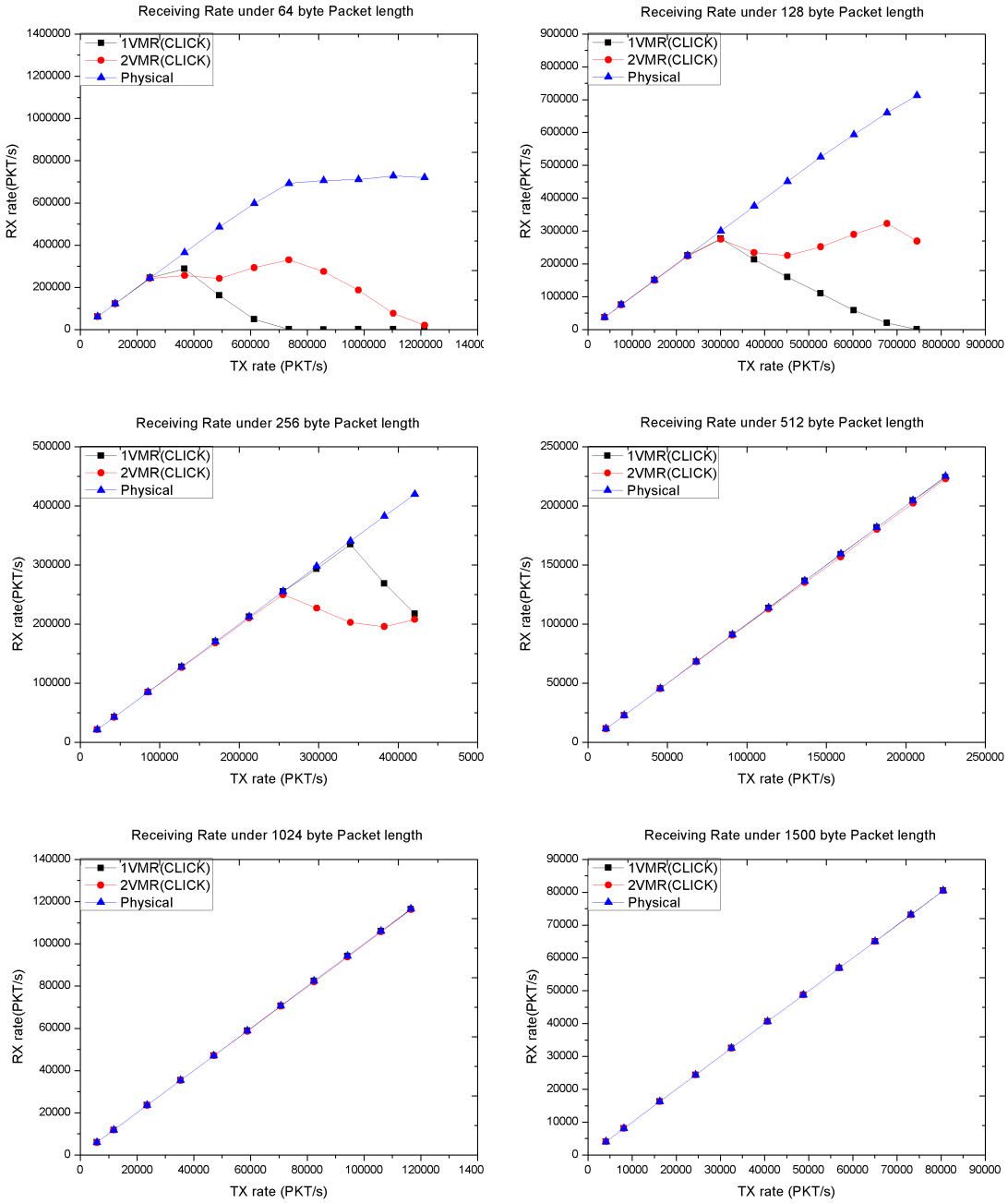

Figure 4.3: Click routing performance

In the Agilent Router Tester, only one module one port has been used to connect with the data port on the server, through a standard cat5e UTP cable. All the

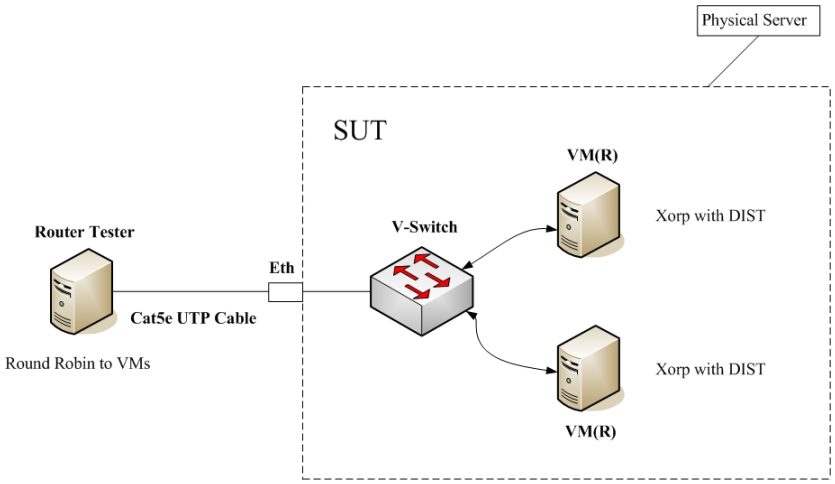

Figure 4.4: Xorp and Dist experiment in VMware

virtual interfaces inside the VM (R) and the physical interface have been connected through a virtual bridge in VMware. All the traffics will go through the data port in the server. By shutting down redirection ability in the Linux system, we can ensure the traffic make a loop from the router tester and the VM routers correctly. Besides the configuration in VM routers, the Router Tester should be set correctly, especially the next-hop MAC addresses is important.

