POLITECNICO DI TORINO

SCUOLA DI DOTTORATO
Dottorato in Ingegneria Elettronica e delle Comunicazioni
XXV Ciclo

Tesi di Dottorato

Energy Saving And Virtualization Technologies in Switching

Nanfang Li
Mtr. 169631

Tutore
Prof. Andrea Bianco

Coordinatore del corso di dottorato
Prof. Ivo Montrosset

December 2012

Summary

Switching is the key functionality for many devices like electronic Router and Switch, optical Router, Network on Chips (NoCs) and so on. Basically, switching is responsible for moving data unit from one port/location to another (or multiple) port(s)/location(s). In past years, the high capacity, low delay were the main concerns when designing high-end switching unit. As new demands, requests and technologies emerge, flexibility and low power cost switching design become to weight the same as throughput and delay. On one hand, highly flexible (i.e, programming ability) switching can cope with variable needs stem from new applications (i.e, VoIP) and popular user behavior (i.e, p2p downloading); on the other hand, reduce the energy and power dissipation for switching could not only save bills and build echo system but also expand components life time. Many research efforts have been devoted to increase switching flexibility and reduce its power cost. In this thesis work, we consider to exploit virtualization as the main technique to build flexible software router in the first part, then in the second part we draw our attention on energy saving in NoC (i.e, a switching fabric designed to handle the on chip data transmission) and software router.

In the first part of the thesis, we consider the virtualization inside Software Routers (SRs). SR, i.e, routers running in commodity Personal Computers (PCs), become an appealing solution compared to traditional Proprietary Routing Devices (PRD) for various reasons such as cost (the multi-vendor hardware used by SRs can be cheap, while the equipment needed by PRDs is more expensive and their training cost is higher), openness (SRs can make use of a large number of open source networking applications, while PRDs are more closed) and flexibility. The forwarding performance provided by SRs has been an obstacle to their deployment in real networks. For this reason, we proposed to aggregate multiple routing units that form an powerful SR known as the Multistage Software Router (MSR) to overcome the performance limitation for a single SR. Our results show that the throughput can increase almost linearly as the number of the internal routing devices. But some other features related to flexibility (such as power saving, programmability, router migration or easy management) have been investigated less than performance previously.

We noticed that 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 could provide many flexible features like hardware and software decoupling, encapsulation of virtual machine state, failure recovery and security, to name a few. Virtualization permits to build multiple SRs inside one physical host and a multistage architecture exploiting only logical devices. By doing so, physical resources can be used in a more efficient way, energy savings features (switching on and off device when needed) can be introduced and logical resources could be rented on-demand instead of being owned. Since virtualization techniques are still difficult to deploy, several challenges need to be faced when trying to integrate them into routers. The main aim of the first part in this thesis is to find out the feasibility of the virtualization approach, to build and test virtualized SR (VSR), to implement the MSR exploiting logical, i.e. virtualized, resources, to analyze virtualized routing performance and to propose improvement techniques to VSR and virtual MSR (VMSR).

More specifically, we considered different virtualization solutions like VMware, XEN, KVM to build VSR and VMSR, being VMware a closed source solution but with higher performance and XEN/KVM open source solutions. Firstly we built and tested each single component of our multistage architecture (i.e, back-end router, load balancer)inside the virtual infrastructure, then and we extended the performance experiments with more complex scenarios like multiple Back-end Router (BR) or Load Balancer (LB) which cooperate to route packets. Our results show that virtualization could introduce 40 % performance penalty compare with the hardware only solution. Keep the performance limitation in mind, we developed the whole VMSR and we obtained low throughput with 64B packet flow as expected. To increase the VMSR throughput, two directions could be considered, the first one is to improve the single component (i.e, VSR) performance and the other is to work from the topology (i.e, best allocation of the VMs into the hardware) point of view. For the first method, we considered to tune the VSR inside the KVM and we studied closely such as Linux driver, scheduler, interconnect methodology which could impact the performance significantly with proper configuration; then we proposed two ways for the VMs allocation into physical servers to enhance the VMSR performance. Our results show that with good tuning and allocation of VMs, we could minimize the virtualization penalty and get reasonable throughput for running SRs inside virtual infrastructure and add flexibility functionalities into SRs easily.

In the second part of the thesis, we consider the energy efficient switching design problem and we focus on two main architecture, the NoC and MSR. As many research works suggest, the energy cost in the Communication Technologies (ICT) is constantly increasing. Among the main ICT sectors, a large portion of the energy consumption is contributed by the telecommunication infrastructure and their devices, i.e, router, switch, cell phone, ip TV settle box, storage home gateway etc.

More in detail, the linecards, links, System on Chip (SoC) including the transmitter/receiver on these variate devices are the main power consuming units. We firstly present the work on the power reduction of the data transmission in SoC, which is carried out by the NoC. NoC is an approach to design the communication subsystem between different Processing Units (PEs) in a SoC. PEs could be different elements such as CPU, memory, digital signal/analog signal processor etc. Different PEs performs specific tasks depending on the applications running on the chip. Different tasks need to exchange data information among each other, thus flits (chopped packet with limited header information) are generated by PEs. The flits are injected into the NoC by the proper interface and routed until reach the destination PEs. For the whole procedure, the NoC behaves as a packet switch network. Studies show that in general the information processing in the PEs only consume 60 % energy while the remaining 40 % are consumed by the NoC. More importantly, as the current network designing principle, the NoC capacity is devised to handle the peak load. This is a clear sign for energy saving when the network load is low.

In our work, we considered to exploit Dynamic Voltage and Frequency Scaling (DVFS) technique, which can jointly decrease or increase the system voltage and frequency when necessary, i.e, decrease the voltage and frequency at low load scenario to save energy and reduce power dissipation. More precisely, we studied two different NoC architectures for energy saving, namely single plane chip and multi-plane chip architecture. In both cases we have a very strict constraint to be that all the links and transmitter/receivers on the same plane work at the same frequency/voltage to avoid synchronization problem. This is the main difference with many existing works in the literature which usually assume different links can work at different frequency, that is hard to be implemented in reality. For the single plane NoC, we exploited different routing schemas combined with DVFS to reduce the power for the whole chip. Our results haven been compared with the optimal value obtained by modeling the power saving formally as a quadratic programming problem. Results suggest that just by using simple load balancing routing algorithm, we can save considerable energy for the single chip NoC architecture. Furthermore, we noticed that in the single plane NoC architecture, the bottleneck link could limit the DVFS effectiveness. Then we discovered that multiplane NoC architecture is fairly easy to be implemented and it could help with the energy saving. Thus we focus on the multiplane architecture and we found out that DVFS could be more efficient when we concentrate more traffic into one plane and send the remaining flows to other planes. We compared load concentration and load balancing with different power modeling and all simulation results show that load concentration is better compared with load balancing for multiplan NoC architecture.

Finally, we also present one of the the energy efficient MSR design technique, which permits the MSR to follow the day-night traffic pattern more efficiently with our on-line energy saving algorithms.

Acknowledgements

Firstly, I would like to express my sincere gratitude to my advisor Prof. Andrea Bianco, for the profound advice, suggestions and helpful discussions during the past three years. His precise observation and deep insight on my research topics helps me significantly.

Secondly, I would like to thank Prof. Paolo Giaccone, who provided me very solid guides on varies aspects such as research methodologies, implementation techniques, presentation skills etc. Without the help and advise from him, it would not be possible to conduct the research works on Network on Chip.

Thirdly, I would like to thank Luca Giraudo, Robert Birke, Fikru Getachew Debele, Prof. Mario Roberto Casu, Prof. Luca Abeni for different helps and collaboration on my diverse research topics. It was very delighted, interesting and helpful to work with them all, their different scientific expertise is pretty valuable to me.

Last but not least I want to thank my parents and my wife Boyun who give me support continuously during my whole study period. Without their help, it would have been never possible to accomplish the Ph.D degree.

Contents

List of Tables

List of Figures

Part I

Virtualization Techniques in Software Router

Chapter 1

Introduction

Routers are the most important components for modern packet networks and of the Internet in particular. The demand 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. Routers have been able to follow the performance growth by offering an ever increasing transmission and switching speed, mostly thanks to the technological advances of microelectronics. But from other points of view, nearly all of these high-end enterprise routers and core routers are based on proprietary architectures from such as Cisco, 3com, HuaWei etc. These may raise the compatibility issue. In the meantime, the configuration and training cost to manage multi-vender routers with different architecture is very high. Under such conditions, the market requirement and new user demands stimulate the emergence of open source software router, which is based on the off-the-shell Personal Computer (PC).

1.1 Software Router

Open source Software Routers (OSRs) represent an appealing alternative to proprietary network devices because of the wide availability of multi-vendor PC hardware, their low cost, the continuous performance evolution driven by the PC-market economy, and the large availability of open-source software for networking applications, such as Linux, BSD, Click[2], XORP[3] and Quagga[4].

Indeed, despite of the limitations of bus bandwidth, CPU and memory-access speed, current PC-based routers have a traffic-switching capability in the range of some Gigabits per second, which is more than enough for a large number of applications. Furthermore, keeping this in perspective, performance limitations are compensated by the natural PC architecture evolution driven by Moore's law. However, high-end performance cannot be easily obtained today with PC-based routers.

Thus the research community devoted a lot of efforts either to optimize the internal architecture of OSR [5, 6] or to devise strategies to aggregate software routers to build more powerful routing units [7, 8, 9, 10, 11, 12, 13].

1.2 Multistage Software Router

To overcome some of the limitations of OSRs based on a single PC, we proposed to create a large size OSR exploiting a distributed, multi-stage switching architecture [14, 15, 16, 17]. Performance measurements show that routing capabilities may scale up almost linearly with the number of internal elements. Furthermore, implementation of recovery mechanisms into the management plane can increase OSR resilience to close the gap with carrier-grade routers.

The multi-stage router defined in [14] is organized in three stages (Fig. 1.1), characterized by three different internal elements: first stage load balancers (LB), second stage interconnecting switches and third stage back-end routers (BR). In the first stage, LBs permit to scale the number of interfaces, mask the internal router structure to external devices and balance the incoming traffic load by sending packets to selected BRs according to a round-robin- or hash-based balancing scheme. The second stage is composed by one or more Ethernet switches that implement a logical full-mesh among elements in the first and third stages. Finally, the third stage is composed of BRs that forward packets at the IP layer. An internal control protocol, named as DIST [15], runs to manage the architecture, to identify the internal elements, to configure LBs and BRs, to distribute and synchronize the routing tables among BRs and to introduce features such as energy saving mechanisms based on switching on/off internal elements depending on the load [18]. DIST is based on the master-slave paradigm and the *Virtual CP* (i.e. virtual control processor) is the DIST master running on an elected BR, as more precisely described in [15]. This architecture is a centrally controlled distributed router architecture, similar in spirit to the centralized controller based approach pursued in Openflow [19].

The main advantages of this multistage architecture can be summarized 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, that is, reconfiguration can occur in case of any PC/element failure.

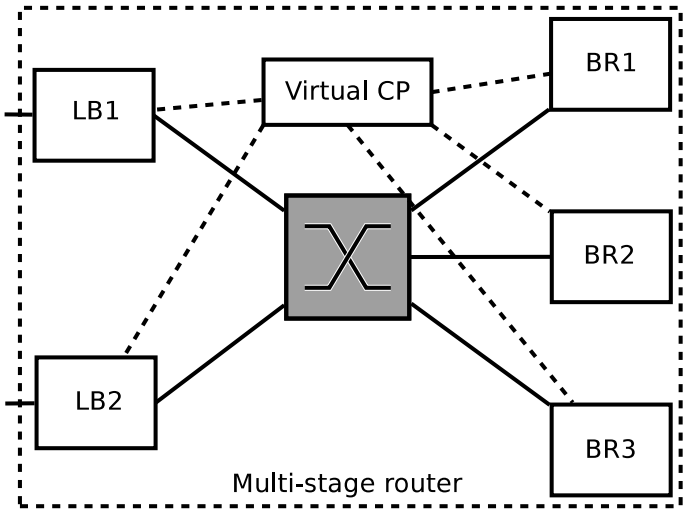


Figure 1.1. Example of the multi-stage router composed by two load balancers and three back-end routers: all internal elements run on a different PC to improve performance and reliability.

- Support a fully asynchronous behavior.
- Provide functional distribution, to overcome single-PC CPU limitations, for example, allowing the offloading of CPU-intensive tasks such as filtering/cryptography to dedicated PC.

1.3 Virtualization on Software Router

As recognized by several researchers [20, 21, 22], virtualization techniques may become an asset in networking technologies in general and in the field of distributed router architectures in particular. Instead of buying high-end proprietary hardware-based routers, an ISP or a network administrator could either manage or even rent logical elements running on VMs (Virtual Machines) to build either a multi-stage router architecture, i.e, a centralized interconnection network, or a more classical meshed network of routers. This enables on one hand to re-use and share the existing computing power available into the enterprise data-centers and on the other hand, to flexibly adjust the router capacity to adapt it to traffic needs, enabling energy-aware control techniques.

More precisely, when building the Multi-stage Software Routers (MSR) based on VMs, three main advantages can be highlighted:

- larger scalability: new internal elements can be deployed in a seamless way when traffic increases or more interfaces are needed. This enables renting of

resources from data center servers, for example when new VMs are needed to add forwarding capacity;

- easier management and reliability: migration of VMs during maintenance periods can be implemented and faster reaction to failures should be expected by booting new VMs on general purpose servers;

- slicing: sharing of the same physical infrastructure among different multi-stage routers which are possibly dedicated to different types of traffic (e.g., logical separation of the operational and of the experimental networks).

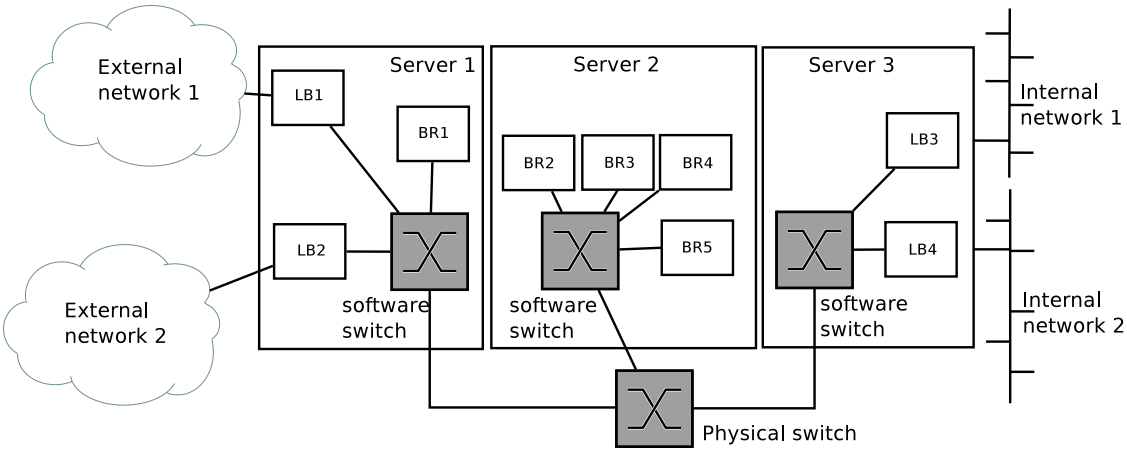


Figure 1.2. Example of a virtualized multi-stage router architecture: three physical servers hosts different VMs building one virtualized multi-stage router.

As an example, we report in Fig. 1.2 a use case referring to a distributed router based on the multi-stage architecture. External and internal network connections are terminated into the computing server farm, where VMs act as LBs, switches or BRs. This solution permits i) to locate VMs on different physical servers to upgrade the overall routing capacity; ii) to share the same physical server among several VMs to increase resource utilization; iii) to build the multi-stage architecture in a mixed approach exploiting both virtual and physical elements; iv) to deploy consolidation mechanisms, e.g. to move all virtual routers to a physical server and turn off unused servers during low traffic periods. These features introduced by virtualization are useful to improve performance and flexibility of the multi-stage architecture. However, some issues need more investigation:

- performance penalties due to hardware abstraction and resource contention [23];

- additional complexity for the management plane, due to the joint presence of physical and virtual resources;
- larger latency due to the introduction of additional virtualization layers and internal operational cost.

In the first party of the thesis we focus the attention on assessing the feasibility of the multi-stage architecture based on VMs and on the identification of performance impairments due to the use of virtualization techniques. We also provide guidelines on how to map the *virtual resources* to the existing physical devices and on properly binding VMs to CPU to CPU cores for higher throughput.

Chapter 2

Virtualization and Related Works

Virtualization has been developed for nearly 40 years since the late 1960s. In the early stage, general purpose computing unit was very expensive, thus it is not possible to own private computers for each single person, then sharing the same hardware among multiple users with different purposes was the key solution to meet with user requirements, this could be considered as the very basic virtualization concept at that time. But in the 1980s and 1990s, the multi-task Operating Systems (OS) and the drop in hardware cost made the "sharing" feature no more interesting. As many minicomputers and personal computers became a reality and popular, virtualization was considered only as a historical curiosity by the academics and industry in late 1990s.

Although the development of multitasking OS and hardware cost dropping prevented the virtualization techniques in 1990s, as time passed on, they became positive force for virtualization. Nowadays, more and more powerful PCs with low cost appear in the markets, which provide the opportunity to run multiple OSs under a single host. Further more, cheap hardware cost lead to a over-provision scenario in current telecommunication networks, which provide not only the under used server and network but also heavy management overhead. The increased functionalities have also made servers and networks fragile and vulnerable. In such situations, virtualization become an appealing solution, which can slice the the hardware for multiple users to increase resource utilization and consolidate light loaded servers into few to alleviate the management effort. Virtualization could also provide security and reliability functionalities more easily than hardware only solution, and it has some peculiar feature like providing researchers the opportunity to test and run innovative experiments on industry network. All these features stimulate virtualization a hot topic in recent years. The leading company like VMware, IBM, Microsoft developed their own solutions for virtualization such as VMware ESX(i) server, Windows Server Hyper-V, System-Z etc [24, 25, 26]. In research centers and universities, people are developing new methodologies based on virtual machines to

solve new problems such as mobility, security and manageability etc.

2.1 Virtualization Techniques

The main idea of virtualization is to represent some physical resource (e.g. CPU and memory) with an abstract description which can be identical or completely different (e.g. emulation) to the physical machine depending on the purpose of virtualization. This concept was firstly used to improve the CPU resource efficiency and to support different requirements from different users (e.g. different operating systems on the same hardware at the same time). Multiple users could share the same hardware without interfering with each other. Even more, they feel like to have exclusive access to the physical resources. This technology is named as *full virtualization*: all features of the physical hardware are reflected into one or more virtual machines (VM), which is an abstract representation of the complete physical machine. A middle layer software, named as *hypervisor* or *VMM* (Virtual Machine Monitor), is used to take control over the real hardware, to manage its abstractions and to rule the contentions to resources (e.g. using resource schedulers for CPU, memory, storage and network too).

Today many virtualization techniques are available and they are differentiated by the completeness of the VM description and by the level of integration among the hypervisor and the *guest* operating system. A brief list of virtualization techniques is as follow:

- **Emulation**: VM and physical machine are completely different, hypervisor intercepts and translate all interactions among virtual and physical devices with high computational costs. VM description is complete. The typical objective is to run the software designed for a different architecture. Bochs [27] is an emulation-based project which gives the ability to run any x86 operating system on other architectures like Alpha or PPC. QEMU [28] is another emulation-based one, supplying the similar functionalities like Bochs but with an alternative running mode: allowing the binary codes compiled for different architecture launch directly on your x86 Linux, a light weight emulation without the I/O periphery.

- **Full virtualization**: The VM description is complete, but virtual and physical devices mostly belong to the same family. The hypervisor is simpler and more efficient than in the emulation case, but it has to virtualize all the features of the physical hardware. Examples are VMware ESX [24] and IBM System-Z [26];

- **Para-virtualization**: The VM description is partial and the guest VM is modified to be aware of the virtualization infrastructure and to interact proactively with the hypervisor. This solution is efficient, but it may be not feasible in some cases when guest O.S. can not be modified and there is no support from the O.S. developer. Para-virtualization examples are XEN [29] and KVM [30][1];

- **Operating System-level virtualization**: the virtualization interface is moved up in the architecture, since O.S. kernel resources are virtualized instead of the physical resources. This approach is very efficient and the integration among hypervisor and guest VMs is very tight, thus strong constraints are imposed on the guest O.S. (e.g. same kernel for the hypervisor and guest O.S.). Examples are OpenVZ [31] and Linux Namespaces [32].

Thanks to the fast evolution of semi-conductor technology and the large availability of hardware resources, the main motivations for the usage of virtualization today [33, 34] are no longer to share the expensive hardware resources but to:

- **Consolidate of resources**: to improve the utilization of resources to minimize costs (CAPEX and OPEX), power consumption and management complexity;

- **Decouple services from servers**: to make services independent from physical resources and to introduce more flexibility (e.g. service migration, early stage cross-platform software test), security (e.g. isolation), redundancy and fault tolerance.

2.2 Networking Virtualization

Since the virtualization of network resources is considered as a major opportunity to foster the innovation in the Internet and to solve the Internet *ossification* [33, 34, 34] problem, many research projects are working on virtualization technologies today. For instance, GENI [35] is a NSF research project (U.S.A.) on future Internet architectures where virtualization technologies are used to create slices of the network. On the European side, the FP7 FEDERICA project [36, 20] uses a similar approach based on virtualization to create network slices to support research activities. Furthermore, network device manufacturers like Cisco and Juniper are supporting network virtualization into their products as well. In most

[1]Both XEN and KVM have full virtualization feature, which we do not consider due to the poor performance.

recently, many companies like Amazon and Microsoft are building services related to network virtualization and cloud computing.

Despite the works related to the high level network virtualization, many other researches focus the attention on how to virtualize the key network component, i.e, router or switching point, and on evaluating or improving its performance. Software routers based on XEN has been studied extensively [29, 37, 23], the main conclusion is that the DomU, i.e, the guest domain, is not suitable for routing purpose due to the high overhead and low performance. More recent works on XEN [38, 39] shows that it is possible to optimize the router internal architecture (i.e, maintain the packet in the same CPU cache, multi-queue network interface support) to highly improve the SR's performance. Some similar works related with other virtualization technologies like OpenVZ, Linux namespace [40], User-Mode-Linux and Click [41] can be found in the literature. Not only inside academia, virtualization is also popular commercially [42], due to their flexibility and capex/opex saving, which also motivated us to build our MSR by virtual resource only.

To improve the SR's performance, we can either optimize the SR's internal architecture as discussed in the aforementioned works, or we can consider from the architectural point of view, i.e, combine multiple routing elements in a systematical way to obtain a "big" router which has the aggregate performance and higher number of interfaces. For instance, Router Bricks [43] exploits server cluster and valiant load balancing schema to build a flat router architecture, i.e, all servers are external load balancers and internal routing elements as well as intermediate nodes. In our research project we focus the attention on the multi-stage architecture and on the adaptations required to run it on top of a virtualization infrastructure. We are interested in using virtualization as a tool to create the internal elements (i.e, routing units) of our distributed router architecture. Our efforts are devoted to understand the functional limitations and to evaluate the unavoidable performance losses on the multi-stage router architecture introduced by the additional virtualization software layers as presented in [23].

Chapter 3

Virtualized Multi-stage Software Router

We wish to test the feasibility of building the multi-stage architecture in a virtual environment, where all internal elements are running on VMs instead of physical PCs. In the original multi-stage implementation, FPGA-based LBs were also considered to improve balancing speed. However, in this thesis we limit our analysis to software LB implementations based on Click Modular Router to take full advantage of the virtualization infrastructure.

Among the existing virtualization frameworks, the emulation-based and the Operating System (OS) level-based (also named container-based) solutions are not suitable for our purposes. On one hand container-based solutions are fast, but the host OS and the guest OS share the same kernel [44]. This is a very tight constraint in the heterogeneous environment that we are considering, where VMs may move among different physical servers to exploit existing processing power. Indeed we should guarantee the same OS on all servers, which may be infeasible from a practical point of view. On the other hand emulation-based approach has different design objectives and it shows very poor performance when compared to full- and para-virtualization [45]. Thus, we draw our attention on two hypervisor-based infrastructures: XEN and VMware ESXi. Both provide similar functionalities (e.g. full virtualization or para-virtualization depending on CPU features) using different approaches. XEN is an open-source project based on the Linux kernel, meanwhile VMware is a closed-source project.

Recall that the proposed architecture is composed of three stages:

- first-stage: layer-2 LBs distribute the input traffic load to BRs.
- interconnection network: a mesh-based switched network between the first stage LBs and the third stage BRs. Multiple paths between the LBs and BRs could exist to support fault recovery.

- third-stage: BRs, i.e, forwarding engines, route packets to the proper LB.

In the next sections we analyze the implementation details of the main components (e.g. click-based software LB, software switches-based internal network and BRs) running as XEN and VMware VMs. We show that the MSR architecture is feasible in both cases, but with different configurations, constraints and performance.

3.1 XEN-based Implementation

The design objective of the virtual MSR is to build the multi-stage virtualized architecture using independent VMs sharing a common physical infrastructure. Every VM is using its own routing or load balancing table and forwarding mechanisms. No resource sharing can be envisioned among LBs and among BRs.