In this experiment, all the VMs are used to route packets with the Linux kernel routing path, but using the route entry update algorithm from Xorp. We want to find out some basic idea of Xorp running inside virtual machines when routing standard IP packets.

The parameters used are reported as below when doing the testes:

- Packet Size : 64, 128, 256, 512, 1024, 1500 Bytes.
- Traffic Load : 50, 100, 200, 300, 400, 500, 600, 700, 800, 900, 990 Mbps , using Round Robin schema to inject into different VM routers.
- Physical Link Speed : 1Gbps.
- Number of Agilent Modules/ports : 1 Module with 1 port.
- Routing Table : 2 subnets with 4 entries for each of the VM router, totally 4 entries with 4 subnets in this experiment.
- Time : 30 seconds for each packet size with each load.

VMware ESXi version 4.0 has been installed into the Dell server on a SATA hard driver. Totally 2VMs in VMware have been set as layer 3 routers along with Xorp enabled, by using the Linux kernel forwarding path to route packets. Unbuntu 9.04 has been used as the operating system because of the highly updating speed. By setting the 2 virtual interfaces as gateways of the subnets in each VM router, through ifconifg, we make a basic configuration of 2 nets communicating through one software router each. The routing table has 4 entries totally, because of the identical routing table update from Dist protocol. 2 IP addresses have been assigned to the virtual interfaces, belonging to different subnets in different VM routers. In the Agilent Router Tester, packets with 253*2 IP addresses in specific subnets have been randomly generated and transmitted to the routers, using a round robin schema. The running VM routers in the physical server have been configured incrementally from 1 VM router to 2 VM routers during the tests. The traffic has been captured in the Router Tester based on the aggregation throughput. Results have been shown in figure 4.5 and 4.6(The physical routing performance has also been reported in the figure to make a compare)

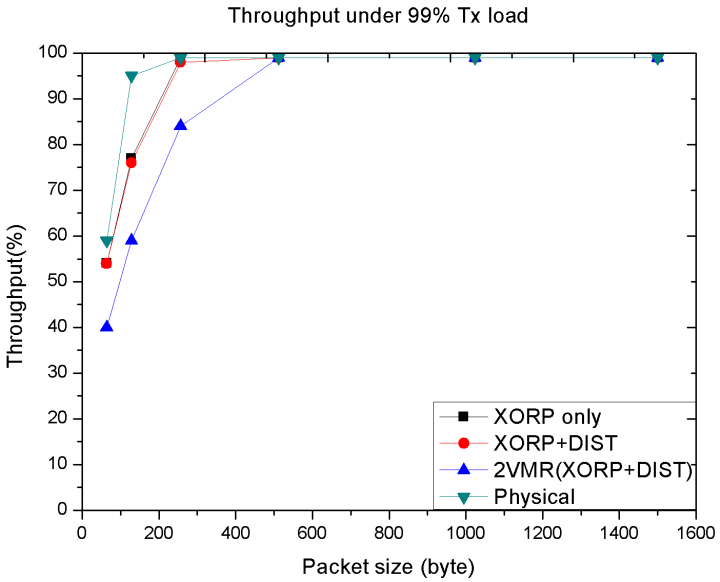

Figure 4.5: Xorp and Dist routing performance under 99% traffic load

From the results we get, we can find out that the overheads introduced by the Xorp and Dist protocol are both trivial if comparing to the results from chapter 3. This is reasonable. Since Xorp is only a routing protocol software. Its main job is to update the routing table according to different routing algorithm such as RIP,

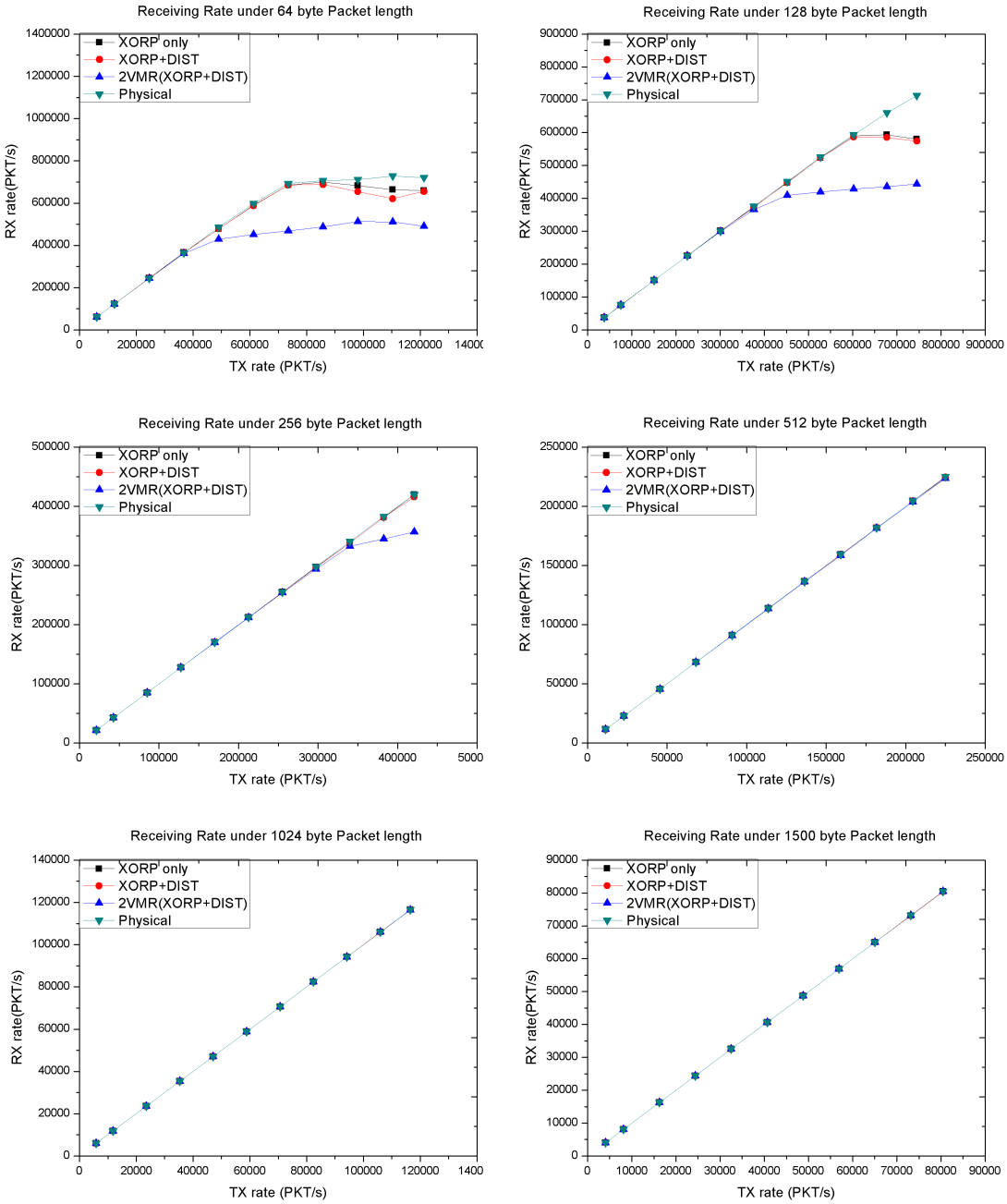

Figure 4.6: Xorp and Dist routing performance

OSPF, BGP and so on. The real routing task is done by the Linux kernel as if there is no Xorp enabled. When a packet comes, the routing stack in the kernel

will process the header and looks up the routing table to perform a longest prefix matching, no matter the routing table is updated manually or by some software like Xorp or Quagga. So the performance should not decrease so much. And from the Dist enable or not, we can see that our Dist protocol introduce a little overhead in small packet size flow when routing packets. That is because in our protocol the internal routing elements keep communicating through hello packets all the time, for preventing the sudden failure. This will cause some interrupts and open UDP port in the physical server, therefore make the routing ability decrease a bit little.

Small packet size flow always suffers more as performance decreases because when the traffic load is the same (100 Mbps 1 Gbps), small packet size means more IP packets and more headers to process by the CPU. The physical interrupts and computational resource are higher also, making the physical server more intense and reduce the total routing abilities from the virtual machines running inside the server.

4.3.3 Multistage architecture in virtual Linux

When things come to the final architecture, everything should be easy to implement since all the pieces have been tested previously. In this part we have configure 3 topologies totally. The first 2 are just toy cases but necessary because we want to maintain the testing condition same as the previous tests. The third topology is the final architecture, the exactly the same as our physical multistage software routers. At this part the 3 topologies will be introduced firstly, as well as the reasons. Then the performance results will be revealed to give out an idea of the feasibility of running the multistage software routers in virtual environment.

The first topology used is shown in figure 4.7, there is one LB in the front-end and 2 routers in the back-end. In the LB VM the Click has been installed to use a simple round robin algorithm to balance the traffic to the back-end routers. In the back routers, Xorp and Dist have been used to update the routing tables and communicate between each other. At the beginning of the experiments, only static routes have been used, and for maintain the consistent with previous tests, we have 2 routing entries in every router. Since all the experiments are proceed in VMware, we need one more physical interface to control the server, shown in the figure as controlling port. This physical port has been isolated from the data port and all the internal virtual interfaces by assigning different virtual switches. All the VMs are running Ubuntu 9.04 as the Linux operating system. In the Agilent router tester, only one module one port has been used to connect with the data port on the server, through a standard cat5e UTP cable. All the traffics will go through the data port in the server. By shutting down redirection ability in the Linux system, we can ensure the traffic make a loop from the router tester and the VM routers correctly. Besides the configuration in VM routers, the Router Tester should be set correctly,

especially the next-hop MAC addresses is important.

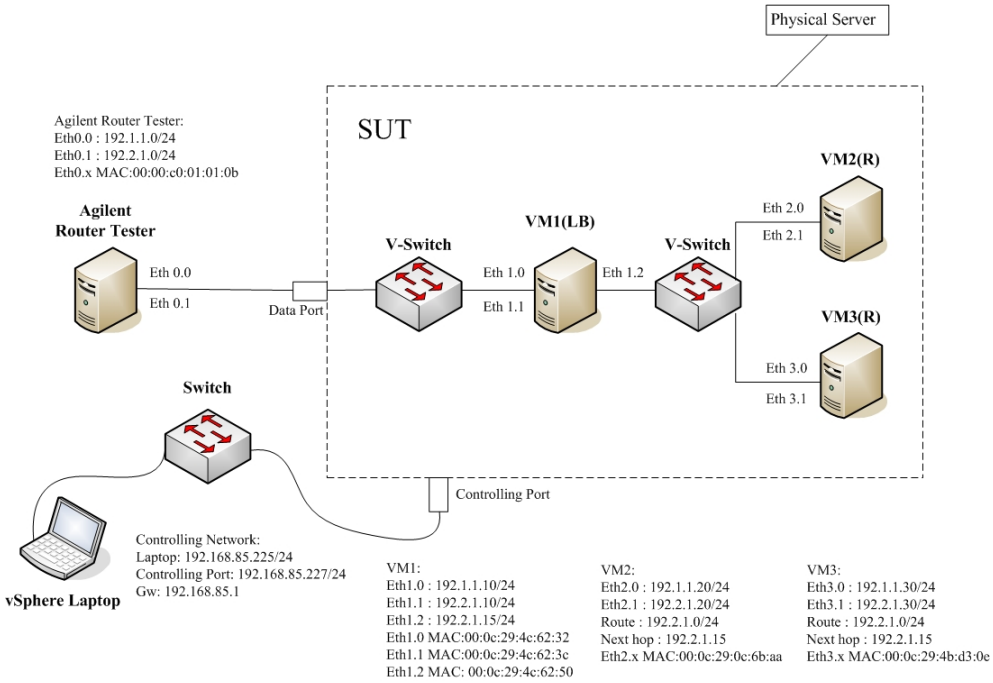

Figure 4.7: one LB and 2 Routing elements on One Interface architecture

The parameters used are reported as below when doing the testes:

- Packet Size : 64, 128, 256, 512, 1024, 1500 Bytes.
- Traffic Load : 50, 100, 200, 300, 400, 500, 600, 700, 800, 900, 990 Mbps
- Physical Link Speed : 1Gbps.
- Number of Agilent Modules/ports : 1 Module with 1 port.
- Routing Table : 2 subnets with 2 entries for each of the VM router
- Time : 30 seconds for each packet size with each load.

The second topology used is shown in figure 4.8, this time we upgrade the LB number to 2 but still use one physical interface as the data port. All the other conditions are exactly the same as the previous one. But the total routing entries have been increased also, from 2 entries to 4 totally. This time every LB has 2 interfaces belonging to 2 subnets, so from this architecture, we can get some idea of slicing the back-end data processing units to multiple users (different subnets). But

at beginning of the tests, we do not concentrated on this, so only the aggregation throughput has been collected in the Router Tester. In the Agilent router tester, only one module one port has been used to connect with the data port on the server, through a standard cat5e UTP cable. All the traffics will go through the data port in the server. By shutting down redirection ability in the Linux system, we can ensure the traffic make a loop from the router tester and the VM routers correctly. Besides the configuration in VM routers, the Router Tester should be set correctly, especially the next-hop MAC addresses is important.

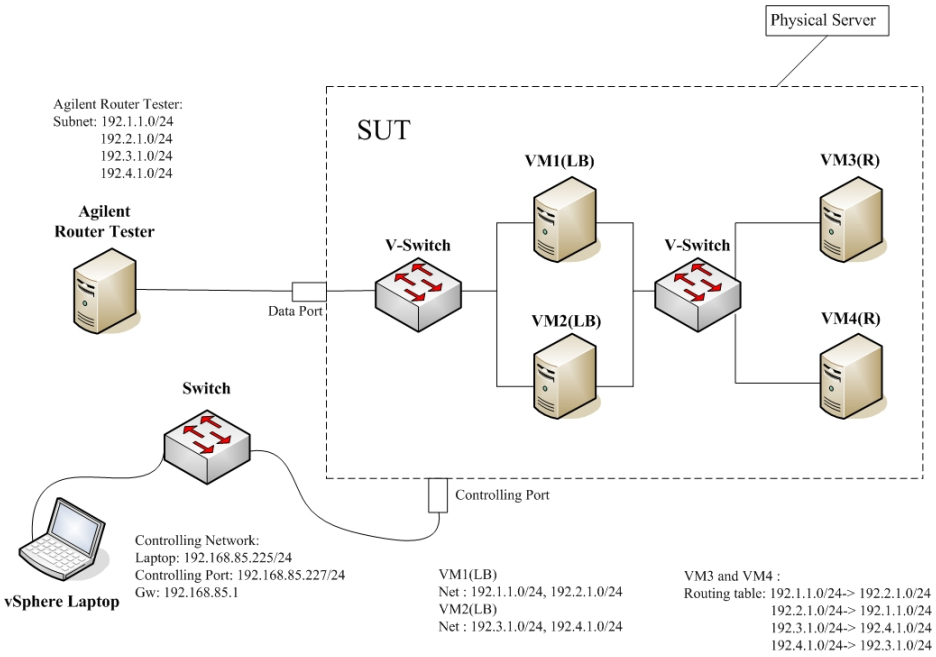

Figure 4.8: Two LBs and 2 Routing elements on One Interface architecture

The parameters used are reported as below when doing the testes:

- Packet Size : 64, 128, 256, 512, 1024, 1500 Bytes.
- Traffic Load : 50, 100, 200, 300, 400, 500, 600, 700, 800, 900, 990 Mbps, using a round robin schema to input the traffic into different subnet.
- Physical Link Speed : 1Gbps.
- Number of Agilent Modules/ports : 1 Module with 1 port.
- Routing Table : 2 subnets with 4 entries for each of the VM router
- Time : 30 seconds for each packet size with each load.

The last one is the real architecture that as our multistage software routers. Figure 4.9 shows the topology used when 2 LBs and 2 routing elements appear. Figure 4.10 shows the configuration topology in VMware. They are actually the same but from different point of views.

We can see that eventually there are as many physical NICs as the IP sub networks the software routers connect. As mentioned before, since the PCI or PCIx slots in a single PC are limited, we can enlarge the number of PCs in our architecture and overcome this limit as well. So for a better performance, we use one NIC with one subnet at this time.

2 LBs are working in Click to balance the packets to the routing elements as described before. Back-end routing elements are identical thanks to Dist protocol embedded in Xorp. Data port and controlling port are isolated by assigning different V-switch. All the interfaces in VMs are connected to the data port somehow, but through multiple switches shown in the 2 figures.

In the Agilent Router Tester 1 module 2 ports are used this time. They have been connected to the 2 data ports in the server through standard cat5e UTP cables. Every port has been assigned an IP address of a different subnet. So totally there are 2 sub nets in the system. By shutting down redirection ability in the Linux system, we can ensure the traffic make a loop from the router tester and the VM routers correctly. Besides the configuration in VM routers, the Router Tester should be set correctly, especially the next-hop MAC addresses is important.