The implementation of the virtualized multi-stage router is relatively easy in XEN, because XEN provides standard Ethernet switch functionalities (e.g. using *bridge-utils* software package) and no limitations are imposed on VMs (both LB and BR). Thus the MSR implementation on XEN is equivalent to implement it on physical Linux PCs. Three kinds of internal networking configurations are available, namely *bridged*, *routed* and *hybrid* [23, 46]. In the multi-stage router context the *routed* or *hybrid* configurations cannot be used although they would show better performance as described in the research literature. Indeed, an important feature of the multi-stage architecture is that the various BRs should be identical in layer-3 (i.e, IP addresses on the interfaces) since each BR could potentially process all the traffic during low load periods. The LBs balance the traffic to BRs based only on their MAC addresses. The *routed* and *hybrid* configurations in XEN require IP routing to send packets from Dom0 to DomU, which implies that different DomU should have different IP addresses. For this reason, we rely on the *bridged* configuration, which is less efficient from the performance point of view, but it permits to build the virtual MSRs inside XEN.

3.2 VMware-based Implementation

Being VMware closed source, it is difficult to obtain details on the internal networking architectures and available features. Thus, we rely mainly on the *vSphere* management interface and VMware drivers to interact with the virtualization infrastructure. In our experiments we choose VMware ESXi, the free version of the most popular VMware ESX server virtualization; it lacks some management features, while the same technological core is used. Both solutions are targeting the server virtualization (e.g. VM hosting). Thus, they show some design limitations and constraints while using them to build a networking infrastructure as in our case. As a

result, the implementation of the multi-stage router in this scenario is more difficult because the internal software switch (named *vSwitch*) is not behaving as a standard Ethernet switch mainly due to security and isolation issues. Indeed, vSwitch only implements a crippled backward address learning mechanism (as described in detail later), which is not the standard one that is fundamental to guarantee the normal operation of our architecture.

More precisely, the vSwitch has two operational modes:

1. promiscuous mode on (hub): broadcasting the packets to every port except the receiving one.

2. promiscuous mode off (switch): allowing for at most one MAC address associated with a switch port. Thus, only the packets with this MAC destination are forwarded to the associated port, whereas packets with other destinations can not transverse this port, i.e, the packets will be dropped.

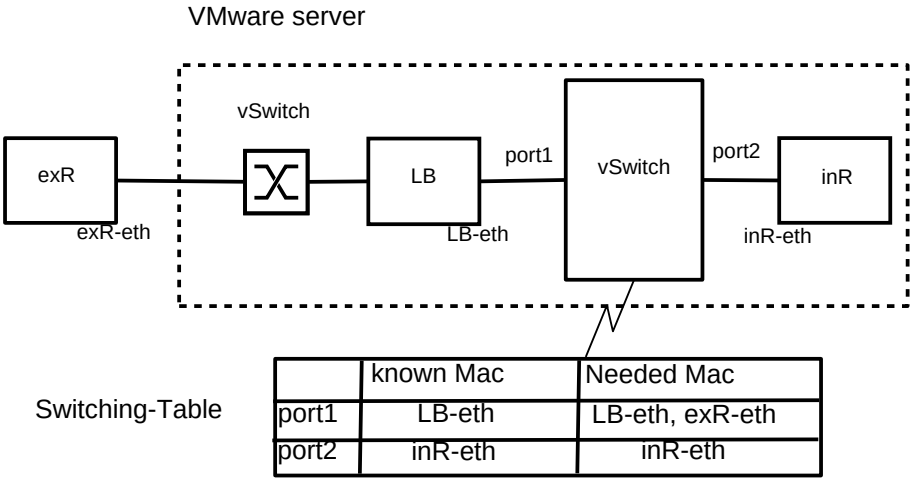


Figure 3.1. VMware Internal Switch limitation: the vSwitch does not support the Ethernet backward-learning mechanism. The internal communication among LBs and BRs is not working properly.

From the performance point of view, the first operational mode is not well suited mainly because of the excessive packet broadcasting mechanism which would disrupt the load balancing functionality. The second solution has to face the limitations in the vSwitch that do not permit to support LBs standard operation, as shown in Fig. 3.1. Indeed, LBs receive from BRs the packets which are not addressed to their internal MAC addresses but to the MAC addresses of the external devices. Unfortunately, all packets with MAC destination addresses different from the LB MAC address are discarded by the vSwitch (in *switch* mode).

Since it is not possible to configure the vSwitch to behave as a standard Ethernet switch, we define a workaround to make the system work in a proper way. Two possible solutions can be considered:

- hub config: use as many two-port hubs (promiscuous mode on) as the number of interconnections among LBs and BRs to create point-to-point links among stages. Every hub connects one LB and one BR only;
- VLAN config: use one vSwitch, as shown in Fig. 3.2, and configure one VLAN for each BR-LB pair.

Both solutions are equivalent from the functional point of view and they permit to implement a fully-functional MSR. Remember, in both cases an additional hub (or VLAN) (e.g. VLAN m in Fig. 3.2) is needed to interconnect all LBs and BRs with a full-mesh network to allow the normal operation of the DIST control plane.

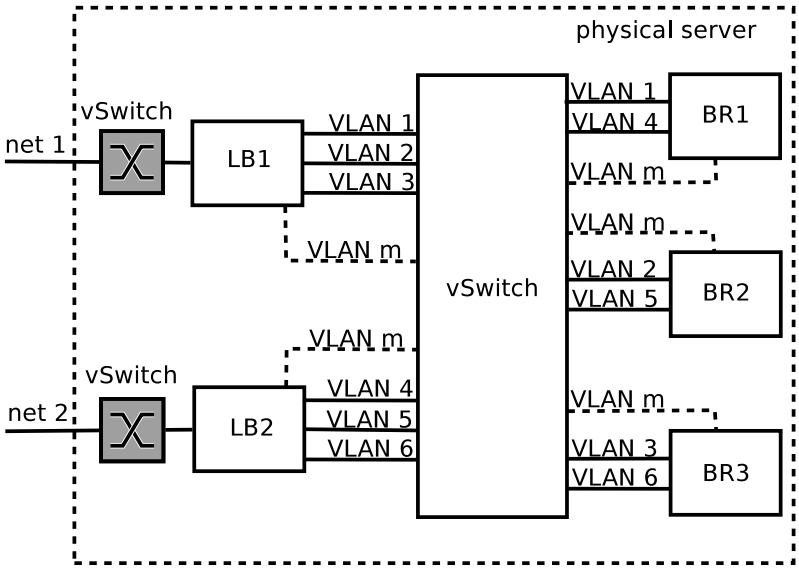


Figure 3.2. Internal configuration of a VMware-based multi-stage router to limit packet flooding: solution based on per-port VLAN tagging as supported by VMware vSwitch.

3.3 Experimental Setup and Results

In this section we implement the virtual MSR and test its performance in different scenarios. All the experiments on a single physical server were run on a Dell Power Edge T100 PC equipped with an Intel Xeon E3110 running at 3.0 GHz with

`vmx` hardware support for virtualization, 2 cores, 8 GB DDR2 RAM and 2 Intel PRO/1000 dual-ports network interface cards (NIC) with 82571EB Gigabit Ethernet Controller and are inserted in PCIe x4 slots. The driver versions are 7.3.21-k3 with the NAPI patch in the router tests and 7.3.20-k2 with or without NAPI patch in the LB tests [1]. When two physical servers are involved, an additional SuperMicro C2SBX server is used. It is similar with the DELL server except for the CPU (Intel Core 2 Duo E6750 @ 2.66 GHz). The chosen hypervisors are XEN 3.3 and VMware ESXi 4.0. VMs run Ubuntu 9.04 with Linux kernel 2.6.28-11-generic. Traffic is generated and received by an Agilent N2X RouterTester [47], the Gigabit Ethernet modules have been used and they have the capability to generate and receive Ethernet frames of any size in line rate. Graphs report the average of five independent runs; the performance difference among the five tests is negligible because the tests provided by the measurement tool are well-designed, precise and long enough to obtain stable results. When a single stage is tested, we use one vSwitch to directly connect VMs to one physical NIC; in the case of the multi-stage architecture, a more complex configuration scheme is needed, as described in Sec. 3.2.

Results related to XEN are not reported because the *bridged* configuration exhibits very poor performance, as also reported in [23]. For instance, our experiments show that in a 64 bytes packets scenario, the XEN DomU is only able to forward at most 150 kpps, and in deep overload scenarios (e.g. input load larger than 1.2 Mpps) throughput drops to 10 kpps (roughly 5 Mbps). Similar results hold when increasing the number of virtual routers in DomU. Since the throughput provided by XEN-based VMs is really poor, it is not possible today to build a virtual MSR with reasonable forwarding rate. Thus, we focus on VMware based solution only. All the results refer to VMware ESXi 4.0, which uses a hybrid approach of binary translation and hardware assisted virtualization [48] to achieve better performance.

Graphs also reports the routing performance of the physical machine running Linux (1 or 2 active CPU cores, identified by the *Phy* prefix) to provide an upper bound to the VM forwarding performance. Preliminary tests not reported in this thesis show that VMware performs better when both CPU cores are active. Thus, all VMware ESXi tests refer to the 2-cores scenario. We first focus on performance of LBs and BRs in isolation, then the whole virtualized multi-stage architecture is examined.

3.3.1 Load Balancers

VMware ESXi exports to the guest OS different virtual network interfaces. We consider here only the VMXNET and the Intel e1000 emulation due to the availability of the performance enhancing Click patch for e1000 driver. The VMXNET

[1] The slightly difference of the driver is due to the lack of patch in Click Modular Router

is a custom VMware network interface based on para-virtualization, thus additional drivers are needed in the VMs, meanwhile in the second case a virtual Intel e1000 hardware is emulated and the standard Linux driver is used (in some cases with the addition of the e1000-related Click NAPI patch). Because of para-virtualization, we expect better performance from the VMXNET based solutions.

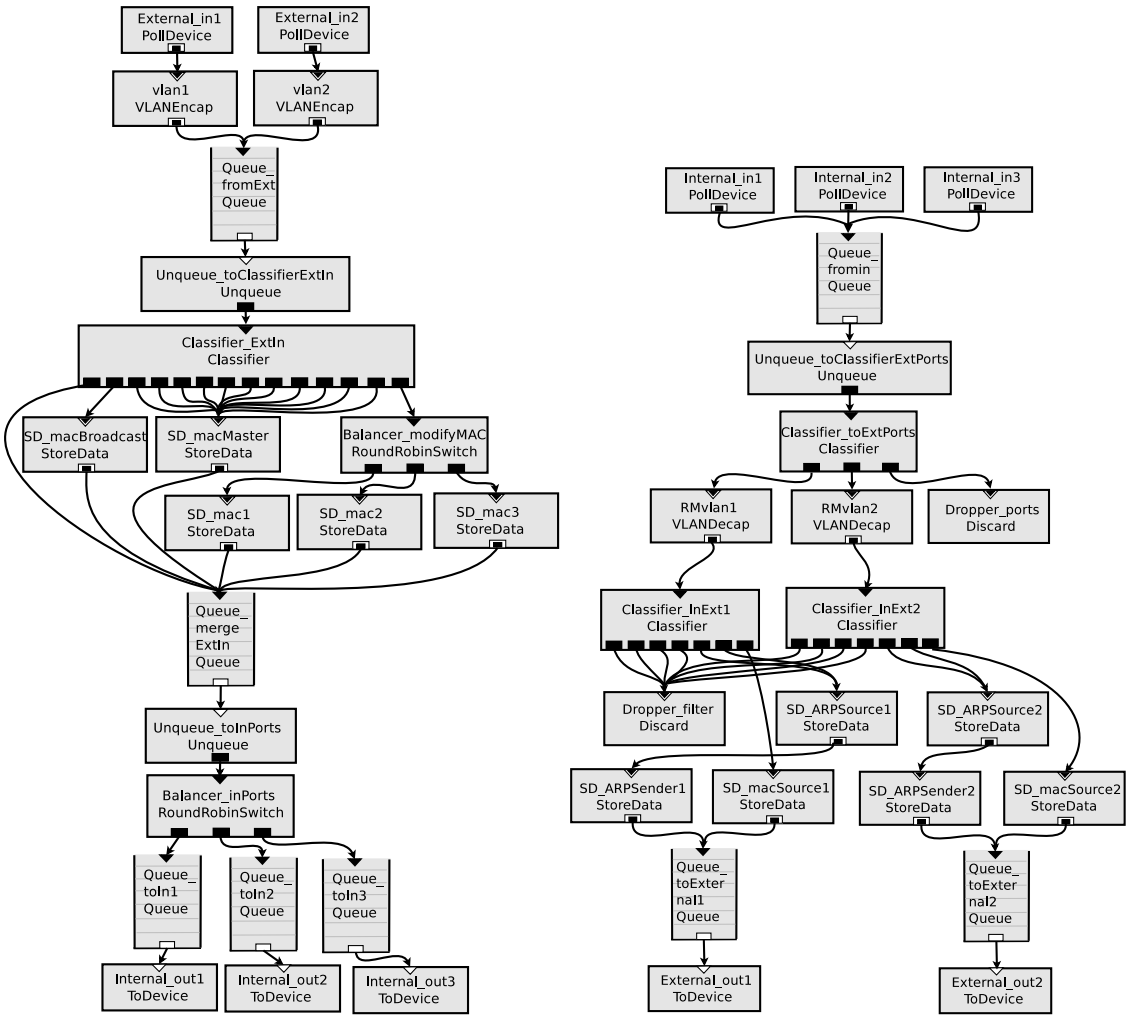


Figure 3.3. Click based load balancer configuration: 3.3(a) from external to internal and 3.3(b) from internal to external.

Figs. 3.3(a) and 3.3(b) show the internal Click configurations of our LB implementation, the number of internal and external interfaces can vary depending on the need. Interested readers may refer to [2] and [49] for more details about the

different blocks functionalities and Click diagram. To make a fully comprehensive evaluation of the LB performance inside VMs, we also tested our Click LB implementation on a standard Linux distribution running on the same hardware without any virtualization infrastructure.

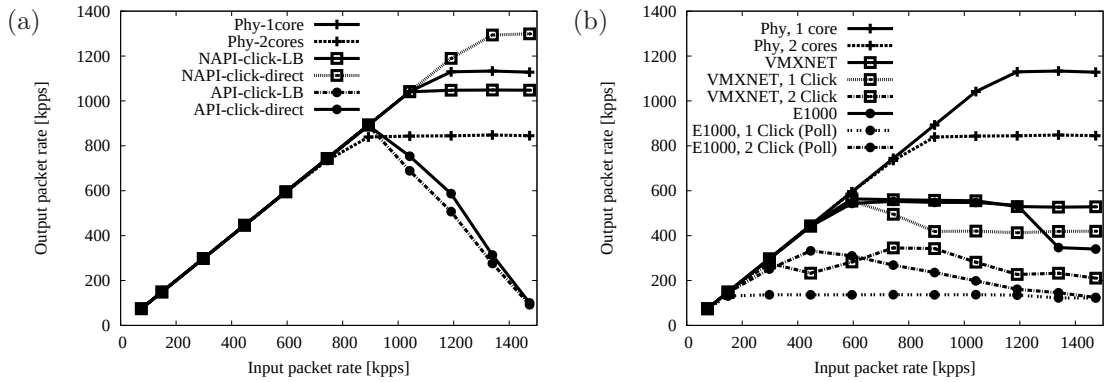


Figure 3.4. Performance evaluation of Click-based load balancer in a Physical server 3.4(a) and VMware ESXi 4.0 environment 3.4(b) using 64 bytes packets.

In Fig. 3.4(a) we report LB performance in the scenario without virtualization. We compare a standard OSR running in Linux (named *Phy*), a Click configuration directly connecting the input and output interface without any packet processing (named *Click-direct*) and the Click-based LB described in Fig. 3.3 (named *Click-LB*). In the *Phy* case either one or two CPU cores are activated, with the well-known results of bad performance when more cores are forwarding packets, as explained in [8]). In the Click-based experiments we test the basic networking API (e.g. no interface polling) and the new-API (NAPI) based interface polling. Results are consistent with [50]: NAPI is more stable than API under heavy load due to the interrupt mitigation by polling and the Click-direct is faster than Linux routing because of its very basic operations, i.e. packets are moved directly from the input port to the output port without any forwarding table lookup and without any filtering or modification. Furthermore, results show that our LB configuration can achieve performance around 1 Mpps with minimum sized packet. The cost of queueing, packet classification and header re-writing compared to the Click-direct is approximately a throughput reduction of 20% in the worst-case high-load scenario.

We report in Fig. 3.4(b) a comparison among different virtual network drivers when running LBs in VMs and we also include the routing performance of the chosen drivers as references, namely *VMXNET* and *E1000* respectively. One physical interface is used to connect the PC to the router tester, whereas VMs are connected to a single vSwitch directly connected to the physical interface.

Regardless of the chosen NIC driver, virtualization introduces a large overhead. In the best case of the VMXNET driver, throughput drops to about 600 kpps in 64 bytes scenario, almost half of the reference physical server routing capability. Regarding the Click LB throughput under VMXNET driver, we could inspect fluctuation and forwarding rate degradation in the high load case since no Click patch is available for such proprietary drivers. Furthermore, running multiple Click LBs which share the same physical resources introduces additional overheads. For instance, in the case of 2 VMXNET-based LBs, throughput drops to approximately 250 kpps due to context switching and resource contention (cache misses and interrupts management) among different VMs. Indeed, the cost of context switching is higher when more VMs are sharing the same resources, because more VMs contend for the same resource and the VM state has to be restored each time before execution starts. Thus, sharing physical resources among different virtual LBs significantly limits the performance.

Finally, the emulated Intel e1000 NIC obtains less throughput than para-virtualized VMXNET NIC because of the more computationally intensive operations based on hardware emulation. Even when using the polling patch (NAPI-aware) for the driver, throughput is still unsatisfactory. An interesting result is that resource sharing is beneficial in this case, since 2 VMs running Click-based NAPI-aware LB are able to obtain a higher aggregated throughput than a single VM. This is a clear indication that the Click-based NAPI-aware LB running on top of an emulated Intel e1000 hardware is not able to efficiently exploit the hardware resources because of the complex interaction among the emulated NIC and the drivers based on the NAPI polling mechanism [2].

3.3.2 Back-end Routers

The experiments in this section use the same internal configurations as described in the previous one, but instead of the Click-based LBs, we test the VMware ESXi routing performance. Only the VMXNET driver is considered due to the higher throughput. Results are reported in Figs. 3.5(a) and 3.5(b). The virtualization overhead is significant in the routing case too: the aggregated throughput is roughly 600 kpps when a single VM is hosted and 250 kpps when 4 VMs forward packets concurrently. Resource sharing among BRs also negatively impacts the throughput. This phenomenon becomes stronger when increasing the number of active VMs concurrently performing routing operations on the same server, because more CPU cycles are wasted to solve the contention problem. However, the performance drop in the routing case is less than in the LB case. Indeed, the aggregated forwarding

[2]This result can also be verified through the host CPU load profiling, i.e, only one VM with e1000 Click NAPI driver can not consume all the CPU resource.

rate of the LBs is already 250 kpps when 2 LBs are present in the same server and even worse for 4 concurrently running LBs. Resource sharing has no impact if the other VMs are not routing packets, as reported in Fig. 3.5(b). The results show that the impact of resource sharing depends on the VM activity level. Indeed the throughput of a single active VM does not change when increasing the number of idle VMs, i.e, VMs are powered on but they are not used as routing engines.

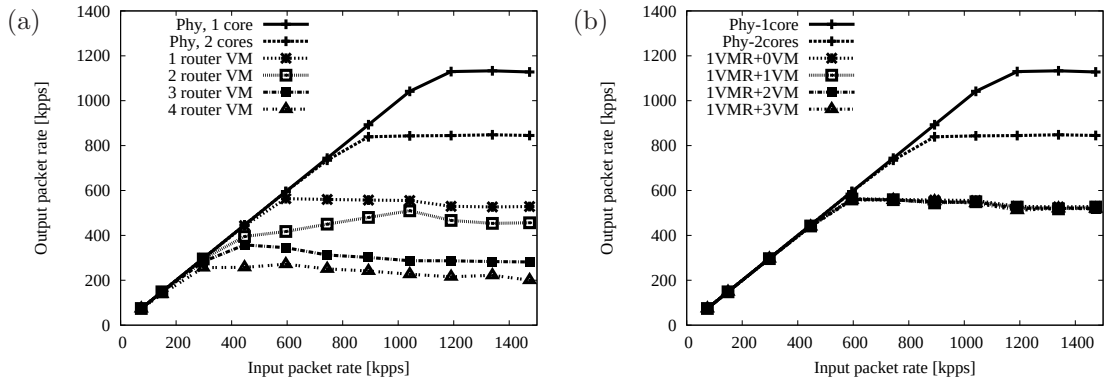


Figure 3.5. Performance evaluation of OSR in the VMware ESXi 4.0 environment using VMware's VMXNET driver and 64 bytes packets. All VM are routing packets concurrently 3.5(a) and only 1 active VM is routing packets 3.5(b)

Besides throughput measurements, we focus on fairness issues generating two flows to two VMs with different load distributions. The VMs perform the standard Linux routing functionalities. We report the results in Tab. 3.1, where we show fairness measurements using two flows: first in two equal load scenarios (e.g. 50% and 100% of the maximum input load per flow, i.e, 1Gbps) and then with different load unbalance between the two flows (i.e, 0.5-0.5, 0.3-0.7 and 0.1-0.9). For instance, when the input load is 50% and flow 1 (flow 2) is 0.3 (0.7), the bandwidth of the input flow 1 (flow 2) is 150 Mbps (350 Mbps). If the output load is 42.1% (97.5%) for flow 1 (flow 2), only 63.15 Mbps (341.25 Mbps) have been received on the output. Clearly, regardless of the aggregated input load, the low load flow exhibits increasingly smaller throughput for increasing load. VMware ESXi is unable to well isolate flows, significantly penalizing low load flows.

3.3.3 Local Testbed for a Multi-Stage Software Router

We build and test a virtualized multi-stage software router architecture considering three different scenarios:

1. one physical server, one LB and one BR;

Total input load: 50%				Total input load: 100%			
flow 1		flow 2		flow 1		flow 2	
in	out	in	out	in	out	in	out
0.5	68.3%	0.5	64.4%	0.5	37.6%	0.5	39.2%
0.3	42.1%	0.7	97.5%	0.3	17.9%	0.7	47.9%
0.1	75.0%	0.9	97.5%	0.1	13.3%	0.9	42.9%

Table 3.1. Fairness tests for OSR on VMware ESXi: throughput of two flows with 64 bytes when varying the relative flow load.

2. one physical server, two LBs and two BRs;

3. two physical servers, one LB and one BR per server.

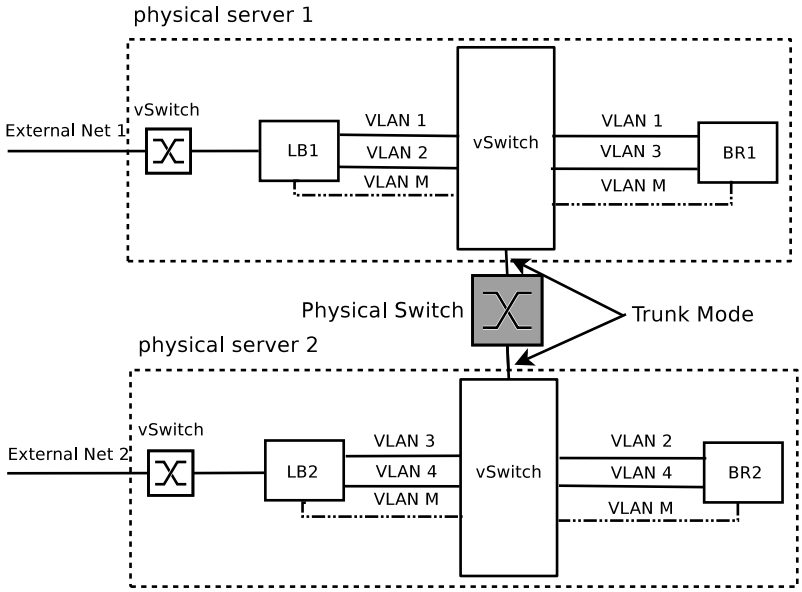
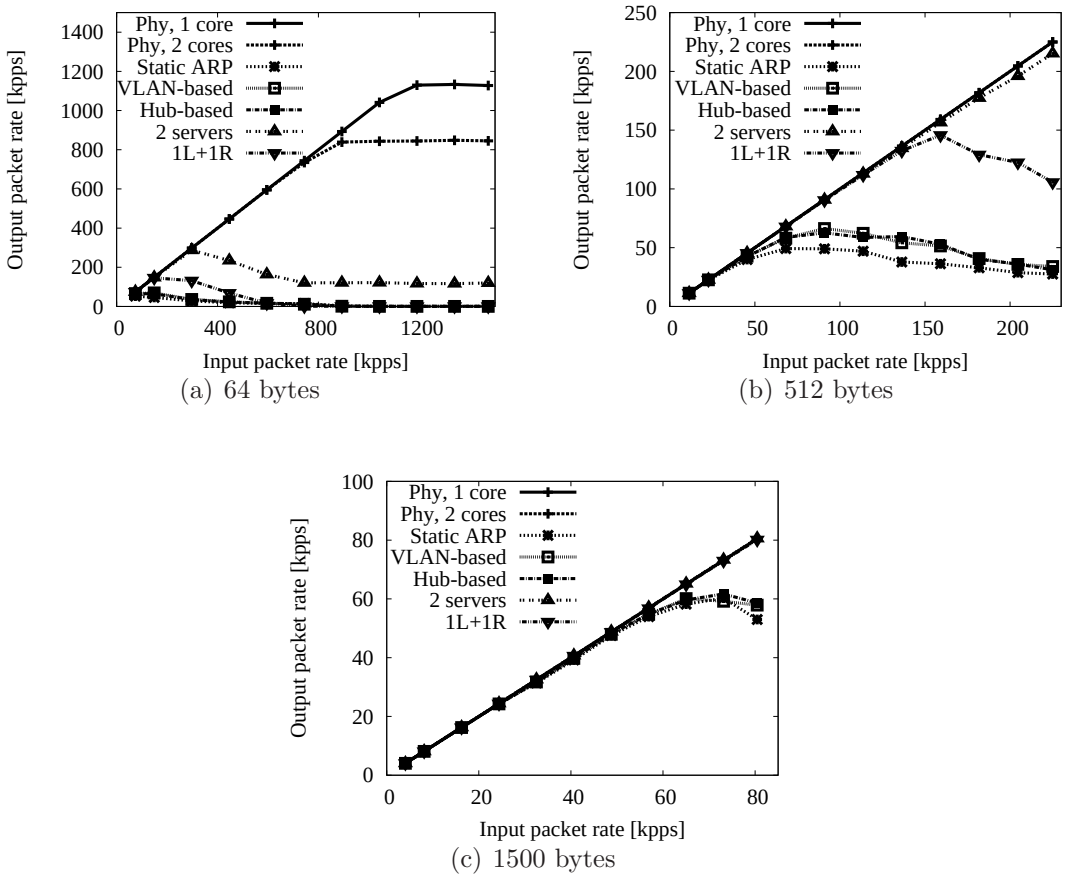


Figure 3.6. The implementation of the multi-stage software routers in the two physical servers scenario.

In the first scenario, which represents the minimal configuration, we consider the hub configuration only for the internal vSwitch because complex configurations are not needed. In the second scenario we compare three different internal configurations: the VLAN and hub based solutions, as described in Sec. 3.2, and a *static ARP* solution consisting of a simple ARP table modification needed in BRs to use vSwitch in *switch* mode. This solution provides correct performance indications, but it causes wrong packet addressing at the MAC level thus it must not be used in

a working solution. Finally, in the third scenario, we consider performance scaling with additionally deployed resources. In this case we use the VLAN-based solution only, as described in Fig. 3.6.

Results are reported in Fig. 3.7. Throughput is rather poor for small packet size. This is expected due to the performance limitations of single element induced by virtualization and resource sharing (e.g. frequent interrupts to run different VMs, context switching and execution status restoration). Indeed in the first scenario (one LB and one BR) we obtain good performance for large packets only (almost wire speed when approaching the maximum packet size). In the second scenario, where more elements (two LBs and two BRs) share the same resources, the CPU becomes the bottleneck and the upper bound for performance becomes around 70 kpps for large packet size, not enough to reach wire speed.



(a) 64 bytes

(b) 512 bytes

(c) 1500 bytes

Figure 3.7. Performance evaluation of multi-stage router (2 LBs + 2 BRs) in the VMware ESXi 4.0 environment with different internal configurations. From left to right: 64, 512 and 1500 bytes packets.

Running the multi-stage in two physical servers, as in the third scenario, leads to performance improvements. The throughput is still unsatisfactory for 64 bytes packets, but it permits to reach wire-speed from approximately 512 bytes packet size onwards. This is due to the larger amount of deployed resources and to reduced contention, as in the first scenario.

Finally, no major performance differences can be observed among the various internal networking configurations. Thus, the utilization of hubs or VLAN tagging functionalities does not influence performance in the studied scenario, where the bottleneck is represented by the overloaded CPU.

These measurements prove the feasibility of the architecture on the local infrastructure with multiple servers. The performance results show that single server-based solutions obtain limited throughput, but the capacity can scale up easily by integrating into the multi-stage router virtual resources hosted on different servers.

3.3.4 FEDERICA-slice Based Experiments

The introduction of virtualization technologies permits to dynamically add new components borrowed either from the local data-center or from external computing resource providers, i.e., the cloud.

To assess the feasibility of this approach, we created an instance of the multi-stage router running on top of the FP7 FEDERICA infrastructure [36], which is an European-wide network dedicated to network researchers. As shown in Fig. 3.8, the MSR is composed of 3 LB VMs (hosted in Poland, Czech Republic and Germany) and 3 BR VMs connected by a physical switch (all in Poland). The connectivity among PoP is guaranteed by VLANs on top of GÉANT dedicated links. The multi-stage router is connected to 3 external host VMs in the same PoPs of the LBs to generate traffic and to collect statistics.

To measure throughput performance, we use iperf [51] to generate multiple UDP flows to overcome software traffic generator's speed limitation and to create enough traffic to overwhelm the multi-stage router. To estimate latency we exploit a limited bit rate ICMP traffic flow. Several traffic patterns were tested on the FEDERICA slice. Due to space limitation, we report results only for a uniform 3x3 traffic pattern. We repeated the measurements 20 times for each different packet size, for experiments lasting one week. Fig. 3.9 shows the averaged throughput and delay per source/sink pair. We include also the standard deviation values for latency and the minimum/maximum values for throughput, being the standard deviation negligible. Results in Fig. 3.9(a) show limited throughput only for small packet size. Note that the maximum throughput of the FEDERICA slice is around 350 Mbps, not reaching the line rate even for 1500 bytes packet size. This is due to the fact that the MSR FEDERICA slice shares the same physical infrastructure with other simultaneously running FEDERICA experiments. Throughput is affected by other

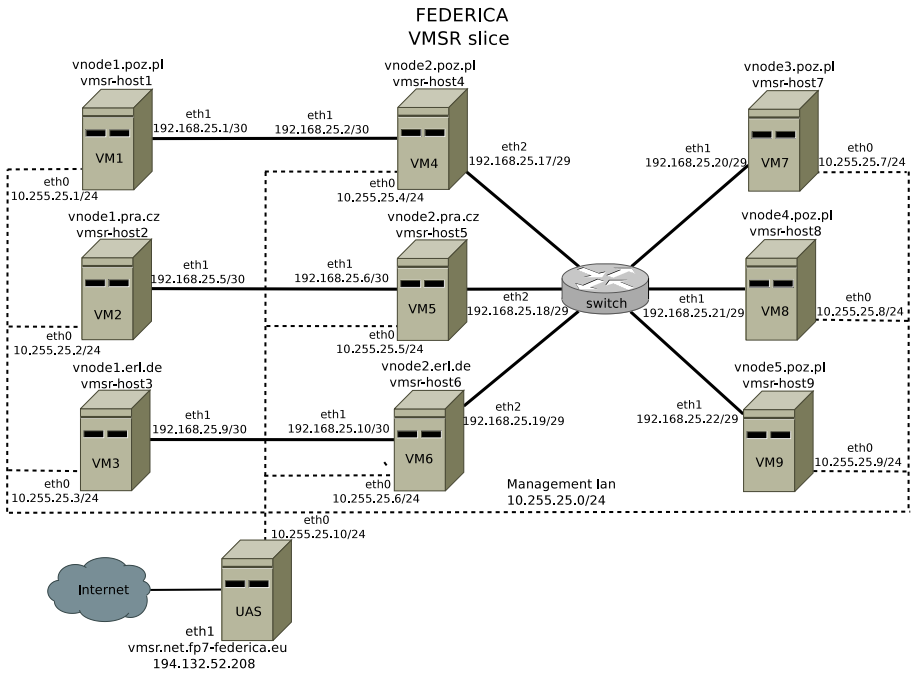


Figure 3.8. A slice of the FEDERICA infrastructure and the corresponding multi-stage software routers implementation.

running experiments sharing the same hardware. Furthermore, we can observe a slight variability in the throughput, due to the packet loss when transmitting packets across a long distance over different countries. Looking at the latency in Fig. 3.9(b), overloaded and underload regions are clearly visible. When the system is overloaded we obtain almost constant delay curves for different packet size flows. The average delay value varies widely among different packet sizes, from 200ms for 64 bytes to 1600ms for 1500 bytes. The reason is that with 64 bytes flows, packets have been dropped significantly due to the lack of processing power, then the actual load on the network is less compare to 1500 bytes flows. Hence, the ICMP flow obtains lower delays in the low network load (i.e, 64 bytes flow) scenario than in the high network load (i.e, 1500 bytes flow) scenario.

Performance results for the MSR running on top of the FEDERICA infrastructure are coherent with those of the local testbed measurements. This shows that it is possible i) to build a MSR using VMs running in different locations and ii) to integrate the local forwarding capabilities with additional externally rented forwarding capacity.

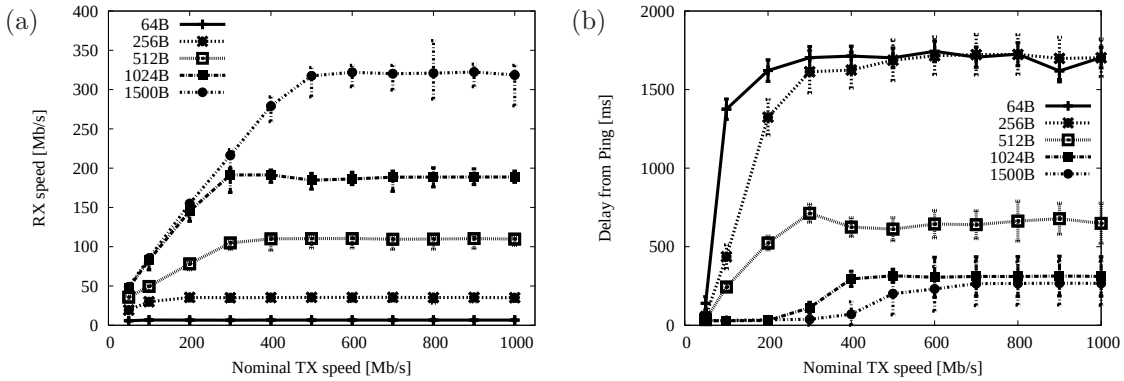


Figure 3.9. The FEDERICA-slice based multi-stage software routers packet receiving rate with minimum and maximum value 3.9(a) and packet delay with standard deviation 3.9(b) under full mesh traffic scenario.

3.4 How to Improve MSR's Performance

The measurements reported in previous sections assess the feasibility of the virtualized multi-stage router architecture in LAN and WAN. Our tests highlight performance issues due to software limitations and resource sharing. In the next future, the natural evolution of hardware and software solutions will help to solve the inefficiency issues by adding more capacity and by introducing improved virtualization infrastructures. Instead, the resource sharing issue is more interesting from an architectural point of view. In this section, we propose two techniques to alleviate the resource sharing and contention problems.

3.4.1 Mapping of VMs to Physical Servers

We experimentally investigate the problem of mapping VMs to the available servers. This problem consists of selecting the best placement of a set of VMs (e.g. components of the multi-stage router) over a set of physical resources (e.g. computing servers) and of configuring the network among the physical and virtual components (e.g. VLAN configuration). Because of the complete decoupling among logical functions and physical resources, every VM can be hosted on every server, but some constraints and guidelines must be considered:

- Network bottlenecks: the network among physical servers may become a bottleneck when the bandwidth required among the virtual components on different servers is larger than the physical bandwidth available on the physical interconnection network (i.e, middle stage);

- Resource sharing issues: according to measurements reported in previous sections, resource sharing must be minimized to get maximum throughput, especially in the case of LBs.

We present a set of measurements designed to highlight these problems, showing the impact of resource sharing and physical bottlenecks on our architecture. We consider the case of a multi-stage router composed of two LBs and two BRs deployed on a physical infrastructure composed by two physical servers (described in Sec. 3.3) and we test several deployments of the multi-stage router over the two physical server (e.g. 2 LBs on the first server and 2 BRs on the second server, 1 LB and 1 BR on both servers, etc.). Furthermore, we add some baseline measurements to assess the behavior of LBs or BRs in isolation.

The measurement scenarios reported in Tab. 3.2 are:

A) Single server: a single server hosts 1 LB and 1 BR. In the first case 1 external interface is used (A.1), in the second case 2 external interfaces are used (A.2);

B) Balanced mapping: every server hosts 1 LB, 1 BR and 1 external interface (Fig. 3.6). A single link connects the servers and two different balancing schemes are used inside the LBs: standard Round-Robin (B.1) and local-only (B.2), where all incoming traffics are sent to the BR located on the same physical server (the LB configuration is much simpler);

C) Logical separation: 2 LBs are on a server (using 2 external links), 2 BRs on the other server. The servers are connected by a single link to introduce a physical bottleneck (C.1, Fig. 3.10(a)) or by two links to avoid the bottleneck (C.2);

D) Unbalanced mapping: a server runs 1 LB and 2 BRs (D.1, Fig. 3.10(b)) or 2 LBs and 1 BR (D.2), the remaining components are on the other server. A single link connects the server without introducing bottlenecks.

Scenario	Servers	Mapping		Additional
		server 1	server 2	notes
A.1	1	1 LB, 1 BR	–	1 ext. NIC
A.2	1	1 LB, 1 BR	–	2 ext. NICs
B.1	2	1 LB, 1 BR	1 LB, 1 BR	Round robin
B.2	2	1 LB, 1 BR	1 LB, 1 BR	local only
C.1	2	2 LBs	2 BRs	bottleneck
C.2	2	2 LBs	2 R	no bottleneck
D.1	2	1 LB, 2 BRs	1 LB	–
D.2	2	2 LBs, 1 BR	1 BR	–

Table 3.2. Description of the test scenarios for VM mappings

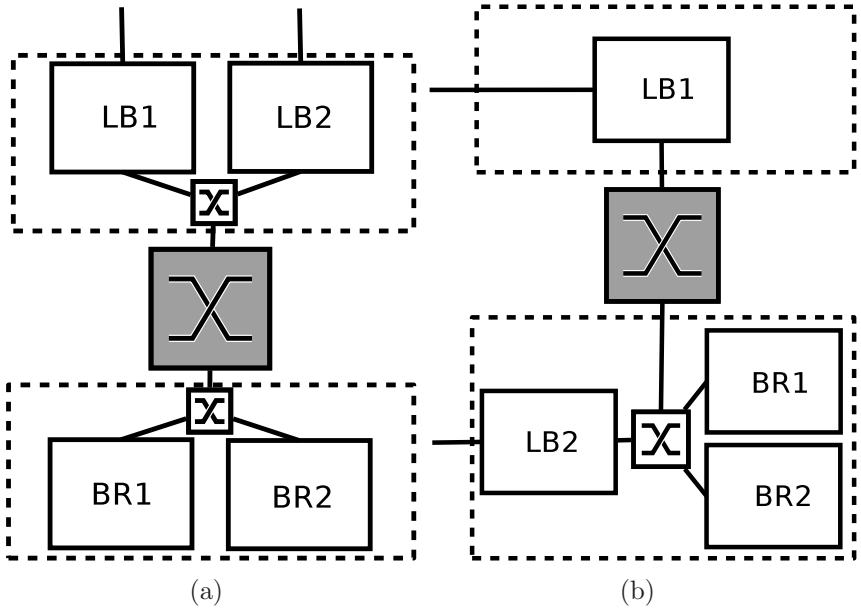


Figure 3.10. Logical separation among servers. (a) Scenario C.1: A physical bottleneck among the servers and (b) Scenario D.1: No physical bottleneck among the servers

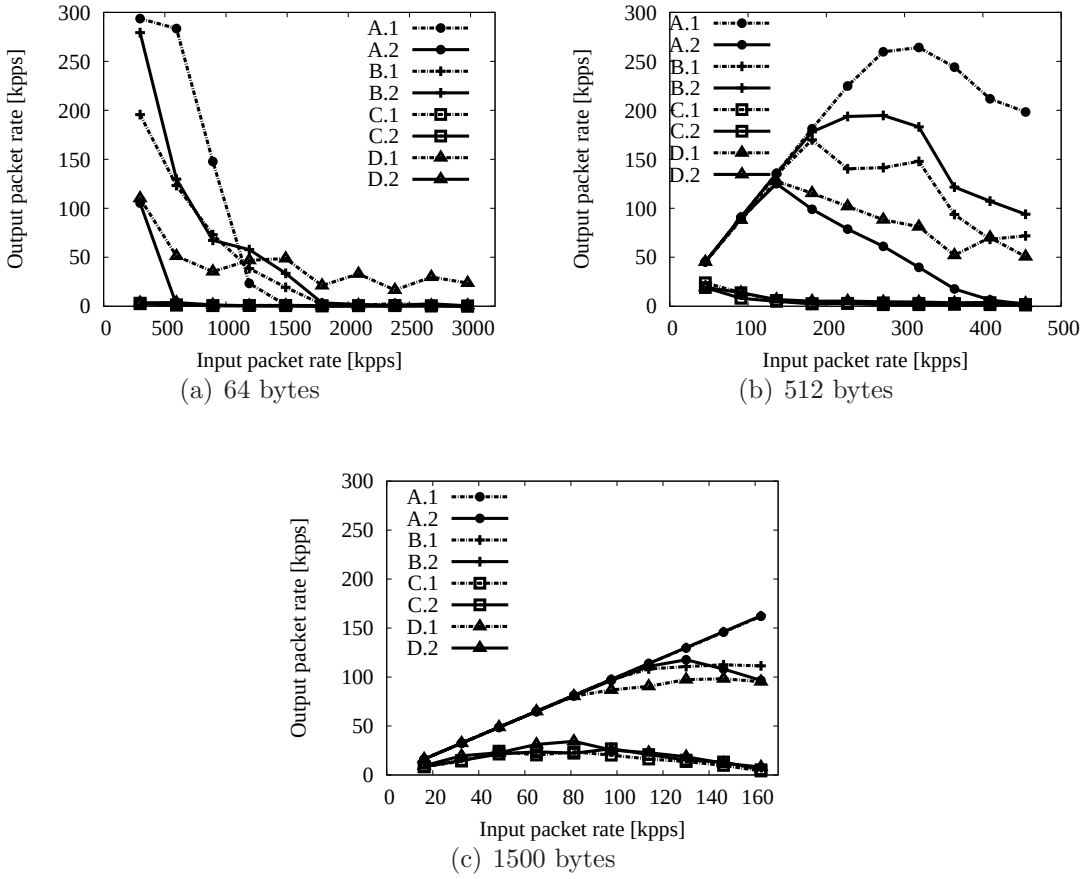
Results are reported in Fig. 3.11 where we present throughput measurements using different packet sizes (e.g. 64, 512 and 1500 bytes). The main results are discussed below.

A.1 and A.2: The throughput gap between A.1 and A.2 is huge, regardless of packet size, even if the configurations are similar. A.1 is a one port router while A.2 is a 2 port router from the external point of view, which results in a double traffic load for A.2 and lower throughput than A.1 because the system is much more overloaded and interrupt management cost becomes dominant. The difference is smaller for small packet size because of excessive interrupts. Furthermore, A.2 achieves good results only if the packet size is large.

B.1 and B.2: B.1 and B.2 are based on the same topology, but on two different balancing schemes. B.1 exploits the standard round robin mechanism, meanwhile B.2 uses a simplified LB which sends incoming packets only to the local routing VM, which translates into less Click elements (e.g. queues, schedulers, packet modifiers) and simpler policies, introducing less per packet overhead. Indeed B.2 outperforms B.1 for all packet sizes, though the difference is negligible when receiving small packets because both systems are overwhelmed by excessive input load. Our results suggest that the optimization of Click based LB implementation is critical to improve the overall performance.

C.1 and C.2: The performance are very similar regardless of the physical bottleneck in C.1. Due to low LB performance, two LBs on the same hardware obtain much smaller aggregated throughput than the bottleneck (Sec. 3.3.1). Under this condition, C.1 and C.2 are equivalent and they achieve similar throughput.

D.1 and D.2: D.1 provides better performance than D.2, as in the previous case. D.1 is the best with small packets and large load, because it is possible to exploit at best hardware resources when the LB is alone. D.1's throughput suffers from larger packets because of the resource unbalancement which is limiting the performance of the server hosting three components.



(a) 64 bytes

(b) 512 bytes

(c) 1500 bytes

Figure 3.11. Virtual functional units optimal location mapping tests inside VMware ESXi 4.0: left is under 64 byte packet size, middle is under 512 byte packet size and right is under 1500 byte packet size

In fact, the excessive hardware interrupts dominates performance when the packet size is small while additional internal traffic overheads are crucial when the packet size is large. The Click LB implementation is more resource hungry than

routing elements. Based on these observations, some rules of thumb should be followed when mapping a virtual multi-stage router to a physical infrastructure:

- Assignment of more physical resources to LBs.
- Isolation of virtual resource leads to better throughput regardless of packet size, especially for LBs.
- Bottlenecks among servers may sometime limit performance, thus smart *virtual-to-physical* mapping algorithms are needed to obtain good performance.

A knowledge about the features of the required service will be a key element in helping resource providers when mapping VMs to physical devices.

3.4.2 VMs CPU Affinity Exploration

We study the performance impact of mapping VMs virtual core to physical cores with different strategies to assess the best VMs CPU affinity settings for the multi-stage architecture. A similar approach was pursued in [52], where the authors test how to allocate input and output interfaces to CPU cores, while our experiments concentrate on the VM's CPU core allocation problems because the CPU computational power limits multi-stage architecture's performance.

As shown in Fig. 3.12, we consider 9 configurations inside a single server. Recall that all physical servers are dual-core machines. In the figures, HC means pHysical Core while VC means Virtual Core. The box with a dashed line stands for VMs. For instance, the top-right sub-figure *1VM-2v2p* means 1 VM with 2 virtual cores, and each core is mapped to a different physical core; while figure *4VM-1v2p* located in the bottom-right stands for 4 VMs with a single core each, and each core is mapped to 2 physical cores.

Two types of VMs are needed in the MSR architecture, LBs or BRs, the first or third stage elements. Fig. 3.13 reports results for the LB and the BR case in isolation: the multi-stage architecture is not involved. The tests are performed on one physical server only. We consider only the VMXNET driver because it outperforms the e1000 driver in all scenarios. In Fig. 3.13(a), it is easy to group the curves referring to one LB or two LBs, regardless of CPU affinity configuration. This means that hosting multiple LBs to share the same physical resource leads to a frantic performance reduction, as explained in previous sections. Focussing on the three lines corresponding to one LB only, observe that activating one VC and mapping it to two HCs instead to one HC slightly improves performance because *1VM-1v2p* exploits extra computational power. The same results can be observed in Fig. 3.13(b) when testing BRs. When activating two VCs in one LB, we obtain better throughput in low load scenario. However, in Fig. 3.13(b) we only observe

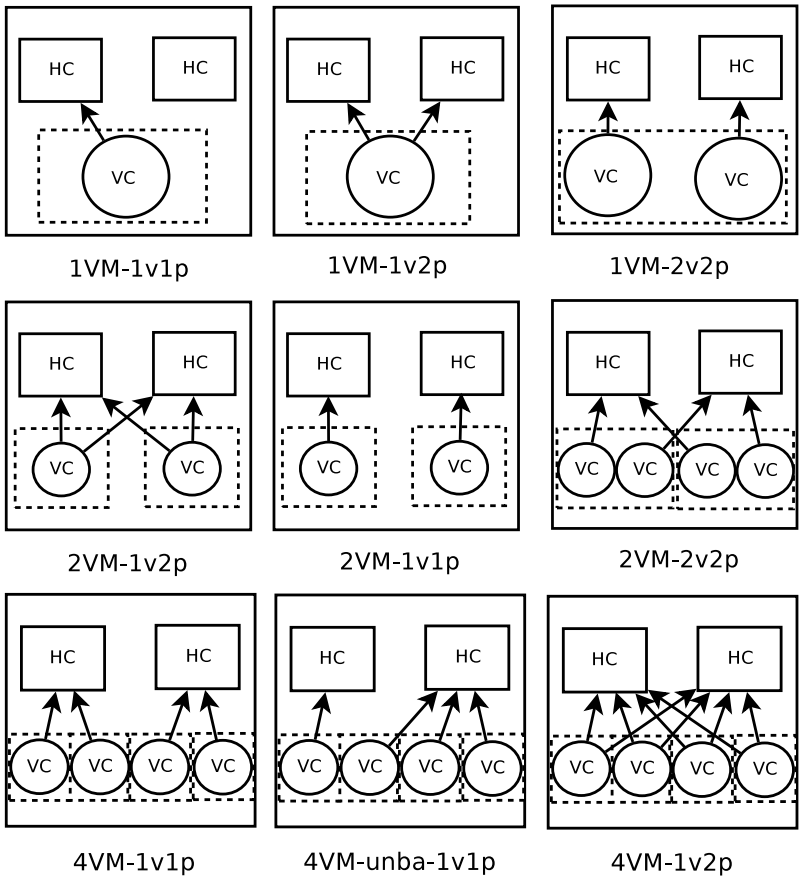


Figure 3.12. VMs CPU affinity test under VMware EXSi 4.0: the demonstration of different CPU mapping schemas.

throughput degradation when activating two VCs irrespective of the input load. The reason for the dropped routing performance is again cache misses as explained before. For the LB, indeed, the Click kernel driver substitutes the Linux networking stack and different elements are mapped to different cores: thus, no cache miss can happen with proper element association and higher throughput should be expected. As the input load increases, the reception element, i.e, PollDevice, can only access half of the physical resource. Thus, more packets are lost at the reception side, worsening throughput in the high load scenarios.