The parameters used are reported as below when doing the testes:

- Packet Size : 64, 128, 256, 512, 1024, 1500 Bytes.
- Traffic Load : 50, 100, 200, 300, 400, 500, 600, 700, 800, 900, 990 Mbps
- Physical Link Speed : 1Gbps.
- Number of Agilent Modules/ports : 1 Module with 2 ports.
- Routing Table : 2 subnets with 2 entries for each of the VM router
- Time : 30 seconds for each packet size with each load.

From the third topology, we can get that the only difference from the second one is 2 data ports have been used. And this is different to the other tests in this thesis. All the tests before have been done in one physical data interface, we only want to get the routing ability idea of virtual machines, so no need to tuning the parameters as well as the number of physical interfaces connected with virtual NIC. But when things move to the multistage software routers migrate, besides the basic routing

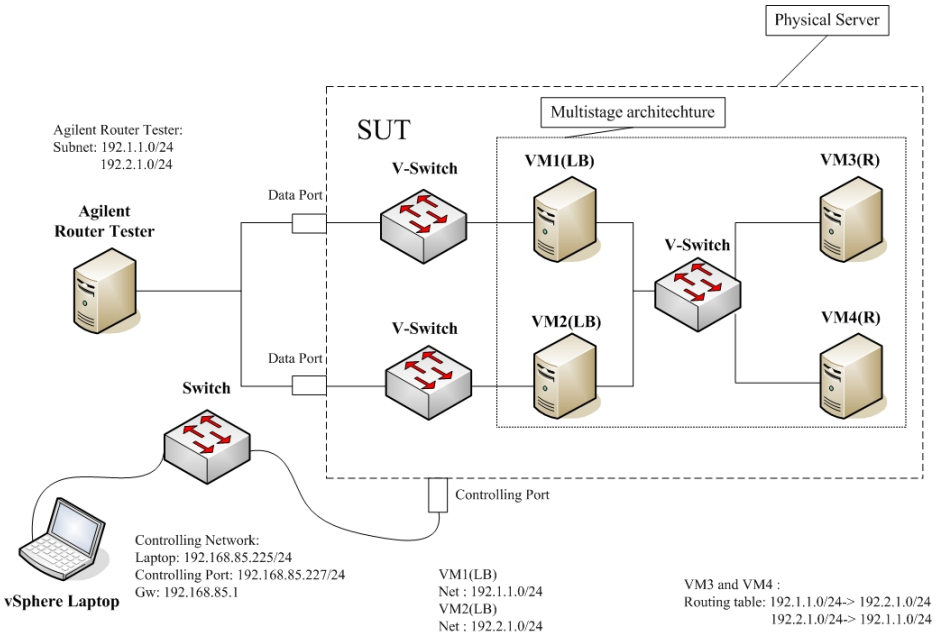

Figure 4.9: Two LBs and 2 Routing elements on 2 Interfaces architecture

functions configured inside the VM Linux operating system, other issues should be considered to make the routing performance good enough to future use.

Here all the throughput results belonging to the three different topologies will be shown, along with the physical server routing performance as the bench of mark.

Actually, from this series of results, the multistage software routers seem to perform very poor in virtual environment, when the packet size is small, the throughput decrease to hundred packets/s level. But if analyzing every stage carefully, we can find that this result is reasonable. Recall from the experiments on the LB, the aggregation throughput is very low compare to Linux only, because of the forwarding path in the Linux kernel. And when multiple LB running in Click appear in one physical server, we find out that the aggregation throughput is higher than only one VM acting as LB. This is also true here. From the figure xx and figure xx, we can see that when the traffic load is high in small packet size, the physical server should be over flooded, and 2 LBs curve is working better than only one VM LB. Here we are concentrated on single data port only.

On the back-end stage, when we active the Xorp and Dist protocol, the performance is decrease 5% 10% of the routing ability compare to not use this special software. This should be amplified when we connect everything together. So from the figures, we can see that even in 512 and 1024 packet size, the decrease of the

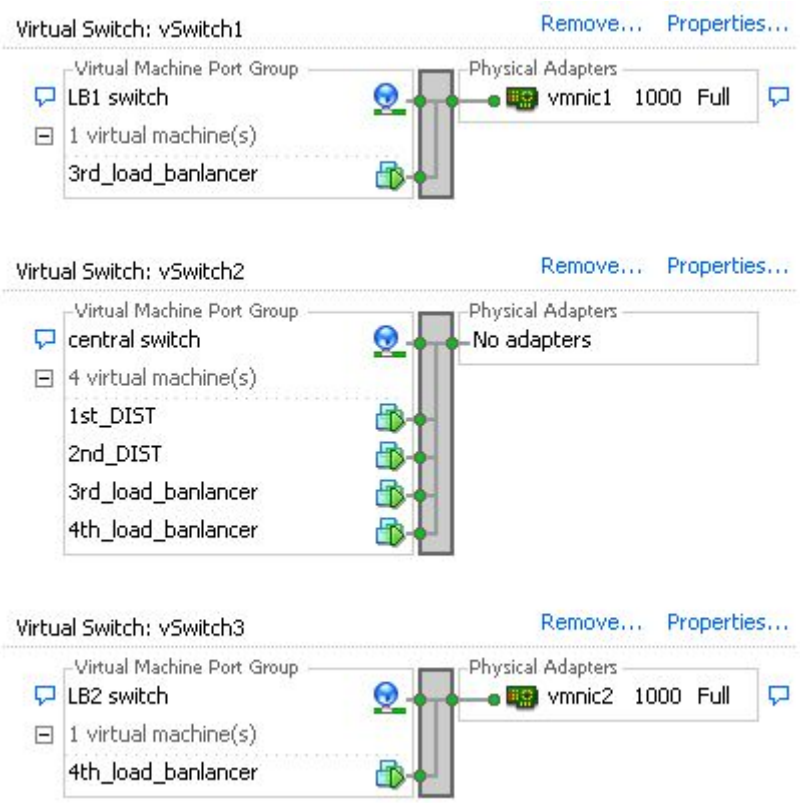

Figure 4.10: The Multistage Architecture in VMware Console

performance is vivid. But here if we consider the packets path inside the architecture, then we get the idea that the performance decrease mainly in the Click. The IP packets from the Agilent Router Tester go through the Click forwarding path one time when they reach the back-end routers, then another time when they come back to the Router Tester. So if the impairment is due to the substitution of the forwarding path, everything should be easy to understand. From the figure of 256 packet length, the total throughput in Click test is 300000 packets/s, and here is 150000 packet/s in the black lines, and red lines with 30000 packets/s drop, but the trend is the same. So we think the performance reduction in the multistage software is mainly due to Click when running the architecture in VMware.

The 2 LBs show better performance then only 1 LB works, this is the same as we discussed before. The phenomenon maybe has some relation with the VMXNET 3 drivers and NAPI. And by associating this results with the impairment in VMware ESXi 3.5 (256 and 512 byte packet flows), we can find that this is reasonable.

The last thing reflected from the figures is very interesting. The routing ability

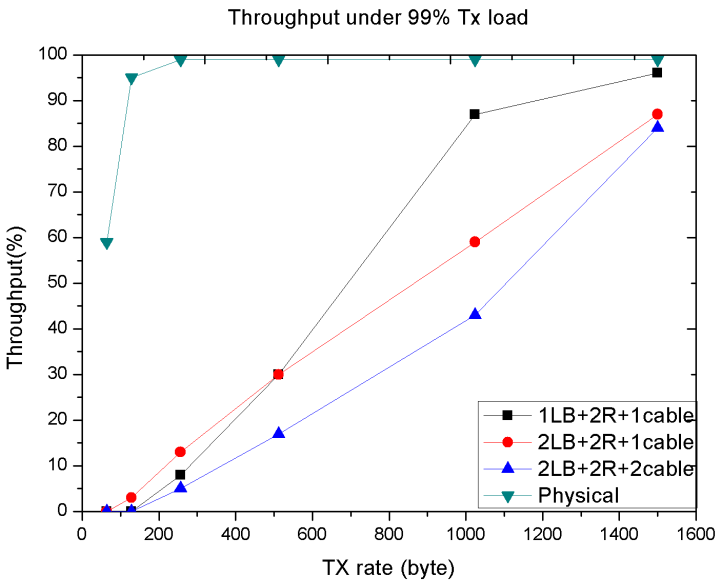

Figure 4.11: Multistage software routers routing performance under 99% traffic load in virtual environment

is poor when we use 2 physical NIC on the server compare to only one NIC. This is strange from what we thought. In physical environment, multiple NICs should be performed well as long as the back-end routing elements can sustain the huge traffic flow. But here in virtual machines, things are opposite to physical scenario. Multiple physical NICs show worse throughput than single NIC. This because our routing elements are inside one physical server, the total amount of traffic routing ability is limited by the server itself. If we add more interfaces and connect them to different virtual switches to create sophisticated topology inside the server, it will drag down the routing throughput because of all the context switch among the physical equipment on the mother board. All of these works consume the CPU resource and leave less for routing packets. So from the figures we can see that 2 physical interfaces' curve perform worse always.