Results related to the two VMs are similar to those with one VM if activating two VCs in each VM. If multiple VMs with one core each are available, a 50kpps throughput improvement can be observed at maximum load when mapping each core to a dedicated physical core. Isolation provides less resource contention and higher throughput.

We report in Fig. 3.14 the affinity tests for the whole multi-stage architecture,

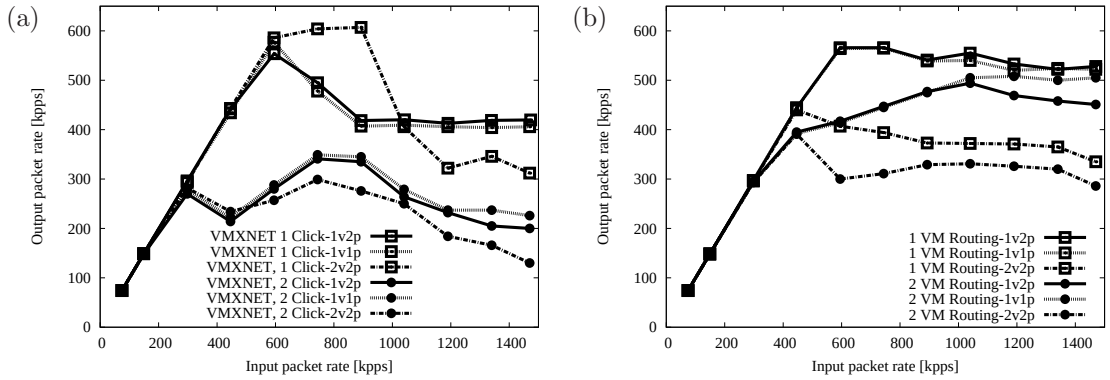


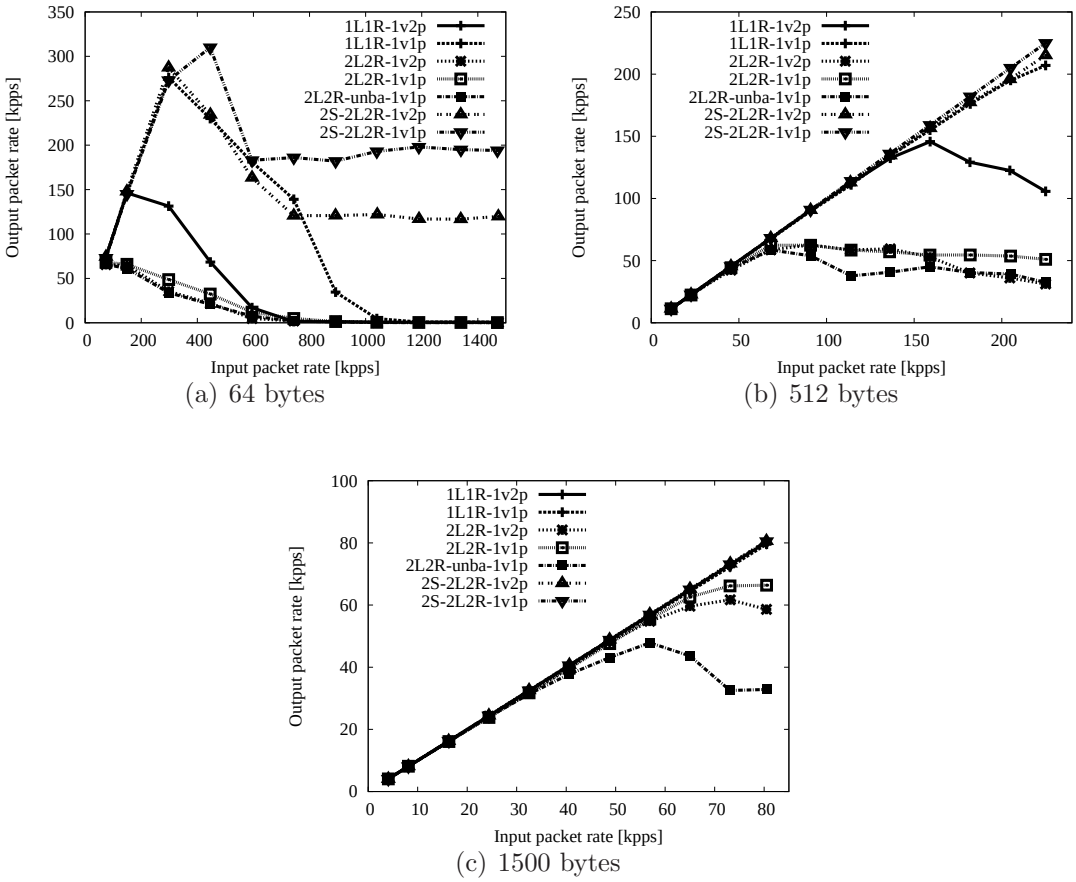
Figure 3.13. VMs cpu affinity tests under VMware EXSi 4.0 for LB 3.13(a) and BR 3.13(b).

with different internal configurations and packet sizes. *1L1R* means that there are 2 VMs in one physical server and *2L2R* means 4 VMs in one server, whereas *2S-2L2R* stands for 2 physical servers in total, each hosting 1 pair of BR and LB (2 VMs). The main results are:

- **1L1R**: Better throughput is obtained when mapping each VC to a dedicated HC compared with mapping each VC to two HCs. Furthermore, the *1L1R-1v1p* configuration reaches line rate for 512 byte packet size. Our results indicate that separating physical cores for different VMs can dramatically improve performance.

- **2L2R**: Three different configurations are available, as shown in the three mapping scenarios at the bottom of Fig. 3.12. The reason for an unbalanced mapping is that we want to separate one resource hungry LB from other elements to see if it is possible to improve the results. Unfortunately, this separation does not increase the throughput but rather exhibits negative impact on the whole architecture, due to more elements mapped to one HC. The results related to *1v1p* and *1v2p* when more VMs exist still hold: isolated physical resource control always obtains better throughput performance compared to resource sharing among all VMs.

- **2S-2L2R**: We increase the physical resource and consider two scenarios: sharing the physical core (*2S-2L2R-1v2p*) or assigning each VM to a dedicated core (*2S-2L2R-1v1p*). In both scenarios line rate is reached for 512 byte packets, due to the deployment of more hardware resource. In Fig. 3.14(a), the configuration *2S-2L2R-1v2p* routes 150kpps of 64 bytes, but *2S-2L2R-1v1p* reaches 200kpps, a 25% throughput increase can be obtained by carefully allocating VMs to physical CPU cores.



(a) 64 bytes

(b) 512 bytes

(c) 1500 bytes

Figure 3.14. Multi-stage software routers virtual component affinity configurations against physical cpu cores experiments inside VMware ESXi 4.0.

In summary, to improve the virtual MSR's performance, it is very important to assign the minimum number of virtual elements to each physical core. However, if the number of VMs is less than the number of physical CPU cores, it is better to consider a mapping which exploits all physical resources to take the full power of the system.

3.5 Conclusions and Future Work

In this chapter we demonstrated the feasibility of a multi-stage router architecture in a virtualized environment. We highlighted strong performance limitations that makes this approach rather difficult to pursue today. To improve performance, different mapping techniques have been tested and we showed that by carefully

assigning VMs to servers and assigning virtual cores to physical cores, better performance can be achieved. Research work is needed in many areas to further improve this kind of approach. The main areas of interest are as follows;

- hypervisors: reduction of overhead costs and optimization of NIC virtualization, solving isolation/fairness issues (mainly on CPU and NICs sharing);
- MSR internal elements allocation: optimization of VMs allocation on different physical servers and VMs cpu bindings to balance resource sharing to improve performance;
- virtualization approaches: selection of less invasive virtualization approaches like operating-system level virtualization (e.g. OpenVZ and Linux Vserver) and minimized OS images for VMs. An interesting approach to be considered is Denali OS [53], where hypervisor and OS are designed from scratch to tightly collaborate to reduce virtualization overhead.

Nevertheless, considering the natural evolution of PCs in many areas (e.g. CPU speed, multi-queue support in NICs, networking stack), and the optimization techniques proposed in this chapter, we are confident that this approach will obtain reasonable performance in the near future, increasing the interest on adopting these solutions also in production networks.

Chapter 4

Tuning KVM to Enhance Virtual Routing Performance

In this chapter we show how to use an open virtualization architecture to analyze and improve the forwarding performance of a virtual router. In particular, the forwarding performance of the Linux kernel running inside a KVM virtual machine and the performance of some more advanced architectures based on virtual routers aggregation are analyzed, showing how increasing the number of used CPU core can improve performance and how properly setting the CPU affinity of the various virtualization activities affects virtual router throughput.

4.1 Introduction

In the last decade, Software Routers (SRs), i.e, routers running in commodity Personal Computers (PCs), become an appealing solution compared to traditional Proprietary Routing Devices (PRD) for various reasons such as cost (the multi-vendor hardware used by SRs can be cheap, while the equipment needed by PRDs is more expensive and their training cost is higher), openness (SRs can make use of a large number of open source networking applications, while PRDs are more closed) and flexibility. The forwarding performance provided by SRs has originally been an obstacle to their deployment in real networks. For this reason, a good amount of recent research focused in increasing such a performance by either tuning single devices (as in Packet Shader [6], Click [54] and Netmap [55]), or aggregating multiple single devices to form a more powerful routing unit like the Multistage Software Router [14], Router Bricks [43] and DROP [56]. While some recent studies show that even using a single device it can be possible to reach multiple tens of Gigabit forwarding speed [6], other works show that by aggregating multiple routing units the forwarding speed can scale almost linearly with the number of devices [14].

Such promising results suggest that SRs could be readily adopted in terms of performance, but some other features related to flexibility (such as power saving, programmability, router migration or easy management) have been investigated less than performance. As recognized by several researchers [20, 22], virtualization techniques could become an asset in networking technologies, providing SRs with the needed flexibility and easy management. As an interesting example, Virtual Machine (VM) migration could be adopted for consolidation purpose to save energy. Thus running SRs in virtualized environment can easily inherit such new features that are valuable to build the flexible SRs.

For example, running a SR in a VM can provide three main advantages:

- renting routing resource instead of buying new hardware to improve the performance. This feature is especially useful when high processing power is necessary for a short term;

- easier management and reliability: Migration of VMs during maintenance periods can be implemented and faster reaction to failures should be expected by booting new VMs on general purpose servers;

- slicing and sharing the same physical infrastructure among different users to improve hardware efficiency.

However, Virtual Software Routers (VSRs) presents some other issues with respect to SRs. For example, if the communications between VMs and hosts are not correctly understood, the complex interactions between hardware and VMs could easily compromise the performance provided by a SR. This chapter focuses on analyzing such interactions to identify and remove the various performance bottlenecks of a VSR. We exploit an open-source virtualization mechanism (KVM, the Kernel-based Virtual Machine [30]) which permits to easily analyze the VM and SR behavior to identify performance bottlenecks. Indeed, since KVM is tightly integrated into the Linux, it is possible to exploit all the available Linux management tools. Furthermore, KVM has been included into the Linux mainline from 2.6.20 on. Thus, almost all Linux system can host KVM guest, which gives us the possibility to migrate the VSRs easily.

The VSR performance can be improved by carefully tuning various parameters such as the mechanism used to connect the VMs, threads priorities and CPU affinities, and the technique used to build modular VSRs. We will show in the following sections that a proper configuration and optimization of the virtual routing architecture and the aggregation of multiple VSRs as the MSR architecture permit to forward almost 1200*kpps* (with 64 bytes packets) in a commodity PC, close to the physical speed of a Gigabit Ethernet.

4.2 KVM Virtualization Framework

The performance of a VSR mainly depend on the amount of available CPU time, and by the amount of CPU time consumed by the VSR, which is mainly due to two different activities:

1. CPU time consumed by the forwarding code running in the guest (this can be the packet forwarding subsystem of the Linux kernel, or Click, or something else)

2. CPU time consumed by the host to move packets from virtual switches (or from physical interfaces) to the virtual interfaces of the guests (or vice versa).

The time consumed by the host to forward packets to/from the VM (item 2) can be spent in the OS kernel, in the hypervisor, or in the user-space virtual machine, depending on the virtualization architecture.

Running the VSR on a "closed" virtualization architecture such as VMWare [57], it is not easy to understand how much of the CPU time is consumed by the host, by the guest, or by OS kernel. Hence, in this chapter an open-source virtualization architecture is used. The two obvious candidates are Xen [29] and KVM [30]. Since the KVM architecture is more similar to the standard Linux architecture (and hence, does not require to learn new profiling and performance evaluation tools), this chapter is based on KVM.

KVM (the Kernel Virtual Machine) is based on a kernel module which exploits the virtualization features provided by modern CPUs to directly execute guest code and on a user-space VM, based on QEMU [58], which virtualize the hardware devices and implements some virtual networking features.

When considering VSR, the most relevant features provided by the user-space VM are the emulation of network interfaces (CPU virtualization is not an issue, as KVM allows guest machine instructions to run at almost-native speed): when a packet is received, the VM reads it from a device file (typically the endpoint of a TAP device, or similar) and inserts it in the ring buffer of the emulated network card (the opposite happens when sending packets). When emulating a standard network interface (such as Intel e1000), the VM moves packets to/from the guest by emulating all of the hardware details of a real network card, and this is pretty expensive, causing poor networking performance (especially when considering small packets, and/or high interrupt rates). This problem can be solved by using `virtio-net`, which does not not emulate real hardware but uses a special software interface (virtio [59]) to communicate with the guest (that then needs special virtio drivers). In this way, the overhead introduced by emulating networking hardware is removed, and network performance is improved. In particular, virtio is based on a ring of buffers shared between guest and VM, which can be used for sending/receiving packets. Guest and

VM notify each other when buffers are empty/full, and the mechanism is designed to minimize the amount of host/guest interactions (by clustering the notifications, and allowing to transfer data in batches).

As already said, when using virtio-net, the user-space VM code is still responsible for moving data between the (endpoint of the) TAP interfaces and the virtio buffers. Hence, when a packet is received:

- The host kernel notifies the user-space VM that a new packet is available on the TAP device file
- The VM is scheduled, and reads the packet from the device file
- The VM copies the packet in the virtio-net buffer, and notifies the guest
- The guest receives a virtio interrupt, so the guest kernel executes and can receive the packet

In summary, the large number of switches between host kernel, VM, and guest, can introduce overhead and decrease the virtual router performance. This problem is solved by using`vhost-net` [1], which is a helper mechanism provided by the host kernel, able to directly copy packets between the TAP interface and the virtio-net buffers. In this way, the copy is not performed by the user-space VM, but by a kernel thread (the vhost-net kernel thread) and some context switches can be avoided. As a result, the network performance of the guest are largely improved.

When using vhost-net, the user-space VM does not need to execute when the guest sends and receives network packets, and the CPU time consumed by the host to move packets is not used by the user-space VM but by the vhost-net kernel thread. Notice that there is 1 vhost-net thread per virtual interface. The guest code executes in a different thread, named vcpu thread. Notice that there is 1 vcpu thread per virtual CPU.

4.3 Monotonic Virtualized Software Router Performance

Thanks to the fact that KVM is an open virtualization architecture, it becomes possible to analyze the performance bottlenecks of a VSR. For example, consider the packet forwarding performance of a Linux-based OS running inside a VM: when forwarding small packets (64 bytes long), a Linux-based VSR is not able to forward more than 900*kpps* (900000 packets per second), as shown in Fig. 4.1. This figure shows the forwarding packet rate (as a function of the input packet rate) of a 3.0

[1] `http://www.linux-kvm.org/page/VhostNet`

Linux kernel running inside qemu-kvm 1.1.0 on an Intel Xeon E5-1620 at 3.66GHz. Each experiment has been repeated 10 times, and the figure also shows the 99% confidence intervals. Since this CPU has multiple cores, the VSR has been tested while using a singe core (all interrupts are processed on the first CPU core, where the KVM vcpu thread and the vhost-net kernel thread also run), 2 cores (interrupt processing and threads execution on the first 2 CPU cores), and all 4 CPU cores.

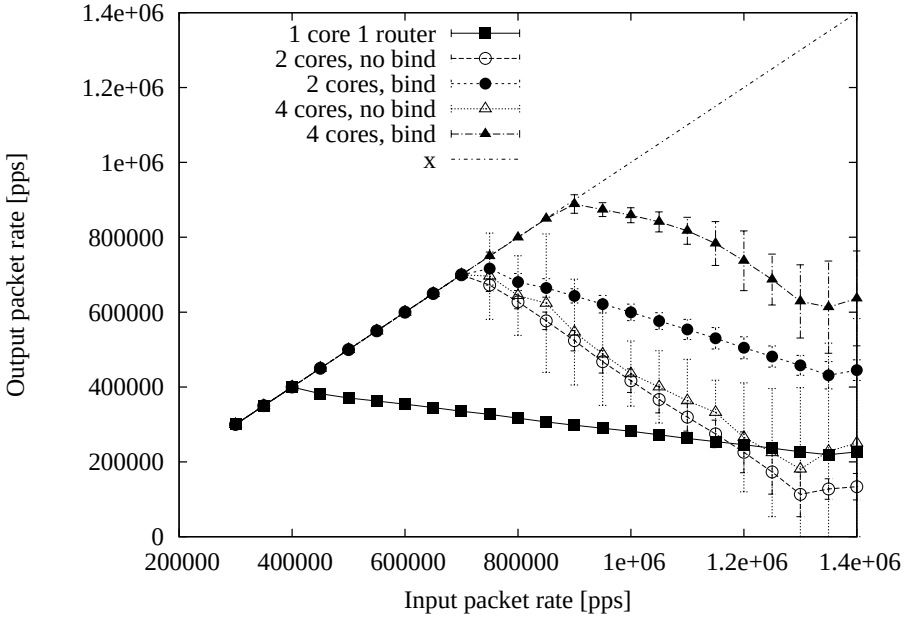


Figure 4.1. Performance of a monolithic virtual router, increasing the number of CPU cores.

Fig. 4.1 shows that using multiple cores improves VSR performance. However, if the vcpu thread and the vhost-net thread are not properly scheduled (no binding), performance strongly decrease in overload conditions. In other words, there is no graceful performance degradation when the router is overloaded. For example, looking at the "2 cores, no bind" and "4 cores, no bind" lines in the graph, it can be observed that performance increase moving from 2 cores to 4 cores is not relevant, and for high input rates (around 1200*kpps*) 2 cores perform worse than 1 single core (thus, having more processing power does not help). Analyzing the Linux scheduling statistics and the amount of time consumed by the various threads (by using the `top` utility), it has been obvious that this performance drop is due to *migration overhead*: When the task scheduler (implemented in the Linux kernel) realizes that a CPU core is almost overloaded, it tries to *migrate* some tasks from it to a different CPU core. Unfortunately, if all of the usable cores are overloaded, tasks keep bouncing between different cores, and most of the CPU time is consumed

by these thread migrations (hence, the throughput decreases). This problem can be fixed by *binding* threads to CPU cores: Forcing specific threads to execute only on some CPU cores by using the `CPU affinity` mechanism provided by the Linux kernel. The "2 cores, bind" line shows the results achieved when forcing the vcpu thread to run on the first core, and the vhost-net thread to run on the second core. The maximum throughput does not increase with respect to the "2 cores, no bind" case, but the performance degradation in overload is now more graceful, and 2 cores are able to always perform better than 1 single core. Finally, the "4 cores, bind" line shows the results achieved when using the first 2 cores for interrupt processing, and binding the vcpu thread and the vhost-net thread to the third core and to the fourth core, respectively. Again, the graph shows that correct CPU bindings allow to better exploit additional cores. However, even when 4 CPU cores are used (and `top` shows a high amount of idle CPU time in the system) the virtual router is not able to forward more than 900*kpps*.

When the system is unable to forward all received packets, for input rates larger than 900*kpps*, the bottleneck is the vhost-net kernel thread, which consumes all the CPU time on a core. Since the issue is that a single thread (the vhost-net thread) needs more than 100% of the CPU time of a single core, playing with CPU bindings cannot help anymore (because a single thread cannot simultaneously execute on 2 different CPU cores) and it is not possible to exploit the huge amount of idle time on the other cores.

This unused computational power can be exploited by moving to a more modular virtual router architecture, using more VMs as routers, thus using multiple vhost-net threads (so that their load can be shared on more CPU cores). This is theoretically possible by using aggregating multiple VSRs (as suggested, for example, in the Multistage Software Router - MSR - architecture [14]).

4.4 Aggregating Multiple Virtual Routers

As shown in Fig. 4.1, even with the correct priority and optimal thread binding, a "monolithic" VSR (that is, a VSR based on a single VM running a SR) is not able to forward more than 900*kpps* (when using small packets). Since the bottleneck lies in the *vhost-net* thread (that is, such a thread consumes 100% of the CPU time of the core where it is running), virtualizing a multiprocessor machine does not improve performance.A solution to this issue could be to move from a monolithic VSR to a routing architecture based on the aggregation of multiple SRs running inside multiple VMs.

Such an aggregation can be performed using several different virtual routing architectures, exhibiting different characteristics in terms of performance and flexibility. In the following two different architectures are discussed.

A very flexible and feature-rich example of SR aggregation is the Multistage Software Router architecture (MSR) [14], shown in Fig. 1.1.

Recall that a Multistage Software Router (MSR) is composed of three stages as follows:

- first stage: layer-2 *Load Balancers* (LBs) distribute the input traffic load to *Back-end Routers* (BRs)
- second stage: interconnection network. A mesh-based switched network between the first stage LBs and the third stage BRs. Multiple paths between the LBs and BRs could exist to support fault recovery.
- third stage: BRs, i.e, forwarding engines, route packets to the proper LBs

A virtual control processor is used to coordinate the BRs and LBs as well as to unify the BRs' routing table. The MSR hides its internal architecture and presents itself to external devices as a single router. As shown in previous work, the MSR architecture, when implemented on a cluster of physical machines, provides several interesting features, such as extending the number of interfaces one PC can host (limited PCIe slot), dynamically shutting down unnecessary BRs at low traffic load while turning on BRs at high load, and seamlessly increasing the overall routing performance. In particular, it has been shown that the MSR's forwarding speed can scale almost linearly with the number of BRs (if LBs are implemented using a fast FPGA hardware - otherwise, LBs can become the performance bottleneck).

To implement a *virtual MSR*, the LB has been implemented in software, using the Click modular router [54]. In this way, an MSR can be easily hosted in VMs by just substituting each physical PC with a VM. Hence, multiple VMs are created to run LBs and BRs.

The interconnection network is implemented inside the host as shown in Fig. 4.2, using a standard Linux "software bridge" and some pairs of tap interfaces. Connection with the lower device like the physical NIC can be implemented using the Linux `macvtap` feature to improve performance.

Implementing the first stage (LBs) with Click provides high flexibility: it become possible to build MSRs with a variable numbers of LBs and BRs, with a wide range of interconnection networks allowing for BRs distributions on different hosts, redundancy/fault tolerance, etc. However, it comes at the cost of consuming lots of CPU power in the vcpu threads running the LBs, and in their vhost-net kernel threads. This means that the number of CPU cores needed to provide high performance becomes extremely high (the "non virtual" MSR implemented with real PCs could use an FPGA for load balancing, to avoid this kind of issues).

If the focus is on forwarding performance (and some features/flexibility can be traded for higher performance), then a different SR aggregation strategy can be

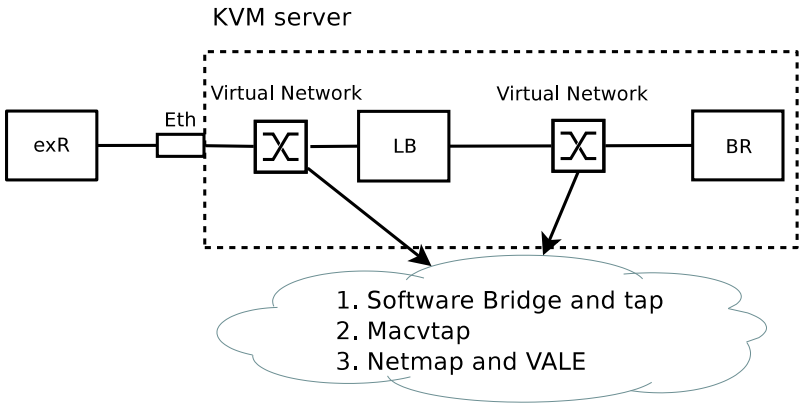


Figure 4.2. The implementation of the multistage software router inside a KVM server: 1 load balancer and 1 back-end router, with different interconnection networks.

used. The Linux `macvtap` interface provides a multi-queue functionality that can be used for load balancing: a single `macvtap` interface can split the traffic on multiple queues (currently based on network flows, but can be modified to distribute packets in a round-robin fashion). Such packet queues can be used by a single VM (using a multi-queue virtual network interface), or by multiple VMs. In this chapter, this feature is used for running multiple identical copies of the same SR, each one of them using a different `macvtap` queue. All of these SRs run in identical VMs (having the same number of Ethernet interfaces, with the same IP and MAC addresses) and are seen from outside as a single VSR (hence, the multi-queue `macvtap` aggregates all the SRs in a single VSR with the same configuration). This architecture, referred as Parallel Virtual Routers (PVR) architecture in this chapter, is less flexible than MSR, but removes the LB performance issues (as shown in the next section).

4.5 Performance Evaluation

Before starting to analyze the virtual routing performance of the proposed architectures (MSR and PVR), a set of preliminary experiments were run to understand the performance impact of some configuration parameters (such as scheduling algorithm and task's priorities, the mechanism used for implementing the virtual switch component, etc...).

4.5.1 Virtual Network and Linux Scheduler Tests

In KVM virtualization framework, different mechanisms can be used to connect the physical NIC with virtual interfaces. In the preliminary experiments[2], a `software` bridge plus `tap` interface and `macvtap` interface have been considered.

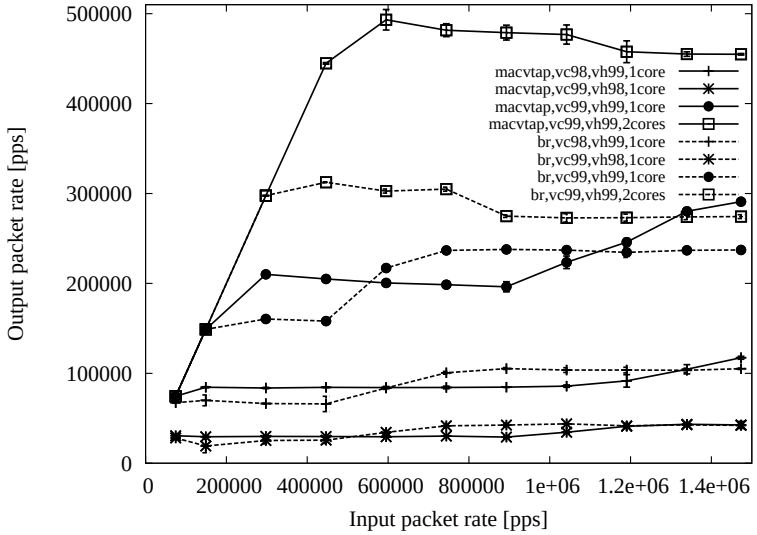


Figure 4.3. Forwarding performance in KVM virtualized environment, with macvtap and bridge configurations, different priorities for the vhost and vcpu threads.

These experiments are based on a very simple configuration with a KVM instance hosting a SR. In this case, there are two CPU-consuming threads: the vcpu thread (running the forwarding code) and the vhost-net kernel thread (moving packets between the physical and virtual interface). These two threads were bounded to one or two CPU cores, and scheduled with fixed priorities (using the `SCHED_FIFO` scheduling class). From Fig. 4.3 it is possible to appreciate the throughput differences for the different priority configurations (98 and 99). The best results are obtained when assigning the same priority to the two threads. Furthermore, assigning a higher priority to the vhost thread performs better than assigning a higher priority to the vcpu thread. This is due to the fact that vhost is responsible for packet reception and transmission before and after the packet processing phase by the vcpu thread. Thus, if the vhost thread does not have enough resource to move the packets from the VM back to interface, the vcpu could waste precious CPU resources to process packets only, without moving them to the external network. On the contrary, a higher priority for the vhost can guarantee that each packet processed by the vcpu can be correctly received by the external world. Therefore, it can reach a higher

[2]Experiments are performed on a server with Intel Xeon E5520 at 2.27GHz

throughput. The fact that assigning the same priority to the vcpu thread and to the vhost-net thread provides the best performance might suggest that priority scheduling is not the best option for this workload. Hence, as a future work, a more advanced scheduler (namely, a reservation-based CPU scheduler [60, 61]) will be tested. Finally, `macvtap` always performs better than `bridge` plus `tap` connections and should be preferred when building a high performance VSR.

4.5.2 Parallel Virtualized Router Performance

The next experiment focus on analyzing the PVR performance, to understand if the PVR architecture permits to outperform a monolithic VSR. Fig. 4.4 displays how the performance of a PVR using the multi-queue macvtap mechanism for load balancing is affected by the number of aggregated routers. 4 CPU cores are used, and the setup is the same used in Figure 4.1 (same CPU, same number of runs per experiment, and the 99% confidence interval is displayed). Note that this VSR is able to outperform a monolithic VSR (Fig. 4.1 - the best curve from that figure is repeated in Fig. 4.4 as "1 router, bind") and reaches a forwarding rate of more than 1100*kpps*. Note that when aggregating 2 routers the CPU bindings are not important for the forwarding performance. When, instead, aggregating 3 routers, using proper bindings permits to better exploit the CPU time. Increasing the number of routers, the bindings become less relevant, but performance do not improve. This probably indicates that 4 CPU cores are not able to forward more than 1200*kpps* in a virtual architecture.

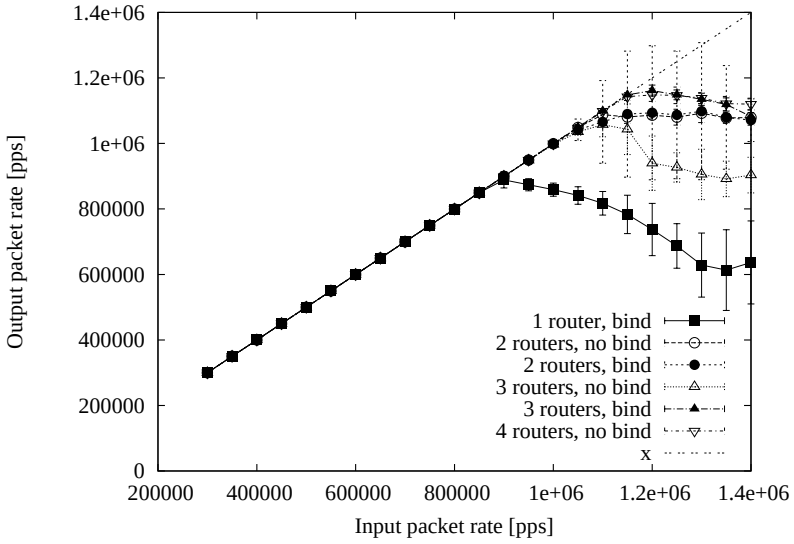


Figure 4.4. Performance of a PVR on 4 CPU cores, increasing the number of aggregated routers.

4.5.3 Multistage Virtualized Router Performance with Optimization

The last set of experiments focused on analyzing the virtual MSR performance. Fig. 4.5 shows how the number of available CPU cores and the usage of correct CPU bindings affect the performance of an MSR. For the sake of simplicity and to easily understand the results, we discuss the MSR configuration with only 1 LB and 1 BR, but similar experiments with more complex setups have been also performed providing results consistent with the ones presented here. Before analyzing the results, consider that this MSR configuration (1 LB and 1 BR) creates 5 CPU-consuming threads: 1 vcpu thread and 2 vhost-net threads for the LB (since the LB has 2 virtual Ethernet interfaces), plus 1 vcpu thread and 1 vhost-net thread for the BR. As a result, the performance when executing on a single CPU core are pretty bad (the 5 threads easily overload a single core).

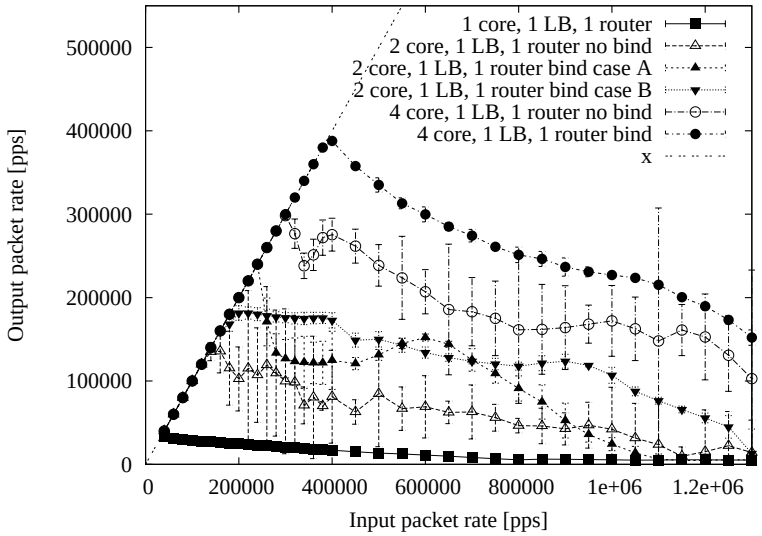


Figure 4.5. Effects of CPU bindings on the MSR performance.

When increasing the number of CPU cores to 2, performance slightly improve and become very sensitive to the CPU bindings: When no bindings are used, the MSR can route up to 140*kpps*. In overload, the 5 threads tend to overload the CPU and keep bouncing between the 2 available cores. As a result, the variance in the forwarded throughput is high (see the confidence interval), and performance decrease. Binding the vcpu threads of the LB and of the BR to the first core, and binding all the vhost-net kernel threads to the second core ("bind case A" in the figure) allows to achieve a higher maximum throughput (about 240*kpps*). However, when the router is overloaded throughput decreases dramatically, and for an input

rate of more than 1000*kpps* it performs worse than without bindings. This happens because the three vhost-net threads overload the second core, when there still is some idle time on the first one. Distributing the threads between cores in a different way ("bind case B" in the figure: the vhost-net kernel threads of the LB execute on the first core, together with the vcpu thread of the BR, all other threads are on the second core) leads to a lower maximum throughput (about 180*kpps*) but to a more stable behavior in overload.

Finally, when increasing the number of cores to 4 the MSR performance further improve (because more CPU time is available for the 5 threads). Also in this case proper CPU bindings permit to improve the performance. Note that MSR performance are affected by the fact that there are only 4 available CPU cores, for executing 5 threads. By looking at the scheduler statistics and at the CPU usage inside the host, it has been possible to see that the main performance problems are due to the vcpu thread of the LB, running Click, that consumes 100% of the CPU time on a core. This explains why the PVR architecture (which does not use Click for load balancing) is able to better exploit the computational power provided by the 4 CPU cores.

4.6 Conclusions and Future Work

In this chapter we showed how using an open virtualization architecture permits to analyze and improve the forwarding performance of a virtual router. In particular, the forwarding performance of the Linux kernel running inside a KVM virtual machine have been analyzed, seeing how increasing the number of used CPU cores can improve performance and how properly setting the CPU affinity of the various virtualization activities affects router throughput. It has been shown that a more modular router architecture can help in better exploiting the computational power provided by additional CPU cores, and two different architectures based on virtual routers aggregation have been analyzed and optimized to outperform the monolithic architecture.

Since a proper task scheduling turned out to be fundamental to achieve good forwarding performance, we plan to investigate the impact of different CPU scheduling algorithms. In particular, as discussed in the comments to Figure 4.3 we feel that fixed priority scheduling is not the appropriate solution for scheduling the vcpu and vhost-net threads, and we plan to try reservation-based CPU scheduling.

Finally, alternative high-performance inter-VM communication mechanisms such as Netmap and VALE [55, 62] will be tested for improving the virtual routing performance.

Part II

Energy Saving Techniques in Network on Chip and Multistage Software Router

Chapter 5

Introduction

As mentioned in [63], according to different studies [64, 65], the carbon footprint of Information and Communication Technologies (ICT) is constantly increasing, representing today up to 10% of the global CO2 emissions. Among the main ICT sectors, 37% of the total ICT emissions are due to telecommunication infrastructures and their devices, while data centers and user devices are responsible for the remaining part [64]. It is therefore not surprising that researchers, manufacturers and network providers are spending significant efforts to reduce the power consumption of ICT systems from different angles.

To this extent, networking devices waste a considerable amount of power. In particular, energy consumption has always been increased in the last years, coupled with the increase of the offered performance [66]. Actually, power consumption of networking devices scales with the installed capacity, rather than the current load [67]. Thus, for an Internet Service provider (ISP) the network power consumption is practically constant, unrespectively to traffic fluctuations, since all devices consumes always the same amount of power. In turn, devices are underutilized, especially during off-peak hours when traffic is low. This represents a clear opportunity for saving energy, since many resources (i.e., routers and links) are powered on without being fully utilized, while a carefully selected subset of them can be switched off without affecting the offered Quality of Service (QoS).

In the literature, different approaches have been proposed to reduce the gap between the capacity offered by the network and the resources required by users (see [68, 66] for an overview). The proposed approaches can be divided into two main categories: power proportional techniques that adapt the capacity (and thus consumption) of the devices to the actual load, and sleep mode approaches, that leverage on the idea of introducing idle mode capabilities. While the first approach involves deep modifications in the design of hardware components, the second approach requires coordination among networking devices to carefully distribute the extra load that results from putting into sleep mode some devices.

In this party of the thesis, we exploit the idea of power proportional design in two main switching architectures, namely the Network on Chip (NoC) and Multistage Software Router (MSR). In the last decade, the fast evolution of semi-conductor and silicon technology make the integration of large number of cores on a single chip become reality, and more preferable due to attractive low inter-chip communication cost compare to intra- ones. This mew design paradigm has driven the System on Chip (SoC) from bus or peer-to-peer (P2P) communication architectures towards short distance multi hop network connection, namely Network on Chip (NoC) architecture. Indeed the bus based SoC lacks of spatial reuse, possess more area $O(n^3 \times \sqrt{n})$ and consume more power $O(n \times \sqrt{n})$, even the connection from the Processing Element(PE) to the bus needs to be specific for different chips. In the mean while NoC interconnection inherits the advantage from networking hop communication which consumes less power($O(n)$), uses less space($O(n)$), equips with standard interface and links can do statistic multiplexing and spatial reuse. All of the above features make NoC more promising for large scale interconnection no chip network [69]. Whereas as many network design principle, the NoC is devised to handle the peak traffic load. This provides us the opportunity to save energy when the workload on chip is low. In chapters 6 and 7, we present how to exploit Dynamic Voltage and Frequency Scaling (DVFS) technique combined with routing and architecture design to save power in two different NoC architecture, namely the single plane and multiple planes NoC architecture respectively. In both works, we modeled the NoC power saving as quadratic programming problems to obtain the lower bound and we found that the current chip power strategy exists quite a gap with the optimal value. Then we proposed some algorithms with lower implementation complexity to approach the lower bound. Our results have been validated by customer NoC simulator, which gets the help from the SystemC and TLM libraries that are intended for fast hardware simulation. We also considered different power models, one of which are quite accurate to trace not only the dynamic power but also the static one contributed by such as clock pump or leakage.

In chapter 8, we present a work related with energy efficient MSR design. As shown in the first part, the MSR architecture composed of many physical servers to compensate the performance limitation of software router. Although we can extend the forwarding speed by increase the number of device, it is clear that the total energy consumption of such architecture is huge. For example, if we want to build a MSR which has the similar routing capacity of a Juniper T320 (16 interfaces and each of which works at 10Gbps, 160Gbps forwarding capacity in total, it consumes 2.88kWatts power), we need to have 16 LBs (each of them equips with single 10Gbps link and consumes 80w), 20 back-end routers (each of them equips with 8 links at 1Gbps and consumes 80w) and one switch. The total power consumption is 3.2kWatts (1.92kWatts by back-end routers) and the total cost is $16,000 assuming each PC costs $400 comparing to Juniper T320 which cost $45,000. This MSR

architecture consumes significant energy if we leave it 24 hours on. In the other hand, we can save energy consumed by unused routers and links by switching them off during low load period. In chapter 8, we will show how to build an energy efficient MSR architecture when providing the realistic traffic trace with different budget constraints.

Chapter 6

Balancing Traffic to Save Power Through DVFS in NoC

A Network on Chip (NoC) provides the interconnection among Processing Elements (PEs) through routers, which permit hop-by-hop communications between PEs. To cope with higher traffic demands, PEs and routers are running at increasingly higher clock frequencies. Thus the chip power consumption grows rapidly and limits NoC scalability.

In this chapter we consider a Manhattan-like mesh (grid) NoC topology. We show how to leverage the traffic unbalancing within the topology to fully exploit the classical technique of Dynamic Voltage and Frequency Scaling (DVFS) to minimize the power consumption. We model the optimal NoC power control problem, and we evaluate the maximum achievable power reduction. Furthermore, we propose three different load-balancing routing schemes, simple to implement, that approximate quite accurately the optimal solution. Simulation results show that, in most of the cases, it is enough to consider only two paths among PEs to balance the traffic and to approach the minimum possible power consumption.

6.1 Introduction

The fast evolution of semi-conductor and silicon technology makes the integration of large number of Processing Elements (PEs) on a single chip a reality. Single chip hosting many PEs are becoming increasingly popular due to the attractive low cost intra-chip communications compared with inter-chip ones. This new trend has driven the System on Chip (SoC) design from point-to-point dedicated communications towards the Network on Chip (NoC) architecture. In a NoC, a network interconnects PEs through on-chip "routers", where packetized communication takes

place. PEs access the network by means of proper interfaces, and have their packets forwarded to destination through a multi-hop routing path. Indeed, classical SoCs lack spatial reuse, require more area and consume more power. Instead, NoCs permit to cope with the growth of SoCs complexity and inherit advantages from multi-hop communications. NoCs, besides reducing area and power requirements, are equipped with standard interfaces, exploit statistical multiplexing and spatial reuse, and are the new frontier in large scale chip network design [70]. Interestingly, many communication paradigms devised for "large-scale" standard computer networks can be adapted in this "small-scale" network scenario.

In this chapter we consider NoCs based on a mesh (Manhattan-like) topology, since they are robust to traffic fluctuations and their design complexity scales quite linearly with the number of PEs [71]. In such context, we revisit the Dynamic Voltage and Frequency Scaling (DVFS) mechanism to reduce the NoC power consumption. In general, DVFS is a technique adopted in CMOS gates to jointly adjust the supply voltage and the clock frequency. We take into account only the dynamic component of the power cost, i.e. the power due to the bit-by-bit switching activity on the links to transmit data among PEs. The dynamic power P depends on voltage V and frequency f as: $P \propto fV^2$. To properly receive the bits, the maximum bit frequency must be (approximately) proportional to V, i.e. reducing the voltage implies increasing the bit transmission duration, as shown in Fig. 6.1. Thus, the maximum power consumption is roughly proportional to f^3 when DVFS is exploited.

To reduce chip implementation complexity, we consider a NoC architecture in which all communication components use the *same* voltage, and hence the same frequency, at a given time. This architecture is known as the Single Voltage/Single Frequency (SVSF) chip [72]. and the frequency/voltage of all links is simultaneously regulated through DVFS. The SVSF solution has been proved to perform better than link-by-link DVFS in [73]. Furthermore, we propose to adopt only deterministic routing algorithms, oblivious of the current load of each link. This choice permits to avoid additional load detection circuits introduced by adaptive routing. The traffic matrix, i.e. the amount of traffic that PEs exchange, is statistically known either by estimation or measurements. Our control scheme chooses the proper frequency and voltage values for the whole chip to minimize the required power while satisfying communication demands. The approach is compatible with the traffic being stationary over a time window large enough to amortize the voltage/frequency reconfiguration overhead during such time window.

We define the NoC power saving problem as a combined traffic routing and voltage selection problem. We provide a mathematical formulation of the problem and solve it through a linear programming solver. Our results highlight the potential power reductions when considering DVFS into NoCs. Furthermore, we observe that if the link utilization is unbalanced, one or more bottleneck links constrain the choice of the voltage, and prevent the efficient use of DVFS on underutilized links to save

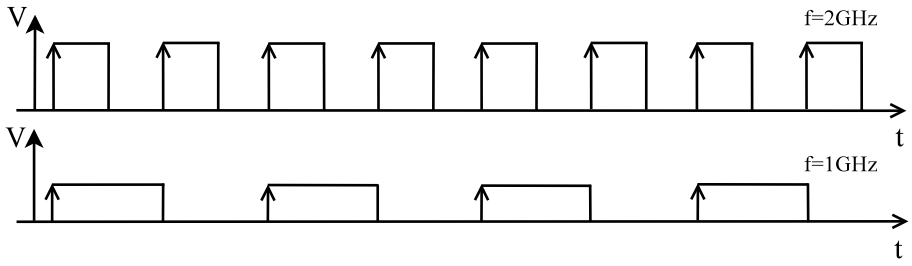


Figure 6.1. Reducing the NoC supplying voltage to half implies reducing the chip working frequency to half. Bit transmission duration is doubled.

power. However, PEs process multiple tasks, and each task contributes to the traffic flow toward a specific PE. Thus, several flows can be identified among each pair of PEs. Data belonging to different traffic flows can follow different routing paths without any out-of-order delivery. This fact permits to exploit routing schemes to better balance the network load. We propose to couple the Load Balancing (LB) technique with deterministic routing to spread the traffic flows in the network, to avoid load concentration and fully exploit DVFS. Simulation results show that our LB approach closely approximates the optimal solution, but with lower algorithmic complexity.

6.2 Related Work

Introducing DVFS into SoC has been proposed and studied in the literature for the past few years. The main idea is to slow down the PE frequency speed during the idle time. Indeed, [74] proposes to decouple the PE processing time from the transmission time; when the communication rate is slow and the PE has been stalled, the supply voltage for the PE is decreased to match the rate of communication channel. [75] is a similar work which explicitly considers the PEs leakage power. [76] proposes an on-line detection method which uses a built-in performance monitoring unit to decide the PE processing requirements, and regulates the PE speed accordingly.

The above mentioned papers aim at PE power cost reductions. Few works investigate the transmission cost when link frequency can be adjusted. [77] considers a 8×8 NoC with grid topology, in which each router can adjust its links frequency independently, according to the prediction of traffic load on links. Since each link can work on a different frequency, such scheme is very complex to be implemented. [78] and [79] show the hardware implementation of serial and parallel transmission lines adopting DVFS within a NoC. [80] shows how to design a specific purpose NoC, to save power and achieve high performance. DVFS is adopted for the capacity planning of PEs and links. The final architecture is a specialized NoC, less

appealing than a generic purpose interconnection network.

The most relevant work with ours shown in [73] compares adaptive routing with deterministic XY routing, in two scenarios: for SVSF NoC and for heterogeneous DVFS, i.e. in which each link of the NoC can be configured with its own frequency and voltage. The paper shows that adaptive routing with SVSF NoC scheme offers the best performance-complexity tradeoff, since the overhead due to frequency reconfiguration in heterogeneous DVFS negatively affects performance. Differently from [73], we present the mathematical model of the power saving problem and we derive the lower bound to the chip power consumption with DVFS. Then, we focus on simple deterministic routing algorithms that exploit load balancing. We do not consider frequency/clock islands as in [81] because this solution may improve power savings at the cost of more complex synchronization.

6.3 NoC Model Description

6.3.1 Network Topology and Traffic model

We assume that the NoC interconnection exploits a grid (mesh) network, one among the most common topologies [82] due to its packing simplicity and low power dissipation [83]. The topology is shown in Fig. 6.2, and it is divided into logical areas. Each logical area, named *tile*, is composed of one physical router and one logical PE, which represents one or more physical PEs connected to the same router through a proper Network Interface (NI). We represent the network topology with N tiles as a directed graph with node set $\mathbb{V}$, in which each tile is a node $\in \mathbb{V}$ and adjacent tiles are connected through two opposite unidirectional edges. Each PE generates traffic depending on its running tasks. Indeed, each PE could be a memory bank, a microprocessor, a DMA engine, etc. The router supports very basic switching capabilities, because of design constraints related to performance, area and memory. To reach a destination PE, simple deterministic routing is adopted. "XY" routing is very common: data is first routed in the X direction, until reaching the Y coordinate of the destination, and then routed in the Y direction. The interconnections among all components (e.g. router-router, PE-router) are provided by CMOS-based channels. FIFO buffers with limited size are available in PEs and routers. Wormhole routing is typically adopted: data packets are divided in small *flits*.

We assume that the traffic demands among PEs are given. Depending on the actual application running on the chip, such demands can be either known in advance, or estimated on-line. The traffic demand from node i to j is denoted by λ_{ij}; let $\Lambda = [\lambda_{ij}]$ be the corresponding $N \times N$ traffic matrix.

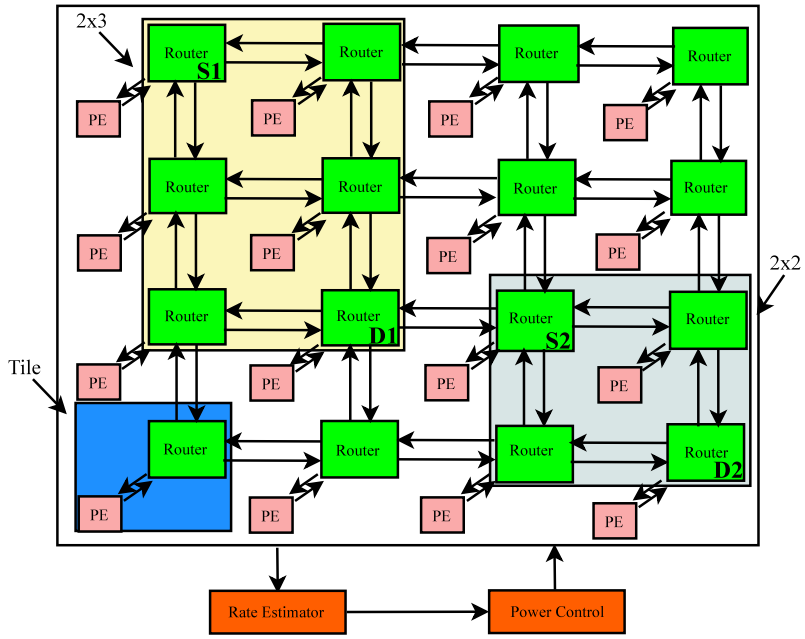


Figure 6.2. A 4×4 NoC grid topology

6.3.2 Power Model and Power Control

The objective of our power control is to find a routing that minimizes the power consumption while satisfying the traffic demands and meeting the link capacity constraints.