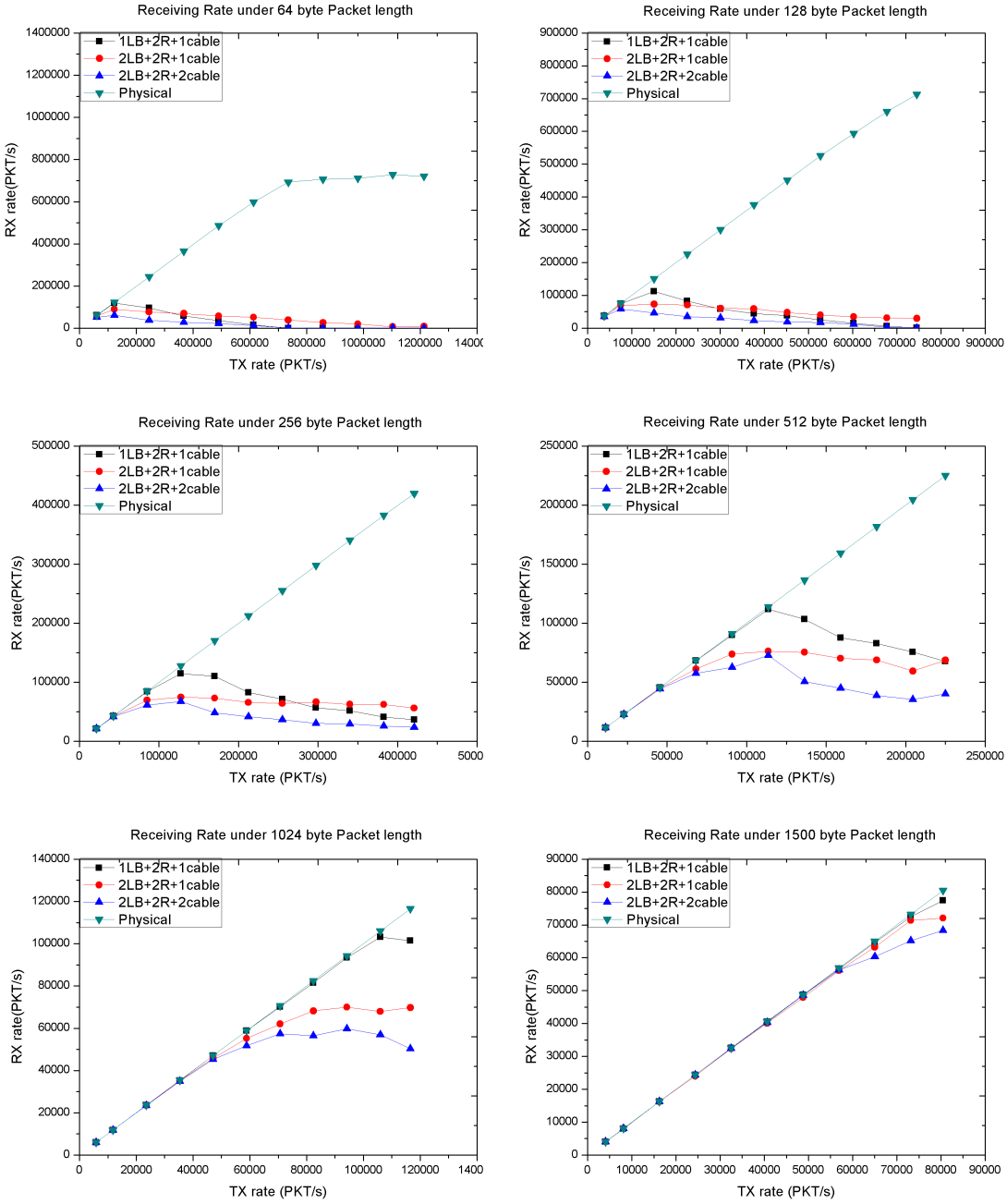

Figure 4.12: Multistage software routers routing performance in virtual environment

Chapter 5

Conclusion and Future Works

In this thesis, the routing ability in virtual environment based on open source architecture has been studied thoroughly. Then after describing the multistage software routers shortly, we have implemented this topology in virtual server successfully. Extensive tests to check the correct behavior of the router were run and data plane performance was examined.

Firstly, four types of virtualization techniques have been introduced, respectively known as hardware emulation, full virtualization, para virtualization and virtualization at the operating system level. The thesis focus on the full virtualization because of the small modification required to the operating system and its flexibility. Since VMware ESXi is a free version and owns fast evolution ability in full virtualization, we choose it to carry out most of the experiments However, to make a comparison between different virtualization technologies, XEN, exploiting the para virtualization paradigm was analyzed at the same time. Routing performance tests are implemented in both environments.

After discussion and analyzing the requests in the multistage software routers, three categories of tests have been done during the thesis work. The first one is concentrated on the influence from low power consumption virtual machines to an active virtual router inside one physical server. This work is associated with the energy saving function in future work. From the results we get, it can be said that the impact from the low functional virtual systems are not so much. Second category of tests is about the aggregation throughput of multiple virtual routers running inside one physical server. One to four virtual machines have been active as software routers incrementally. Round robin schema has been used to generate traffic flows to these routers from Agilent Router Tester. From the results we can say that multiple virtual routers perform worse than only one virtual router is active, due to the context switches and interrupts reside in the physical server. The more virtual routers are active, the worse the aggregation throughput it shows. So find a

trade off between the hardware and virtual routers it can sustain is an interesting task for the future. The last part of the tests for searching the routing ability in virtual machines is the fairness tests among multiple virtual routers. This time the proportion of the flows and the number of the flows are the parameters. Fair or not is a key performance index when we try to slice the multistage software routers to different users. And the tests' results show some dissimilarity in VMware and XEN. In VMware when the traffic input matrix is a uniform one, the output is nearly fair, but if the flow is divided into heavy and light ones, the heavy traffic flow is getting more resource from the hardware. In XEN, there is nearly no fair among the different flows. All the resource has been absorbed by the first open virtual machine in the domU. So at this moment, the VMware ESXi, especially version 4.0 (3.5 and 4.0 have been tested and reported both in the thesis) appears good in routing IP packets.

The second part of the thesis work is trying to implement the multistage software routers into virtual environment. Since among the three stages, the middle one is just a standard Ethernet switch, there is no modification inducted, so the thesis only deals with the first and third stage of the multistage routers. In the first stage load balancing functions have been implemented in Click physically now. By implementing LB into virtual machines, we have also made performance tests in some simple scenarios. But unfortunately, the performance is poor in Click mainly because it substitutes the Linux forwarding path to its own forwarding path. About the back-end routing array, Xorp and Dist protocol have been deployed in this thesis. The routing ability is nearly the same as the previous tests without Xorp. One thing need to be indicated is that during the tests we use only static routing table for simplicity, so Xorp functionally has been restricted because no routing entries are learned from routing protocols such as RIP, OSPF, BGP and so on.

By assembling the three stages together and with some configurations in VMware, we succeed in making physical multistage software routers into virtual server with different load balancers in the first stage and different physical NICs in the server. Performance tests have been done to show the correctness of the behavior inside the router. The data plan function can operate as desired, but the throughput is not so high. As we described in the thesis, every packet traverses the Click forwarding path 2 times is the main critical issue that drag down the total performance of the virtual multistage software routers.

In the future, a lot of works can be done to enhance this multistage architecture. A deep study is needed on Click, maybe through internal profile analyzer to get some idea of the internal operations in Click, for it is the most critical part for decreasing the performance now. VMware internal architecture should be revealed as possible as we can. Especially the virtual NIC driver VMXNET 3 should be

tested more. A trade off between the number of virtual machines inside one physical server and performance is also very important. As soon as the throughput has been tuned to a better degree, we will try to implement our multistage architecture in the FEDERICA networks to do more tests in wild network, hoping to make this architecture could cooperate with commercial routers as designed firstly. Besides all the work concentrated on the data plan, in the control plan there still exist a lot to do like resilience issue, database, control processor design etc. Besides these, a neat and kind user interface can be developed for future.