We assume that all links in the NoC operate at a single frequency, powered by the corresponding voltage V, that can vary between $V_{\min}$ and $V_{\max}$. We focus only on the dynamic power contribution due to the data transfer across routers. The energy cost in a CMOS gate depends on the supply voltage V as $E = 0.5CV^2$, where C is the load capacitance. This is usually called dynamic energy cost of a CMOS, due to switching activity when changing the logic state from 0 to 1 or vice versa. We neglect static energy due to leakage currents. Thus, we assume that the total amount of dynamic energy when moving b bits across a path of h hops is proportional to bhV^2.

This hop-based energy model has been validated in the literature [84, 85]. When transmitting continuously at rate λ bit/s, the power consumption becomes proportional to λhV^2.

Let μ be the maximum rate compatible with the maximum available voltage $V_{\max}$; the corresponding power consumption is maximum and proportional to $P_{\max} = \mu hV_{\max}^2$. To avoid traffic overload, we assume that $\lambda \leq \mu$. To exploit DVFS, the bit duration can be expanded (at most) by a factor $\alpha = \mu/\lambda$, denoted as *expansion*

factor. Hence, the voltage can be decreased by α. As a consequence, the total power on a link is proportional to:

$$P = \lambda h \left(\frac{V_{\max}}{\alpha} \right)^2 \propto \frac{h\lambda}{\alpha^2} \tag{6.1}$$

Given the minimum allowed voltage $V_{\min}$ in the considered technology, then $\alpha \leq \alpha_{\max}$, where $\alpha_{\max} = V_{\max}/V_{\min}$ corresponds to the maximum expansion factor [86]. Note that $\alpha_{\max}$ is often around 2 and never more than 3.

6.3.3 Traffic Virtual Load and Power

Given a static routing policy $\mathcal{R}$ and the traffic matrix Λ, $\gamma_\Lambda^\mathcal{R}$ is defined as the maximum offered load among all possible edges in the topology. When DVFS is applied with bit expansion factor α, Λ is said to be *sustainable* if $\gamma_\Lambda^\mathcal{R}\alpha \leq \mu$; the ratio $\rho = \mu/(\gamma_\Lambda^\mathcal{R}\alpha)$ is defined as the *virtual load*.

The overall power consumption of a NoC exploiting DVFS is the sum of all flow power contributions. From (6.1),

$$P_{tot} = \sum_{i,j \in \mathbb{V}} \frac{\lambda_{ij} h_{ij}}{\alpha^2} \tag{6.2}$$

where h_{ij} is the average number of hops for the paths used to route the flow λ_{ij}. The maximum power consumption $P_{tot}^{\max}$ is obtained by setting $\rho = 1$. We prove the following property that will be validated in Sec. 6.5.

Property 1. *Assume ideal DVFS with unbounded $\alpha_{\max}$ (i.e., $V_{\min} = 0$). Given a routing policy $\mathcal{R}$ and a sustainable traffic, then the overall power cost of the NoC is a cubic function of the virtual load ρ: $P_{tot}(\rho) = P_{tot}^{\max}\rho^3$, for $0 < \rho \leq 1$.*

Proof. Assume that Λ is a traffic matrix for which $\rho = 1$. Hence, it is not possible to reduce the voltage without reducing the throughput: $\alpha = 1$. Thanks to (6.2), the corresponding power is $P_{tot}^{\max} = \sum_{ij} \lambda_{ij} h_{ij}$. Now consider to scale the traffic matrix by ρ. In this case,

$$P_{tot}(\rho) = \sum_{ij} \frac{(\rho\lambda_{ij})h_{ij}}{\alpha^2} = \sum_{ij} \rho^3 \lambda_{ij} h_{ij} = \rho^3 P_{tot}^{\max} \tag{6.3}$$

because it is possible to expand the bit duration by $\alpha = 1/\rho$. □

6.4 The DVFS Power Control

The power control aim is to minimize the NoC dynamic power due to the switching activities exploiting DVFS. This problem is denoted as Ideal Power Control (IPC) and can be formulated as follows:

$$\min_{1\leq\alpha\leq\alpha_{\max},f_{ij}^{ml}\geq 0} \quad \sum_{i,j,m,l\in\mathbb{V}} \frac{f_{ij}^{ml}}{\alpha^2} \tag{6.4}$$

subject to:

$$\sum_{i,j\in\mathbb{V}} f_{ij}^{ml}\alpha \leq \mu \quad \forall m,l \in \mathbb{V} \tag{6.5}$$

$$\sum_{m\in\mathbb{V}} f_{ij}^{mk} - \sum_{m\in\mathbb{V}} f_{ij}^{km} = \begin{cases} -\lambda_{ij}, & k=i \\ \lambda_{ij}, & k=j \\ 0, & \text{otherwise} \end{cases} \quad \forall i,j,k \in \mathbb{V} \tag{6.6}$$

where $f_{ij}^{ml} \geq 0$ is the amount of traffic, from source i to destination j, that is sent on link $m \to l$. The IPC formulation is a multi-commodity problem that computes the maximum allowable α compatible with the constraints. (6.4) minimizes the aggregated power consumption for all the flows and it is obtained from Eq. (6.1) by observing that, $\lambda_{ij} = (\sum_{ml\in\mathbb{V}} f_{ij}^{ml})/h_{ij}$ by construction. Eq. (6.5) is the link capacity constraint when considering the bit expansion factor α. Eq. (6.6) is the flow conservation constraint.

The IPC problem assumes implicitly that traffic flows are splittable and it provides a lower bound to the power consumption when unsplittable flows are considered.

The IPC problem cannot be solved directly due to its complexity. However, when α is fixed, the problem becomes linear, and it can be solved in an approximated fashion by discretizing the interval $[1,\alpha_{\max}]$ in Q points and by solving Q different LP problems through an optimal LP-solver. The best solution among the Q available ones is the approximated optimal solution to the IPC problem. This method cannot be implemented in a real NoC, because the time required for the LP solver is unpredictable, whereas NoCs need to react fast to traffic fluctuations. In the following section, we propose simpler algorithms to find an approximated solution to the IPC problem, based on a load balancing technique that permits to route traffic so as to fully exploit DVFS.

6.4.1 Exploiting DVFS with Load Balancing

Recall that we assume that all NoC links are powered by the same voltage and run at the same frequency, providing the same transfer bandwidth. Intuitively, few

links heavily loaded would constrain to a small α, reducing the power efficiency for all other links with much smaller load. On the contrary, the ideal condition is when all links are equally loaded, because the largest possible α would be optimal for all the links and flows. Hence, to solve the IPC problem, we propose a set of Load Balancing (LB) algorithms that try to equalize the link load among all links, for a given traffic matrix.

As a first step, the LB algorithms identify the minimum size rectangle including the source PE and destination PE for each traffic flow. For example, in Fig. 6.2, the upper-left part is a 2×3 rectangle for the traffic flowing from S_1 to D_1, while bottom-right is a 2×2 square for the traffic from S_2 to D_2. Then, for each rectangle, two different routing paths (XY and YX path) covering the rectangle perimeter are considered. The main reasons for considering these two specific routing paths are:

1. Both are shortest paths, an advantage for power saving purposes because more hops require more power (see Eq. (6.1)).

2. Both are oblivious of the paths of other flows, i.e., they do not adapt to the dynamic state of the other flows.

3. XY and YX paths tend to balance the load on the whole network, better than other routing policies like "negative first, north last" discussed in [87].

4. XY (YX) is dead-lock free without introducing virtual channels. This allows to simplify the queueing structure at each router.

The main drawback of combining XY and YX routing paths is that two paths may deliver packets out-of-order. This problem could be solved at the cost of adding some buffers for reordering, and introducing additional delay. However, in-order delivery only need to be guaranteed at task level. Hence, the flows from one node to another can be split among all the communication flows corresponding to different tasks. Since the degree of "splittability" of a flow depends on the particular distribution of tasks in each node, to find general results we consider different split constraints for LB routing:

- **2P-ES** (2 Paths, Even Split)**:** A traffic flow is evenly splittable among two paths, i.e., each PE contains enough tasks to pack flows evenly into two paths. Thus XY and YX routing paths carry exactly 50% of the original load each.

- **2P-RS** (2 Paths, Random Split)**:** A traffic flow is randomly splittable among two paths, i.e., each flow can be split with a random unbalance factor. This models task unbalancing.

- **4P-ES** (4 Paths, Even Split)**:** A traffic flow is evenly split among four paths. To define the paths, a random intermediate node is chosen internally in the

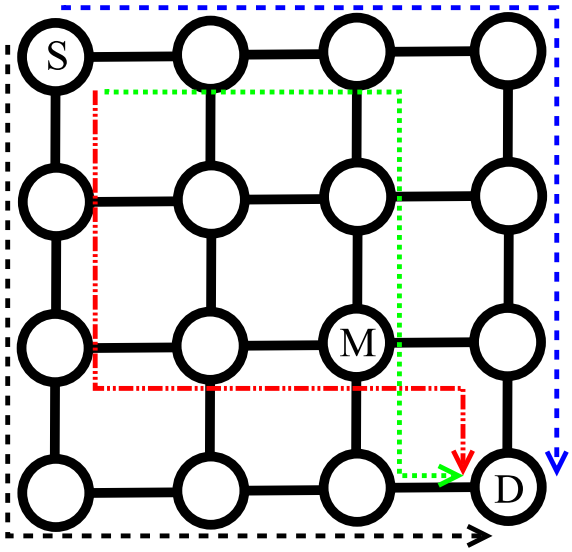


Figure 6.3. Load balancing exploiting 4P-ES routing

rectangle: two paths follow the standard XY, YX routing while the other two are forced to pass through the intermediate node as shown in Fig. 6.3. This algorithm is inspired by the minimum Valiant load balancing [88] technique.

6.5 Simulation and performance evaluation

We developed a flow level NoC simulator to evaluate the whole NoC transmission power cost. We simulated a 5×5 grid topology. To generate the traffic matrix Λ, we considered the following scenarios:

- **Uniform:** All nodes send the same amount of traffic to any other node, i.e. λ_{ij} is a constant for any $i,j \in \mathbb{V}$.
- **Normal:** Λ is obtained as the summation of 25 permutation matrices[1]. By construction $\sum_k \lambda_{ik} = \sum_k \lambda_{kj}$ for any i,j, i.e. all nodes are both source and destination of the same aggregate amount of traffic but the traffic is not uniformly distributed among each node pair.
- **Transpose:** The node in row x and column y (with $x \neq y$) sends traffic to the node in row y and column x.

[1] A permutation matrix is a binary square matrix in which exactly one element is equal to 1 for each row and for each column

- **Tornado:** A node in column x sends traffic to the node in the same row and in column $(x + 2) \mod 5$, i.e., 2 hops to the right (with wrapping).

- **Hot-spot:** Each node sends with probability 0.6 traffic to the hot-spot node located in the center of the topology, and with probability 0.4 uniformly to any other node. This traffic was proposed in [89].

To coherently compare the different scenarios under sustainable traffic for all routing policies, for each scenario we compute the most loaded link across all the routing policies and re-normalize the load to such value, using a parameter $\eta \in [0,1]$, denoted as the *normalized load*. More formally, given a traffic matrix Λ, the offered traffic matrix $\Lambda' = [\lambda'_{ij}]$ for the simulation is computed as:

$$\lambda'_{ij} = \eta \frac{\mu}{\max_{\mathcal{R}}\{\gamma^{\mathcal{R}}_{\Lambda}\}} \lambda_{ij} \tag{6.7}$$

This means that, when $\eta = 1$, there exists a routing policy $\mathcal{R}'$ for which the virtual load is one, i.e. $\gamma^{\mathcal{R}'}_{\Lambda'} = \mu$, and for all the other policies the virtual load is less than one. This guarantees sustainable traffic for any routing policy.

We also evaluated the power consumption when using XY routing with or without DVFS technique, referred as *XY* and *NoDVFS* respectively in the figures. Furthermore, we consider an adaptive routing scheme, derived from the one in [73] and referred as *Adaptive* in the figures. The scheme considers each flow singularly, starting from the largest to the smallest. Each flow is routed by computing, hop-by-hop, all the shortest paths to the destination and choosing, as next hop, the one with the lowest link utilization. Then we compare the NoC power when routing the flows according to the proposed LB schemes: *2P-ES*, *4P-ES* and *2P-RS*. Finally, the minimum power corresponding to the optimal solution of the IPC problem is computed through GLPK [90] solver using the approximated method explained in Sec. 6.4 with $Q = 100$.

Figs. 6.4, 6.5 and 6.6 show the normalized power consumption under normal, transpose and hot spot traffic matrices, respectively. Without DVFS the transmission power grows linearly with the load. On the contrary, the power consumption with DVFS is a cubic function of the load, with a different coefficient depending on the routing algorithm and traffic matrix. This is not surprising for single component (e.g. router, CMOS gate), but it is quite interesting for the whole SVSF NoC chip. Our result validates Property 1 discussed in Sec. 6.3.3.

Clearly a huge gap exists between the minimum power and the XY routing, especially in some traffic patterns like transpose and hotspot. Even under normal traffic scenario, almost 40%[2] power reduction is possible. This suggests that it is

[2]The percentages reported in this Section refer to the maximum load case, i.e. $\eta = 1$. Indeed, when $\eta < 1$, obviously even more power could be saved.

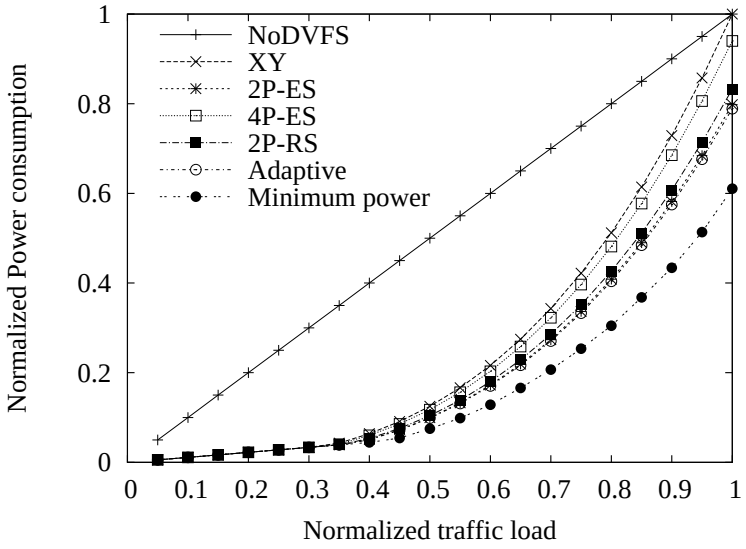


Figure 6.4. Power of a 5×5 NoC under normal traffic pattern

promising to consider DVFS on NoC to save power although the simple XY routing policy shows some limitations. 2P-ES shows power reductions close to the optimal solution in most cases. Under normal traffic, 2P-ES is the best policy among LB techniques, and it saves more than 20% power compared to XY routing with DVFS. Under the transpose traffic patterns, 2P-ES can save up to 78% power, providing savings close to those of the minimum power. The reason for such relevant gain is that XY concentrates the traffic on only one link between the two connecting adjacent nodes, whereas LB is able to distribute the traffic across all the links. Under hot-spot traffic, 2P-ES achieves 60% gain, because it reduces the link load around the hot-spot node.

In general, 2P-ES appears to be close to 2P-RS (within a variation $< 10\%$) and to multiple paths LB 4P-ES, irrespective to the traffic scenarios. The minor performance degradation from 2P-ES to 2P-RS means that if a traffic flow cannot be divided and cannot be evenly sent into two paths due to the lack of tasks, other sub-optimal packing conditions (i.e.random split) can be adopted. The loss due to flow unbalancement is not crucial if compared to the gain obtained by applying LB technique. The multiple paths LB 4P-ES is not always better than the 2 paths LB, because i) XY (YX) is already trying to balance the traffic, ii) random selection of the intermediate node could result in unbalance (i.e. multiple selections of one node), like in normal traffic scenario. Even when 4P-ES outperforms 2P-ES as in the transpose traffic pattern case, the gain is limited to 5%.

Comparing our LB schemes with Adaptive routing, in all considered traffic patterns, deterministic LB routing perform close to the adaptive routing. For instance,

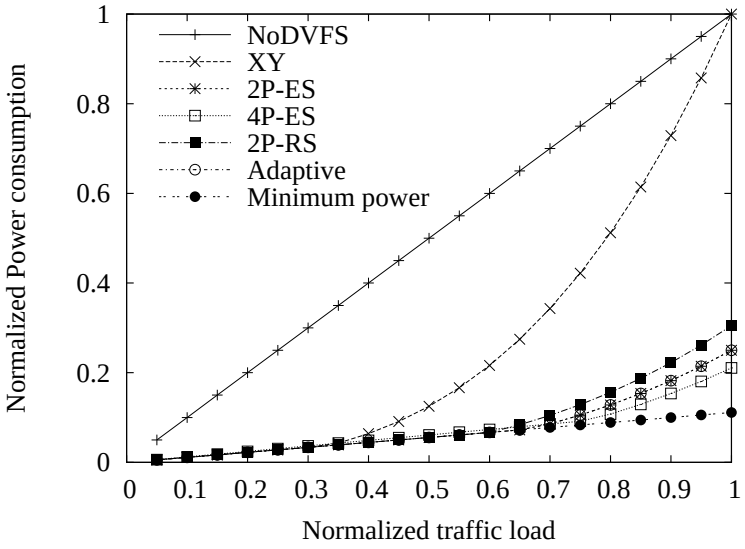


Figure 6.5. Power of a 5×5 NoC under transpose traffic pattern

in Fig. 6.4, 2P-ES and Adaptive behave similarly. In Fig. 6.6, a 6% improvement can be observed when considering Adaptive with respect to 2P-ES, whereas in Fig. 6.5, no improvement can be observed.

Results related to the uniform and tornado traffic scenarios are not reported due to lack of space. Indeed, under uniform traffic pattern, even XY routing balances the traffic and obtains the minimum power. Similarly, 2P-ES and Adaptive find routing paths with minimum power. On the contrary the other two LB routing schemes exhibit performance limitations when the load approaches the maximum (i.e. $\eta = 1$), because the random load balancing reduces the effectiveness of DVFS. Tornado traffic is an adverse traffic pattern, since only one routing path can be found by any shortest-path routing policy per pair. Thus, all the routing policies listed in this chapter, including the optimal one, behaves exactly as NoDVFS.

In summary, our deterministic 2P-ES LB scheme is able to efficiently exploit DVFS and save considerable power for a NoC grid network, close to the minimum achievable power.

6.6 SystemC Verification

For verification purposes, we developed a flit level simulator modeling the flit contentions and the queuing process at the routers and PEs. The simulator is slotted and it is written in C++ adopting SystemC 2.2 and TLM 2.0.1 standard libraries [91], which permit us to build a functional transaction model, cycle accurate

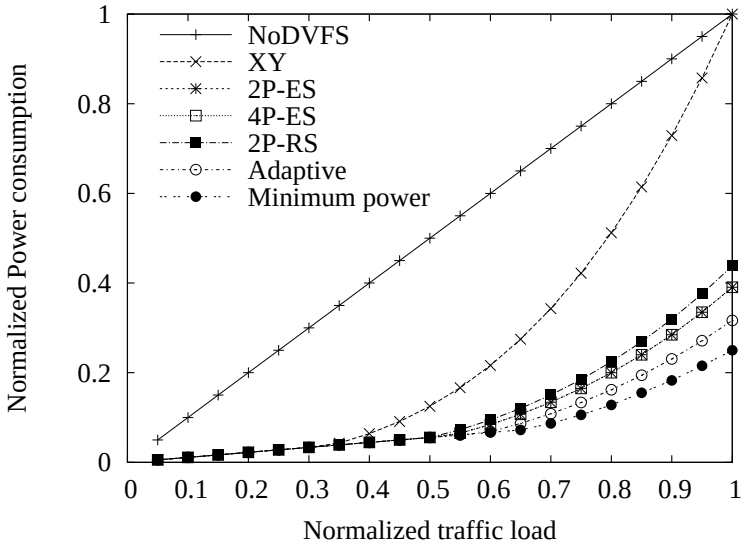


Figure 6.6. Power of a 5×5 NoC under hot-spot traffic pattern

with respect to the real hardware but running much faster than RTL HDL models. We consider input queues with VOQ (Virtual Output Queue) for the routers, with a standard greedy maximal weight matching algorithm to solve contentions.

Two different global clocks were used, f_{NoC} for the NoC (routers and corresponding channels) and f_{PE} for all the PEs; f_{PE} is always fixed at 1 GHz. We consider a network of size 5×5, each VOQ buffer depth is 50 flits with a flit size of 32 bits fixed; the flit duration corresponds to a slot. Packets are generated according to a Bernoulli process, and contain a random number of flits uniformly chosen between 5 to 10. The traffic matrix is hot-spot. To compute the maximum sustainable load, we set $f_{NoC} = f_{PE}$, corresponding to 1 Gbit/s for the link capacity. When NoDVFS is adopted, the maximum sustainable offered load per node is equal to 5.2 Mflit/s ($\rho = 1$). Then, for each traffic load, we slow down the NoC by setting $f_{NoC} = f_{PE}/\alpha$ with $\alpha = \min\{1/\rho, \alpha_{\max}\}$, to saturate the bottleneck link capacity without violating the maximum expansion factor condition.

We only consider XY routing with and without DVFS, under unsplittable traffic. Results are reported in Table 6.1: columns report respectively, i) the traffic load ρ, ii) the power cost of XY routing without DVFS, normalized to the maximum power for $\rho = 1$, iii) the power cost of XY routing with DVFS, iv) their ratio, v) the theoretical ratio obtained with the flow model, that can be shown to be equal to α^2. The normalized power values have been obtained with a precision $< 5\%$ at 95% confidence interval, as an average of 5 individual runs (after removing the transient period). The power savings obtained by introducing DVFS into NoC is observable when traffic load is not saturated as expected. The last two columns show that the

Input load	NoDVFS	DVFS	Actual Ratio NoDVFS/DVFS	Theoretical Ratio NoDVFS/DVFS
1.0	1.00	1.00	1.00	1.00
0.8	0.806	0.513	1.57	1.56
0.6	0.602	0.215	2.86	2.78
0.4	0.412	0.0643	6.40	6.25
0.2	0.201	0.0223	9.02	9.00

Table 6.1. Normalized power cost through SystemC verification under hot spot traffic matrix and XY routing

results obtained from the SystemC simulator agree well with the ones obtained by the flow model discussed in Sec. 6.5. The power gain of DVFS with respect to NoDVFS increases as the input load decreases, before hitting the maximum expansion factor, where the ratio is roughly $\alpha_{\max}^2 = 9$.

6.7 Conclusions

In this chapter, we investigate the power saving problem for NoCs, considering only the dynamic power related to the data transmission due to the bit switching activity. In particular, we exploit DVFS to adjust links speed to match the traffic demands, in a Single Voltage Single Frequency (SVSF) chip. The SVSF makes the approach feasible and promising for hardware implementation.

We model the optimal routing and DVFS problem as an optimization programming problem and solve it through GLPK solver. The optimal solution suggests that considering DVFS into NoC is a promising approach to reduce network transmission power consumption. Furthermore, we propose a set of deterministic load balancing (LB) routing schemes, simple to be implemented if compared to adaptive routing, to balance network traffic and to mitigate the load concentration problem on NoC links. Indeed, a quasi-uniform load on the NoC permits to exploit more efficiently DVFS.

Simulation results show that LB schemes are able to well approximate the performance of adaptive routing and of the optimal solution, but with a lower computational and implementation complexity. Finally, our results have been validated through a flit-level simulator developed with SystemC and TLM libraries.

Chapter 7

Exploiting Space Diversity and DVFS in Multiplane NoC

Network-on-Chips (NoCs) have been proposed as a regular and scalable solution to interconnect multiple components on a silicon chip. In this chapter, we approach NoCs power optimization through Dynamic Voltage and Frequency Scaling (DVFS) under the hypothesis that two NoC *planes* are available, each with a different voltage supply and clock frequency. We show the high potential benefit of applying DVFS independently in each plane. We propose three strategies that allocate the traffic in the two planes and minimize power consumption. We evaluate them through a comparison with an ideal traffic allocation policy based on a linear programming technique. We show that load balancing in the two planes is not aways the best policy. Indeed, in an unbalanced traffic scenario, concentrating the high-load flows in one plane and the remaining low-load flows in the other plane, is more power efficient. Finally, we numerically evaluate the power performance under an accurate NoCs power consumption model.