Bibliography

[1] Bolla,R., Bruschi, R.A high-end Linux based Open Router for IP QoS networks: tuning and performance analysis with internal (profiling) and external measurement tools of the packet forwarding capabilities. Proc. of the 3rd International Workshop on Internet Performance, Simulation, Monitoring and Measurements (IPS MoMe 2005), Warsaw, Poland (2005) pp. 203-214

[2] Bianco A., Finochietto, J. M., Galante, G., Mellia, M., Neri, F.: Open-Source PC-Based Software Routers: a Viable Approach to High-Performance Packet Switching. Proc. of the 3rd International Workshop on QoS in Multiservice IP Networks (QoS-IP 2005), Catania, Italy (2005) pp. 353-366

[3] E. Kohler, R. Morris, B. Chen, and J. Jannotti, "The Click modular router," ACM Trans. on Comput. Syst., vol. 18, no. 3, pp. 263-297,Aug. 2000.

[4] M. Handley, O. Hodson, and E. Kohler, "Xorp: An open platform for network research" in Proc. of the 1st Workshop on Hot Topics in Networks, Princeton, NJ, US, Oct. 28-29, 2002.

[5] GNU, "Quagga." [Online]. Available: http://www.quagga.net

[6] Raffaele Bolla, Roberto Bruschi, "RFC 2544 Performance Evaluation and Internal Measurements for a Linux Based Open Router", Department of Communications, Computer and Systems Science, University of Genova Italy.

[7] Raffaele Bolla, Roberto Bruschi, "The IP Lookup Mechanism in a Linux Software Router: Performance Evaluation and Optimizations", Department of Communications, Computer and Systems Science, University of Genova Italy.

[8] J. Duato, S. Yalamanchili, and L. Ni, Interconnection Networks: An Engineering Approach. Los Alamitos, CA, US: IEEE Computer Society Press, 1997.

[9] Cisco Systems, Carrier Routing System, http://www.cisco.com/ application/pdf/en/us/guest/products/ps5763/c1031/cdccont_0900aecd800f8118.pdf

[10] Juniper Networks, Routing Matrix, http://www.juniper.net/solutions/ literature/white_papers/200099.pdf

[11] P. Pradhan and C. Tzi-Cker, "Evaluation of a programmable clusterbased IP router," in Proceedings of the 9th International Conference on Parallel and

Distributed Systems (ICPADS'02), Taiwan, P.R. China, Dec. 17-20, 2002

[12] IETF, "Forwarding and control element separation (ForCES)." [Online].Available: http://www.ietf.org/html.charters/forces-charter.html

[13] E. Oki, Z. Jing, R. Rojas-Cessa, and H. J. Chao, "Concurrent roundrobin-based dispatching schemes for clos-network switches," IEEE/ACM Trans. on Networking, vol. 10, no. 6, pp. 830-844, 2002.

[14] VMware Inc, "Managing VMware ESXi", Available: http://www.vmware.com/products/esxi/

[15] Andre.Bianco, Jorge M. Finochietto, Giulio Galante, Marco Mellia, Fabio Neri, "Multistage Switching Architectures for Software Routers", Dipartimento di Elettronica, Politecnico di Torino, Networking Lab, Istituto Superiore MARIO Boella.

[16] A.Bianco, R.Birke, J.Finochietto, L.Giraudo, F.Marenco, M.Mellia, A.Khan, D.Manjunath, "Control and Management Plane in a Multi-stage Software Router Architecture", Dip. di Elettronica, Politecnico di Torino, CONICET - Universidad Nacional de Cordoba, Argentina, IIT Bombay, India,

[17] I. Keslassy and N. McKeown, "Maintaining packet order in two-stage switches" in Proceedings of IEEE INFOCOM, New York, NY, US, June 23-27, 2002, pp. 1032-1042.

[18] M. Tim Jones, Consultant Engineer of IMB "An overview of virtualization methods, architectures, and implementations", form IBM technical library, http://www.ibm.com/developerworks/linux/library/l-linuxvirt/

[19] Mendel Rosenblum, Tal Garfinkel, "Virtual Machine Monitors: Current Technology and Future Trends", VMware Inc, Stanford University.

[20] VMware official site http://www.vmware.com/products/esxi/

[21] XEN official site http://www.xen.org/

[22] Norbert Egi Adam Greenhalgh Mark Handley Mickael Hoerdt, Laurent Mathy, Tim Schooley "Evaluating Xen for Router Virtualization",Computing Dept.,Lancaster University,UK, Dept. of Computer Scienc, University College London, UK

[23] Norbert Egi Adam Greenhalgh Mark Handley Mickael Hoerdt, Laurent Mathy, Tim Schooley "The Optimization of Xen Network Virtualization",Computing Dept.,Lancaster University,UK, Dept. of Computer Scienc, University College London, UK

[24] J. H. Salim, R. Olsson, A. Kuznetsov, "Beyond Softnet", Proc. of the 5th annual Linux Showcase & Conference, November 2001, Oakland California, USA.

[25] E.Kohler, R.Morris, B.Chen, J.Jannotti, M.F.Kaashoek The Click Modular Router, ACM Trans. Comp.Sys, vol.18, no.3, Aug.2000, pp.263-297.

[26] XORP, Inc, XORP User Manual, Version 1.6 January 7, 2009.

[27] Brink, P., Castelino, M., Meng, D., Rawal, C., Tadepalli, H.: Network processing performance metrics for IA- and IXP-based systems. Intel Technology Journal 7 (2003)