7.1 Introduction

Network-on-Chips (NoCs) offer a regular and scalable alternative to standard busses for the interconnection of Processing Elements (PEs) in a large-scale System-on-Chip (SoC). Current SoC designs implement aggressive power minimization techniques to stay within a limited power budget. Minimizing the power consumption of each and every SoC component is mandatory, be it a PE, a memory block, or the NoC supporting the traffic between them. Dynamic Voltage and Frequency Scaling (DVFS) is a very effective technique for power optimization. In a previous paper [92] we exploit DVFS and different routing policies to reduce power consumption in single-plane NoCs. Here we consider the multiplane NoC architecture

proposed in [93] and combine it with DVFS to boost NoC power saving without focussing on routing policies. We assume to have two parallel, independent NoC *planes*, as shown in Fig. 7.1. Each PE is connected to two routers, one per plane. Each plane is supplied by a different voltage and clock frequency, to exploit DVFS separately and independently.

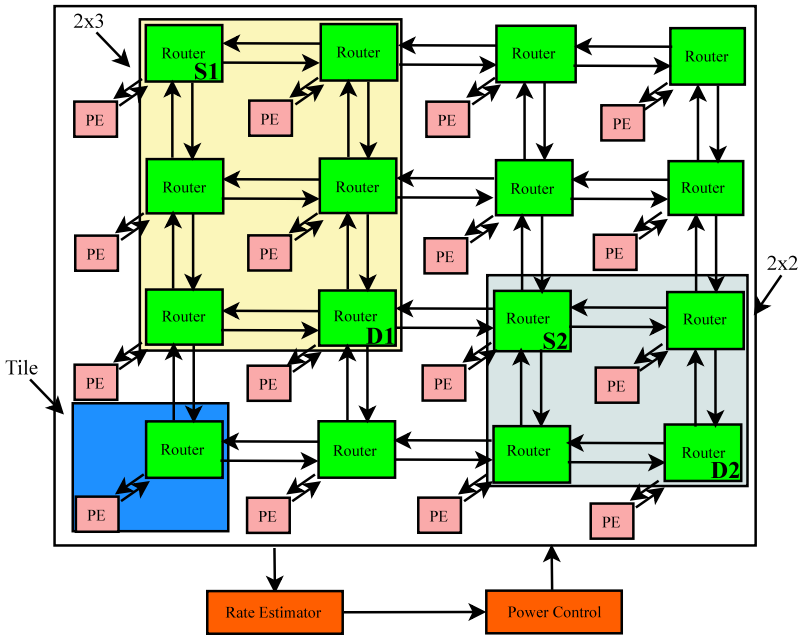


Figure 7.1. A two planes NoC architecture. The interconnection network among the routers in the second plane is the same as in the first plane. Each processing element (PE) is connected to two routers, one for each plane.

In a DVFS setting, clock frequency and supply voltage are jointly reduced when the bit activity is low. This method exploits the dependency of a CMOS gate's dynamic power consumption on the square of supply voltage, rather than its linear dependence on clock frequency:

$$P \propto fV^2 \tag{7.1}$$

In our NoC-based communication framework, clock frequency f is chosen in range $[f_{\min},f_{\max}]$ according to the required average number of bit transitions (from 0 to 1 and vice-versa) per clock period, i.e. the average load. We define $\rho \in [0,1]$ to be such average value and consequently $f = \rho f_{\max}$. Notice that through this definition, clock frequency and bit rate become synonymous. The supply voltage V is chosen in range $[V_{\min},V_{\max}]$. We define α as the voltage reduction factor in such a way that $V = V_{\max}/\alpha$. Due to the voltage lower bound $V_{\min}$, α is also upper bounded by $\alpha_{\max} = V_{\max}/V_{\min}$: $\alpha \in [1,\alpha_{\max}]$. We can now reformulate (7.1) as follows:

$$P \propto fV^2 = \rho f_{\max}(V_{\max}/\alpha)^2 \tag{7.2}$$

Supply voltage and clock frequency are interrelated in a CMOS digital circuit. Given a voltage V, the maximum frequency the circuit can run at is a monotonically increasing function of V: a function which also depends on technology and circuit parameters. When the bit rate decreases, the bit duration, i.e. the clock period in our formulation, increases and the voltage can be decreased. Approximately, when decreasing the voltage by α, the bit duration can be increased by the same factor, and α can be also interpreted as the bit expansion factor: $\alpha = 1/\rho$.

By setting the latter equivalence in (7.2), we get the rule of thumb that dynamic power is a cubic function of average load: $P(DVFS) = \rho^3 f_{\max} V_{\max}^2$ when DVFS is fully exploited[1]. On the contrary, when no voltage and frequency scaling is adopted ($\alpha = 1$), $P(NoDVFS) = \rho f_{\max} V_{\max}^2$ and power scales linearly with ρ. By comparing $P(DVFS)$ and $P(NoDVFS)$, the potential power gain due to DVFS when the load is low is clear.

The motivation for exploiting multiplane NoCs together with DVFS is that the DVFS effectiveness on single plane NoCs is limited by the "bottleneck" link on chip. Indeed, consider a single plane NoC with each link loaded by some amount of traffic, which depends on the applications running on the PEs and on the routing policy. Assume that the whole chip is supplied by a single voltage[2]. The maximum loaded link, the "bottleneck", limits DVFS effectiveness, since the maximum allowed bit expansion factor α is constrained by the load on such link. In a two planes NoC with each plane working at a different voltage, we show that through a proper traffic allocation algorithm it is possible to minimize the impact of bottleneck links and save more power by concentrating most of the bottleneck flows on a single plane.

7.2 Multi-plane NoC model

We consider a mesh with N nodes, one of the most common topologies for NoCs [82] due to its simplicity and low power consumption [83]. Two identical planes are considered, as shown in Figs. 7.1 and 7.2. The network is partitioned into "tiles", and each tile corresponds to one logical PE and two physical routers, one per plane. We assume all the PEs potentially associated with the same router as one logical PE since all the generated/received flows from/by the PEs are routed by the same router. We consider input queuing switch without virtual channels. FIFO buffers of limited size are available in PEs and routers, and wormhole routing is adopted to save buffer space. Each data packet is split into smaller units, called "flits" that are individually routed across the NoC without interleaving. Deterministic *XY routing* is used since deadlock free without virtual channels [87]: data is first routed in the

[1] We have substituted symbol $\propto$ with $=$ assuming a suitable normalization.

[2] We do not consider the case of each link working at an independent voltage as in [77], which is complex to implement, or other sophisticated architectures such as voltage islands [81].

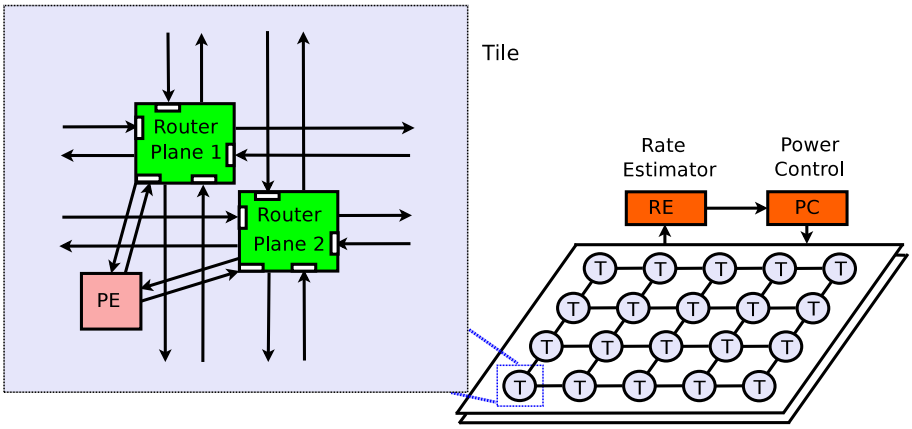


Figure 7.2. The physical implementation of a two planes NoC and one tile architecture

X direction, until reaching the X coordinate of the destination, and then routed in the Y direction. Each flow is transferred across a single plane, to avoid the extra-complexity to route the flow in two planes, possibly with different frequency and voltage pairs. Our approach doubles network resources, but compensates this cost with a high power saving, as shown in Sec. 7.4.

7.2.1 Power Model

All links in a plane are supplied with a unique voltage and frequency pair, similar to a Single Voltage/Single Frequency (SVSF) chip [72], but two planes can work at different voltages chosen in range $[V_{\min},V_{\max}]$ and at different frequencies in range $[f_{\min},f_{\max}]$. The extreme values for frequency and voltage depend on the adopted technology and chip design. We focus only on the minimization of dynamic power due to the data transferred between routers, neglecting leakage power consumed when routers are idle. The power model at the network level is hop based; such model was proposed and validated in the literature [84, 85]. When transmitting continuously at rate r bit/s along a path of h hops, from (7.1) and from the equivalence between bit rate and clock frequency, the power consumption is proportional to rhV^2. To be admissible, the flow cannot overload the links along its path. This implies that $r \leq f_{\max}$ and the normalized load is defined as $\rho = r/f_{\max}$. To fully exploit DVFS and to avoid any throughput degradation, the bit duration can be expanded (at most) by a factor of $\alpha = 1/\rho = f_{\max}/r$, which has been previously defined as the expansion factor; thus, the voltage can be decreased by α, as we already noted. Using (7.2), the total power for transmitting a flow with rate r across h hops is

$$P = rh\left(\frac{V_{\max}}{\alpha}\right)^2 \propto \frac{hr}{\alpha^2}. \tag{7.3}$$

Given the minimum allowed voltage $V_{\min}$ in the considered technology, it must be $\alpha \leq \alpha_{\max}$ where $\alpha_{\max} = V_{\max}/V_{\min}$ corresponds to the maximum expansion factor. Note that $\alpha_{\max}$ is often around 2 and never more than 3 [86]. In addition to such high-level model of power consumption, we have considered a more realistic model that refers to an Intel 48 cores chip [94] and has been obtained through Orion 2.0 [95], an accurate simulator of NoC router power. This validation is discussed in details in Sec. 7.5.

7.2.2 Traffic Model

We assume that the traffic flows among the N PEs are known. Depending on the actual application, such flows can be either known in advance, or estimated on-line by the rate estimator shown in Fig. 7.2. The average traffic flow from node i to j is denoted by r_{ij}, measured in [bit/s]. All the links have maximum capacity μ [bit/s], which is achievable only for maximum frequency $f_{\max}$ and maximum voltage $V_{\max}$.

Definition 1. *Let $\Lambda = [\lambda_{ij}]$ be the $N \times N$ traffic matrix, in which λ_{ij} is the normalized traffic rate from node i to j, defined as $\lambda_{ij} = r_{ij}/\mu$ with $\lambda_{ij} \in [0,1]$.*

Definition 2. *Given a routing policy $\mathcal{R}$ and traffic matrix Λ, $\gamma_\Lambda^\mathcal{R}$ is defined as the bottleneck load, i.e. the maximum offered load, normalized to ρ, among all the edges in the topology.*

Definition 3. *Λ is said to be* admissible *according to the routing policy $\mathcal{R}$ iff $\gamma_\Lambda^\mathcal{R} \leq 1$.*

Since in-sequence delivery of messages belonging to same flows is crucial to avoid complex re-ordering functionality and to reduce the memory requirements, we do not consider splittable flows in our work, i.e, λ_{ij} is routed along one single path on a single plane. Even though, under this assumption it is not possible to get the minimum power that a splittable policy would achieve, as we show later, it is still possible to save considerable power. We will compare our traffic allocation policies with an ideal one that allow splittable flows and allocates flows by solving an optimization problem. The power obtained by such benchmark strategy can be then taken as the lower bound.

7.3 Traffic Allocation for Two-Planes NoC

The objective of our power control is to allocate the traffic for the two-planes NoC, in order to minimize the total power consumption while satisfying the traffic demands and the link capacity constraints. In the following section we describe a toy-scenario in which it is possible to highlight the potential power gain due to different traffic allocation algorithms.

7.3.1 A toy scenario

To fully exploit multiplane NoC for power saving, one option could be to load balance the traffic among the two NoCs. Contrary to common belief, we show that this policy is not always optimal.

As an example scenario, consider two NoC planes, supplied with voltages $V_1 = V_{\max}/\alpha_1$ and $V_2 = V_{\max}/\alpha_2$. As a reminder, all the links on plane i (with $i = 1,2$) run at a maximum frequency $f_{\max}$ when $\alpha_i = 1$. Assume to have $k+1$ traffic flows among routers that are adjacent in the topology, thus all the flows do not share any link. The first flow is at rate $f_{\max}$ bit/s and is denoted as "max-rate flow". All the other flows are at rate $\rho f_{\max}$ bit/s, with some small $\rho \in (0,1)$, and are denoted as "low-rate flows". To emphasize the possible power gains due to DVFS, we assume $\alpha_{\max} = 3$, coherently with [86].

Now consider the following traffic allocation schemes:

- Route all the $k+1$ flows in the first plane. In this case, DVFS cannot be exploited because of the max-rate flow on the bottleneck link. Hence, $\alpha_1 = 1$ and, thanks to (7.3), the overall power consumption is

$$\begin{aligned} P^{(1)} &= \frac{f_{\max}}{\alpha_1^2} V_{\max}^2 + \frac{k\rho f_{\max}}{\alpha_1^2} V_{\max}^2 \\ &= (1 + k\rho) P_0 \end{aligned} \tag{7.4}$$

  having defined $P_0 = f_{\max} V_{\max}^2$.

- Balance the traffic among the two planes, assuming that flows can be split across the two planes. Thus the power is an optimistic lower bound of the actual value achievable with any load balancing scheme that do not allow flow splitting. The bottleneck link is transferring $0.5 f_{\max}$ bit/s per plane and the DVFS can be fully exploited by setting $\alpha_1 = \alpha_2 = 2$. According to (7.3), the overall power consumption is

$$\begin{aligned} P^{(2)} &= \frac{0.5 + k\rho/2}{\alpha_1^2} P_0 + \frac{0.5 + k\rho/2}{\alpha_2^2} P_0 \\ &= (1 + k\rho) \frac{P_0}{4} \end{aligned} \tag{7.5}$$

- Concentrate the max-rate flow on the first plane and allocate all the other small-rate flows on the other plane. Hence, $\alpha_1 = 1$ and $\alpha_2 = \min\{1/\rho, \alpha_{\max}\}$,

thus the overall power consumption becomes:

$$\begin{aligned} P^{(3)} &= \frac{1}{\alpha_1^2}P_0 + \frac{k\rho}{\alpha_2^2}P_0 \\ &= P_0 + k\rho\frac{P_0}{(\min\{1/\rho,\alpha_{\max}\})^2} \\ &= \left(1 + k\max\left\{\rho^3, \frac{\rho}{\alpha_{\max}^2}\right\}\right)P_0 \end{aligned} \tag{7.6}$$

Note that, in a mesh topology with N nodes, the number of links is in the order of N^2; hence, in our toy scenario k grows as fast as N^2. Therefore, for large enough N, (7.4)-(7.6) can be approximated by:

$$\begin{aligned} P^{(1)} &\approx k\rho P_0 \\ P^{(2)} &\approx k\rho P_0/4 \\ P^{(3)} &\approx k\max\{\rho^3, \rho/\alpha_{\max}^2\}P_0 \end{aligned}$$

From the results above, by comparing $P^{(2)}$ and $P^{(3)}$ with $P^{(1)}$, it is clear the power reduction due to the DVFS. Instead, when comparing $P^{(2)}$ with $P^{(3)}$, the third policy is better than load-balancing when $\rho < 0.5$ and $\alpha_{\max} > 2$, even if the load-balancing is allowing flow splitting. In other words, contrary to some common belief, *to exploit fully DVFS and minimize power, load balancing across the planes is not always the optimal strategy.* Intuitively, it is better to "concentrate" all the high-rate flows in one plane, for which the voltage is kept at maximum, and route all the small-rate flows in the other plane, that runs at a lower voltage and fully exploits DVFS.

7.3.2 Traffic Allocation Algorithms

Inspired by the toy scenario above, we consider three algorithms to allocate flows according to different criteria. The first algorithm, denoted as 2P-Balance, balances the traffic flows between the two planes. Then we choose the most convenient frequency and voltage for each plane according to the resulting bottleneck load. As discussed in Sec. 6.3, no flow splitting is allowed. The corresponding pseudo code is Algorithm 1.

Let Ω be the set of all the flows, identified by a couple (i,j) for the source PE i and the destination PE j. Let Ω_1 and Ω_2 be the set of the flows that have been allocated to plane 1 and 2, respectively. They are the outputs of the allocation scheme, together with the corresponding expansion factors α_1 and α_2. The input for the algorithm is the normalized traffic matrix Λ. The algorithm starts to consider all the flows in first plane ($\Omega_1 = \Omega$, $\Omega_2 = \emptyset$), which is called the *Master Plane (MP).* Incrementally, the algorithm considers all the flows contributing to bottleneck link

Algorithm 1: 2P-BALANCE

Input: Traffic matrix Λ
Output: Ω_1, Ω_2, α_1, α_2

1: $\Omega = \Omega_1 = \{(i,j), \forall i,j\}$, $\Omega_2 = \emptyset$
2: $\mathbb{S} = \mathtt{BF}(\Omega_1) \cap \Omega$
3: **while** $\mathbb{S} \neq \emptyset$ **do**
4: $(i,j) = \arg\max_{(i',j') \in \mathbb{S}} \{\lambda_{i'j'}\}$
5: **if** $\mathtt{BL}(\Omega_1 \setminus \{(i,j)\}) \geq \mathtt{BL}(\Omega_2 \cup \{(i,j)\})$ **then**
6: $\Omega_2 = \Omega_2 \cup \{(i,j)\}$, $\Omega_1 = \Omega_1 \setminus \{(i,j)\}$
7: **end if**
8: $\Omega = \Omega \setminus \{(i,j)\}$, $\mathbb{S} = \mathtt{BF}(\Omega_1) \cap \Omega$
9: **end while**
10: $\alpha_1 = \mu/\mathtt{BL}(\Omega_1)$, $\alpha_2 = \mu/\mathtt{BL}(\Omega_2)$

in MP and evaluates the load of the new bottleneck link if each flow was moved in the second plane, named as the *Slave Plane (SP)*. In the pseudo code, function BL returns the load corresponding to the bottleneck link, whereas BF returns the set of all the flows contributing to the bottleneck links, either on MP or SP depending on whether the argument is Ω_1 or Ω_2. Every time a bottleneck flow has been considered for being moved to SP, it is removed from Ω, to avoid further consideration in the following iterations of the algorithm. The expansion factors α_i for each plane are computed as $\alpha_i = \mu/\mathtt{BL}(\Omega_i)$, since the bottleneck load is the only one affecting the DVFS. $\mathbb{S}$ is the set for the flows contributing to the bottleneck link.

Remember, since splittable flows are not allowed, algorithm 2P-BALANCE usually can not perfectly balance the load between the two planes. The algorithm complexity is $\mathcal{O}(N^2 log(4N))$. In worst case we need to consider each commodity (N^2) once, with the need to update link load order once for each commodity ($4N$ links in total and $\mathcal{O}(log(4N))$ to update the order in heap structure).

Whereas 2P-BALANCE tends to distribute the flows among the two planes, the second algorithm we proposed, denoted as 2P-MINI, concentrates the flows with higher traffic into *MP* while the flows with lower traffic in *SP*. The pseudo code of 2P-MINI is described in Algorithm 2.

The key difference for this algorithm compared to 2P-BALANCE is that we force the bottleneck load in SP to be low (i.e., $\mathtt{BL}(\Omega_2) \leq 1/\alpha_{\max}$)), to guarantee the SP running at the minimum possible frequency $f_{\min}$ and exploit fully DVFS in at least one of the two planes. On the contrary, 2P-BALANCE tends to equalize the bottleneck load among the two planes MP and SP (i.e., $\mathtt{BL}(\Omega_1) \approx \mathtt{BL}(\Omega_2)$). In the pseudo code, the additional loop in 2P-MINI (lines 10-16) considers the flows in MP that have not been considered in the first loop (lines 3-9). Since the first loop considers only flows contributing to the bottleneck link, it is still possible to move

Algorithm 2: 2P-MINI

Input: Traffic matrix Λ
Output: Ω_1, Ω_2, α_1, α_2

1: $\Omega = \Omega_1 = \{(i,j), \forall i,j\}$, $\Omega_2 = \emptyset$
2: $\mathbb{S} = \mathtt{BF}(\Omega_1) \cap \Omega$
3: **while** $\mathbb{S} \neq \emptyset$ **do**
4: $(i,j) = \arg\max_{(i',j') \in \mathbb{S}} \{\lambda_{i'j'}\}$
5: **if** $\mathtt{BL}(\Omega_2 \cup \{(i,j)\}) \leq 1/\alpha_{\max}$ **then**
6: $\Omega_2 = \Omega_2 \cup \{(i,j)\}$, $\Omega_1 = \Omega_1 \setminus \{(i,j)\}$
7: **end if**
8: $\Omega = \Omega \setminus \{(i,j)\}$, $\mathbb{S} = \mathtt{BF}(\Omega_1) \cap \Omega$
9: **end while**
10: **while** $\Omega \neq \emptyset$ **do**
11: $(i,j) = \arg\max_{(i',j') \in \Omega} \{\lambda_{i'j'}\}$
12: **if** $\mathtt{BL}(\Omega_2 \cup \{(i,j)\}) \leq 1/\alpha_{\max}$ **then**
13: $\Omega_2 = \Omega_2 \cup \{(i,j)\}$, $\Omega_1 = \Omega_1 \setminus \{(i,j)\}$
14: **end if**
15: $\Omega = \Omega \setminus \{(i,j)\}$
16: **end while**
17: $\alpha_1 = \mu/\mathtt{BL}(\Omega_1)$, $\alpha_2 = \mu/\mathtt{BL}(\Omega_2)$

additional flows from MP to SP. This allows to load the SP as much as possible and better exploit DVFS.

Although 2P-MINI has an additional phase to consider the remaining flows which do not contribute to $\gamma^{\Lambda}_{1\mathcal{R}}$, from step 11 to 16 in the algorithm description, the complexity is still $\mathcal{O}(N^2 log(4N))$ since the additional loop is only a constant factor (less than 2) for the complexity.

The last algorithm we proposed is 2P-4PHASE and it is an extension of 2P-MINI. Indeed, after some preliminary tests, we noticed that 2P-MINI allocates too many flows in MP, and they can dominate the total power cost. To improve the power reduction, we propose 2P-4PHASE, that computes explicitly the power cost for each flow according to the formula in (7.3), to better choose which flows to move from MP to SP after running 2P-MINI. As the name suggests, 2P-4PHASE consists of 4 phases:

P1) Move a flow in $\mathtt{BF}(\Omega_1)$ to SP if $\mathtt{BL}(\Omega_2) \leq 1/\alpha_{\max}$.

P2) Move a flow in Ω_1 to SP if both $\mathtt{BL}(\Omega_1)$ and $\mathtt{BL}(\Omega_2)$ do not change.

P3) Move a flow in $\mathtt{BF}(\Omega_1)$ to SP that increases the value of $\mathtt{BL}(\Omega_2)$ and the SP power cost, but the power increase is lower than the power decrease in MP.

Algorithm 3: 2P-4PHASE

Run Algorithm 2
Calculate W_1 and W_2, update $\gamma^{\Lambda}_{1\mathcal{R}}$, $\gamma^{\Lambda}_{2\mathcal{R}}$, $\mathbb{S}_1$
$Pw_1 = g(\gamma^{\Lambda}_{1\mathcal{R}}, W_1)$, $Pw_2 = g(\gamma^{\Lambda}_{2\mathcal{R}}, W_2)$
repeat
  $\lambda^1_{\max} = \max(\lambda_{ij} \in \mathbb{S}_1)$
  Send $\lambda^1_{\max}$ to $\mathtt{P}_2$, route it according to $\mathcal{R}$
  $W_1 = W_1 - \lambda^1_{\max}, W_2 = W_2 + \lambda^1_{\max}$
  Update $\gamma^{\Lambda}_{1\mathcal{R}}$ and $\gamma^{\Lambda}_{2\mathcal{R}}$.
  $Pw'_1 = g(\gamma^{\Lambda}_{1\mathcal{R}}, W_1)$, $Pw'_2 = g(\gamma^{\Lambda}_{2\mathcal{R}}, W_2)$
  if $P'_1 + P'_2 > P_1 + P_2$ **then**
    Move back $\lambda^1_{\max}$ to $\mathtt{P}_1$, $\mathbb{S}_1 = \mathbb{S}_1 - \lambda^1_{\max}$
    $W_1 = W_1 + \lambda^1_{\max}, W_2 = W_2 - \lambda^1_{\max}$
  else
    $\mathbb{S}_1 = f(\gamma^{\Lambda}_{1\mathcal{R}})$, $Pw_1 = Pw'_1$,$Pw_2 = Pw'_2$
  end if
until $\mathbb{S}_1 = \varnothing$
while $\mathbb{M} \neq \varnothing$ **do**
  $\lambda^1_{\max} = \max(\lambda_{ij} \in \mathbb{M})$
  Send $\lambda^1_{\max}$ to $\mathtt{P}_2$, route it according to $\mathcal{R}$
  $W_1 = W_1 - \lambda^1_{\max}, W_2 = W_2 + \lambda^1_{\max}$
  Update $\gamma^{\Lambda}_{1\mathcal{R}}$ and $\gamma^{\Lambda}_{2\mathcal{R}}$.
  $Pw'_1 = g(\gamma^{\Lambda}_{1\mathcal{R}}, W_1)$, $Pw'_2 = g(\gamma^{\Lambda}_{2\mathcal{R}}, W_2)$
  if $P'_1 + P'_2 > P_1 + P_2$ **then**
    Move back $\lambda^1_{\max}$ to $\mathtt{P}_1$, $\mathbb{M}_1 = \mathbb{M}_1 - \lambda^1_{\max}$
    $W_1 = W_1 + \lambda^1_{\max}, W_2 = W_2 - \lambda^1_{\max}$
  else
    $\mathbb{M}_1 = \mathbb{M}_1 - \lambda^1_{\max}$, $Pw_1 = Pw'_1$,$Pw_2 = Pw'_2$
  end if
end while

P4) Move a flow in Ω_1 but not in $\mathtt{BF}(\Omega_1)$ to SP, that increases the value of $\mathtt{BL}(\Omega_2)$ and the SP power cost, but the power increase is lower than the power decrease in MP.

In order to push the power consumption towards an efficient direction, the first two phases are desirable without additional conditions. The latter two are not clear without introducing extra merit. To cope with this, We insert an analytical power cost formula shown as (7.3), to decide whether a movement is successful or not. After running our 4PHASE algorithm, there is not possible to move a single traffic flow to decrease the whole chip power consumption. This algorithm outperforms than the other two as shown later in next section.

The persudo code of 4PHASE is shown in Algorithm 3, $W_i = \sum_{\lambda_{ij} \in \Lambda}(\lambda_{ij} h(\lambda_{ij}))$ is the weight of plane i, $Pw_i = W_i/(\alpha_i)^2$ is the power cost for plane i, and this calculation is denoted as function g. It is coherent with (7.3) but with less updating complexity since new power calculation is only based on the weight change, i.e, no need to calculate the entire chip power from scratch. We use 2P-MINI as the first two phases and for the following 2 phases, the movement condition is based on the power change directly: after the movement of the selected flow, the aggregate power decrease suggests a success move whereas the power increase means a failure. The final complexity for this algorithm is $\mathcal{O}(N^2\sqrt{N}log(N))$ since the additional operations compare to algorithm 2P-BALANCE are updating W_i and α_i, with complexity $\mathcal{O}(1)$ and $\mathcal{O}(log(N))$ respectively under proper data structure (i.e heap).

To evaluate the performance of the algorithms, we also consider an Ideal Power Control (IPC) that allows flow splitting across the two planes and finds the optimal routing that minimize the power cost. The corresponding problem is quadratic and is formalized as follows:

$$\min_{1 \leq \alpha \leq \alpha_{\max}, f_{ij}^{ml} \geq 0} \quad \sum_{i,j,m,l \in \mathbb{V}} f_{ij}^{ml} \frac{1}{\alpha_1^2} + \sum_{i,j,m,l \in \mathbb{V}} g_{ij}^{ml} \frac{1}{\alpha_2^2} \tag{7.7}$$

subject to:

$$\sum_{i,j \in \mathbb{V}} f_{ij}^{ml} \alpha_1 \leq \mu \quad \forall m,l \in \mathbb{V} \tag{7.8}$$

$$\sum_{i,j \in \mathbb{V}} g_{ij}^{ml} \alpha_2 \leq \mu \quad \forall m,l \in \mathbb{V} \tag{7.9}$$

$$\sum_{m \in \mathbb{V}} f_{ij}^{mk} - \sum_{m \in \mathbb{V}} f_{ij}^{km} = \begin{cases} \lambda_{ij}^{MP}, & k = i \\ -\lambda_{ij}^{MP}, & k = j \\ 0, & \text{otherwise} \end{cases} \quad \forall i,j,k \in \mathbb{V} \tag{7.10}$$

$$\sum_{m \in \mathbb{V}} g_{ij}^{mk} - \sum_{m \in \mathbb{V}} g_{ij}^{km} = \begin{cases} \lambda_{ij}^{SP}, & k = i \\ -\lambda_{ij}^{SP}, & k = j \\ 0, & \text{otherwise} \end{cases} \quad \forall i,j,k \in \mathbb{V} \tag{7.11}$$

$$\lambda_{ij}^{MP} + \lambda_{ij}^{SP} = \lambda_{ij} \quad \forall i,j \in \mathbb{V} \tag{7.12}$$

where $f_{ij}^{ml} \geq 0$ is the amount of traffic, from source node i to destination node j, sent on link $m \to l$ in MP; similarly, g_{ij}^{ml} refers to SP. $\mathbb{V}$ is the set of the nodes. Eqs. (7.8)-(7.9) model the maximum bit expansion compatible with the bottleneck load in each plane, in order to guarantee the maximum throughput. Eqs. (7.10)-(7.11) are the classical flow conservation constraints. Finally, Eq. (7.12) guarantees to serve all the traffic in one or both the two planes.

The IPC problem provides a lower bound on the power consumption when unsplittable flows are considered. The IPC problem cannot be solved directly due to the quadratic objective and large multi-commodity size. However, when we relax the objective by fixing α_i, the problem becomes linear. This fact permits to solve the problem in approximated fashion by discretizing the interval $[1,\alpha_{\max}]$ in Q points and by solving Q different LP problems through an optimal LP-solver. The best solution among the Q available ones is the approximated optimal solution to the IPC.

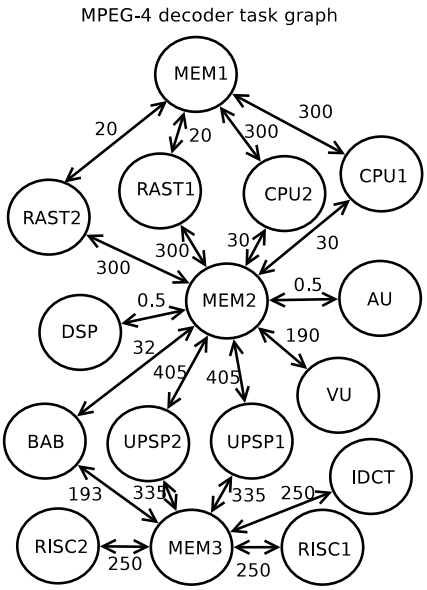
7.4 Performance Evaluation

We developed a flow level NoC simulator to evaluate the whole NoC transmission power cost. We simulated a two-planes mesh network of size 4×4 (for the MPEG4 decoder scenario with $N = 16$ nodes) and of size 5×5 (for all the other scenarios with $N = 25$ nodes). We generated the traffic matrix Λ according to the following scenarios:

- **Uniform:** All nodes send the same amount of traffic to any other node, i.e. λ_{ij} is constant for any pair of nodes.
- **Normal:** Λ is obtained as the summation of N permutation matrices[3]. By construction $\sum_k \lambda_{ik} = \sum_k \lambda_{kj}$ for any i,j, i.e. all nodes are both source and destination of the same aggregate amount of traffic but the traffic is not uniformly distributed among each node pair.
- **Tornado:** A node in column x of the mesh sends traffic to the node in the same row and in column $(x + 2) \mod 5$, i.e., two hops to the right (with wrapping).
- **Hot-spot:** Each node sends with probability 0.6 traffic to an hot-spot node located in the center of the topology, and with probability 0.4 uniformly to any other node. This traffic was proposed in [89].
- **MPEG4 Decoder:** The traffic matrix is derived by a real MPEG4 decoder task graph, shown in [1] and mapped to a 4×4 mesh network according to Fig. 7.3.

To coherently compare the different scenarios under admissible traffic, for each scenario we compute the most loaded link in the case that all the flows are allocated to a single plane and are routed according to XY routing. Then, we re-normalize

[3]A permutation matrix is a binary square matrix in which exactly one element is equal to 1 for each row and for each column



Network on Chip architecture

BAB (3,0)	IDCT (3,1)	RISC1 (3,2)	RISC2 (3,3)
AU (2,0)	UPSP1 (2,1)	MEM3 (2,2)	CPU2 (2,3)
VU (1,0)	MEM2 (1,1)	UPSP2 (1,2)	CPU1 (1,3)
DSP (0,0)	RAST1 (0,1)	RAST2 (0,2)	MEM1 (0,3)

Figure 7.3. On the left, the MPEG4 decoder task graph (derived from [1]) with the traffic demand among PEs, expressed in normalized rate units. On the right, it is shown the mapping of each PE into a 4×4 mesh NoC architecture.

the load to such value, using a parameter $\rho \in [0,1]$, denoted as the *normalized load*. More formally, given a traffic matrix Λ, the offered traffic matrix $\Lambda' = [\lambda'_{ij}]$ for the simulation is computed as:

$$\lambda'_{ij} = \rho \frac{\mu}{\gamma_{\Lambda}^{XY}} \lambda_{ij} \tag{7.13}$$

where γ_{Λ}^{XY} is the bottleneck load when all the traffic is routed according to XY routing on a single plane. This definition implies that when $\rho = 1$, a naive XY routing without DVFS will saturate at least one link. For a fair comparison, values of $\rho > 1$ will not be considered since the traffic is not sustainable on a single plane.

We evaluated the power consumption under XY routing on a single plane, with or without DVFS technique, referred as *XY DVFS* and *NoDVFS* respectively in the figures. Then we compare the power cost with the proposed traffic allocation algorithms when two planes are considered, namely 2P-Balance, 2P-Mini and 2P-4Phase, respectively.

To provide a lower bound on the minimum achievable power, we show also the power corresponding to the solution of the linear programming IPC problem, which was computed through the combination of the GLPK [90] solver and the approximated method explained in Sec. 7.3 by setting $Q = 100$; this guarantees a precision around 1% on the optimal values found for α_1 and α_2. The corresponding curves will be denoted as Minimum Power.

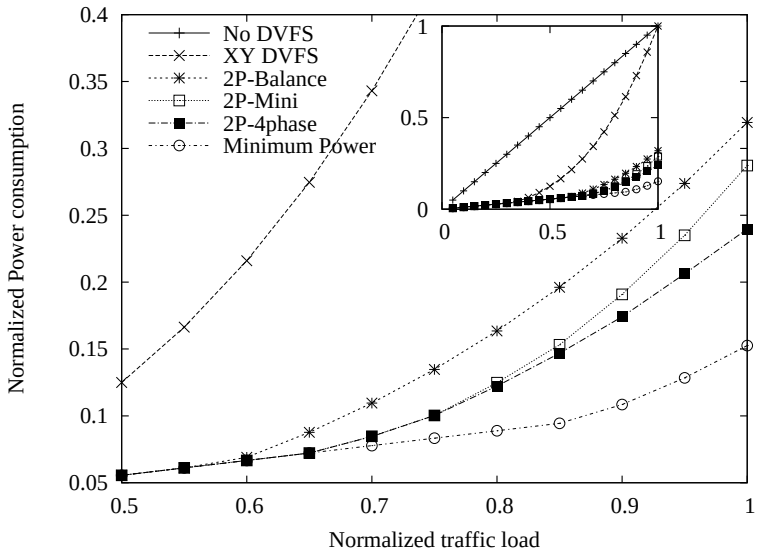


Figure 7.4. Power consumption of 5×5, double plane, mesh network under normal traffic pattern and different loads.

The results are shown in Fig.s 7.4-7.6, with each one corresponding to a different traffic pattern described above. We change the seeds for the traffic patterns generation each time before running the simulation and we average the results in 10 different runs for each traffic pattern. In general, the results suggest that with DVFS, the NoCs power scales down cubicly as the traffic load decreases, when comparing NO DVFS with XY DVFS for a single plane. This result is not surprising for a single hardware component (i.e, CPU, router) with DVFS, but it is quite interesting for the whole chip transmission power cost. Indeed, it can be proved that:

Property 2. *Assume ideal DVFS with unbounded* $\alpha_{\max}$ *(i.e.,* $V_{\min} = 0$*). Given a routing policy* $\mathcal{R}$ *and a sustainable traffic, then the overall power cost of the NoC is a cubic function of the normalized load* ρ*:* $P_{tot}(\rho) = P_{tot}^{\max}\rho^3$*, for* $0 < \rho \leq 1$*.*

The proof has been shown in the previous chapter.

Comparing the results for a single plane and for two planes, it is obvious that the power is lower for the two planes, independently from the proposed algorithms, since we exploit an additional plane and double the switching resources. According to Property 2, by halving the load ρ thanks to the two planes, we would expect a power proportional to $(\rho/2)^3$ for each plane, which implies $\rho^3/4$ for the two planes; so a perfect load balancing (with flow splitting) would allow to achieve a power reduction of factor 4. Interestingly, the observed gain can be larger than 4 using some of our proposed algorithms. Note that the lower bound provided by Minimum Power shows the maximum range of power gain due to a two planes NoC, which

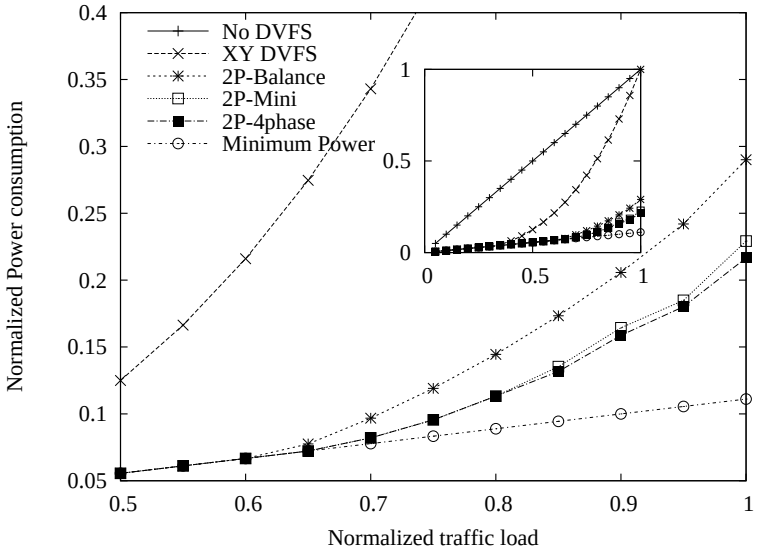


Figure 7.5. Power consumption of 5×5, double plane, mesh network under hot-spot traffic pattern and different loads

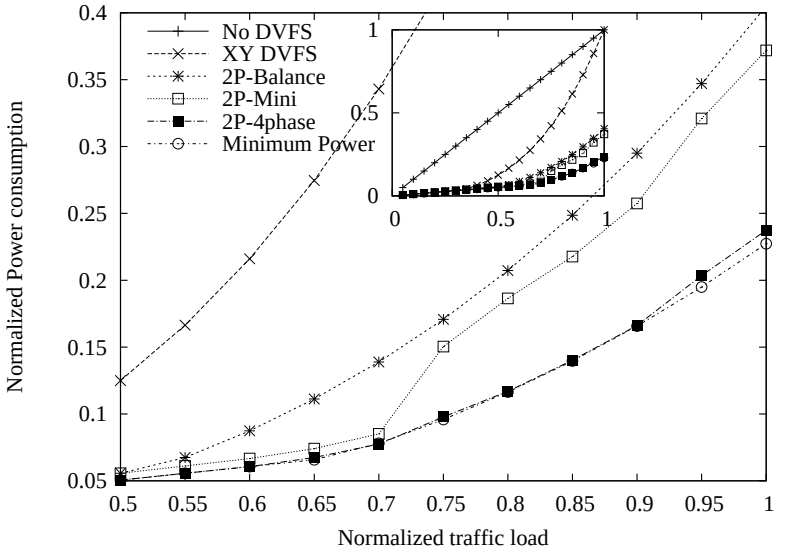


Figure 7.6. Power consumption of 4×4, double plane, mesh network under MPEG4 decoder traffic

can reach also a factor of 6 to 9 with respect to a single plane. Especially in Fig. 7.5, under unbalanced traffic the minimum achievable power is almost approaching the maximum achievable gain $\alpha^2_{\max} = 9$. These promising results show the great potential of multiplane NoCs to reduce the power, and our proposed algorithms are

devised to exploit it.

Regarding the traffic allocation algorithms, the results obtained with 2P-Balance and 2P-Mini show that load concentration is better than load balancing, and the more unbalanced traffic is, the larger performance gap between the two algorithms can be observed. Indeed, since the traffic cannot be split for both policies, 2P-Balance does not balance the traffic between the two planes perfectly. The plane with a higher bottleneck link could accommodate a larger number of minor flows, increasing the total power cost. Instead, 2P-Mini is able to allocate unsplittable flows more efficiently, as long as there are high load and low load links, since 2P-Mini can distribute them into different planes and save more power, as also shown in the toy scenario from Sec. 7.3. Coherently, in Fig. 7.5, 2P-Mini saves a factor of 4.4 in the power for $\rho = 1$, compared to the case without DVFS, better than the perfect load balancing, for which the gain would be 4. This is because hot-spot traffic pattern is very unbalanced and the link load among the hot-spot node is very high whereas the link load "far away" from the hot-spot node is much lower. In general any hot-spot scenario (which is quite realistic) tends to highlight the beneficial effects of load concentration to save power.

Algorithm 2P-4Phase is devised to exploit the space diversity and load concentration more efficiently. Indeed, Fig.s 7.4, 7.5 and 7.6 show a power reduction factor of 4.2, 4.7 and 4.2, respectively, better than the other two proposed algorithms.

We did not include the results about the uniform and tornado traffic patterns, due to the lack of space. Indeed, simulation results show that both tornado and uniform patterns are not suitable for load concentration since all link loads are exactly the same for both patterns. Even worse, the minimum achievable power obtained from Minimum Power is at most 4 times lower than the single plane case; this suggests that no algorithm is able to further exploit the two planes to gain more than a factor of 4.

As a summary, the simulation results show that given a two planes NoC architecture with each one runs its own clock frequency for all the transmission links, load concentration is better than load balancing when the traffic pattern is unbalanced. Our algorithm 2P-4Phase appears to be the best one to exploit the load concentration efficiently for power saving.

7.5 Accurate Power Model Validation

To further back up our methodology we picked two remarkable NoC examples from the recent literature.

The first network is a mesh used by Intel designers as communication backbone for a chip that integrates 80 cores in a 65 nm CMOS technology [96]. Its 8×10 2D mesh topology utilizes a 5-port input-buffered router based on wormhole

switching. Routers are connected through 39-bit unidirectional 2 mm point-to-point links. They feature two logical lanes for deadlock-free routing with 16-flits queues and a non-blocking crossbar. Arbitration, input-output matching and queue reading are arranged in a 5-stage pipeline scheme. Flow control and buffer management between routers are debit-based and use almost-full bits. The router operates at 5.6 GHz at 1.3 V.

The second network connects a pool of 48 cores in a 45 nm chip, again by Intel [94], organized in a 2D mesh of 24 tiles (6×4, two cores per tile). Routers with five ports and with a non-blocking crossbar are used in this design too, with bidirectional 144-bits, 5.4 mm point-to-point links. Virtual cut-through switching was chosen with eight virtual channels over 2 message classes (request and response) together with a classic dimension ordered XY routing. The pipeline scheme is a 4-stage one. A credit-based flow control is used which exerts back pressure on the upstream routers to prevent buffer overflow. The clock frequency is 2.35 GHz at 1.25 V, significantly smaller than the previous case, despite the scaled technology, because of the larger complexity and the shallower pipeline.

We modeled the routers of the two NoCs in Orion 2.0, an accurate router power and area modeling tool which already proved capable of capturing the power characteristics of the first of the two examples at a fixed nominal voltage [95]. Orion accepts router architecture and technology parameters together with power supply voltage and clock frequency inputs. However, it is not able to choose the minimum

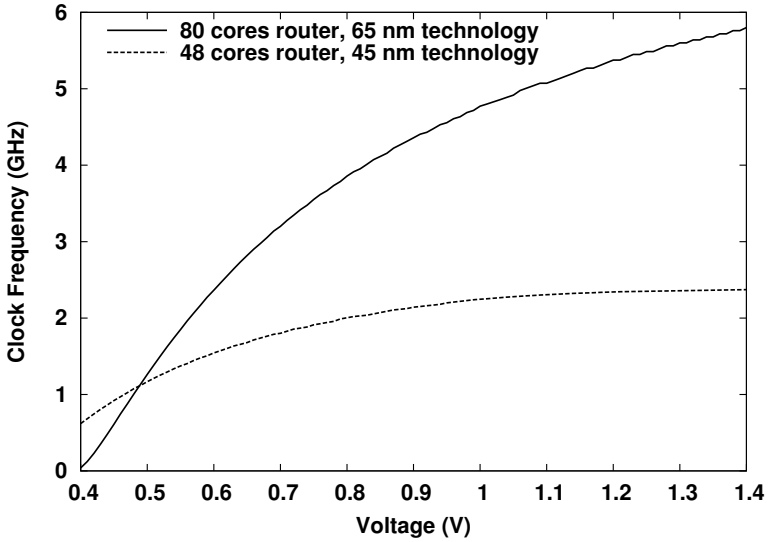


Figure 7.7. Minimum power supply voltage as a function of frequency for Intel 80 core and 48 core router designs.

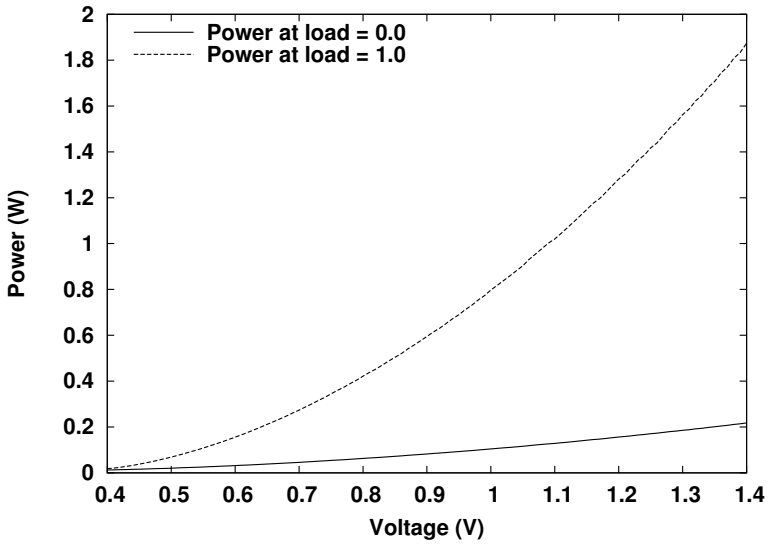


Figure 7.8. Power at full and at no load for the Intel's 80 core router as a function of clock frequency. Supply voltage was set at the minimum value that guarantees correct operation at chosen frequency.

voltage for a given clock frequency (or equivalently the maximum clock frequency for a given voltage), a step necessary for our DVFS-based power evaluation. On top of Orion 2.0, we thus added an accurate voltage versus frequency model we derived from the Mastar Mosfet model, a tool used to compile the ITRS roadmap of CMOS technology [97]. The maximum clock frequency is calculated based on a gate delay model – the delay of an inverter loaded with a fanout-of-four (FO4) load, also known as FO4 delay – which in turn is a function of supply voltage as well as of many technology parameters like transistor doping levels, channel length, oxide thickness, and many others. We obtained with Mastar the typical FO4 delay for the two technologies in which the two routers were implemented, 65 and 45 nm, as a function of the supply voltage. We then made the reasonable assumption that the whole minimum clock period scales with voltage the same way a simple gate does. Therefore, from Intel's published experimental data we took the clock period at a given voltage, and from that we drew a suited proportionality factor between clock period and gate delay. Such constant factor is representative of the length of the switch's critical path in terms of FO4 delays. Using that constant at any voltage (the number of FO4 delays depends on the number of gates in a critical path only, and not on the voltage), we could plot curves that relate voltage and frequency, as shown in Figure 7.7.

To fully characterize a router's power model, we ran Orion at various voltage

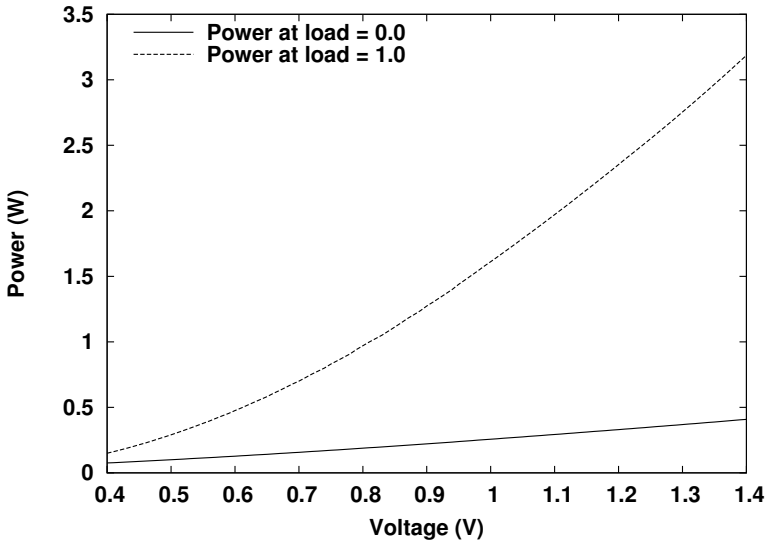


Figure 7.9. Power at full and at no load for the Intel's 48 core router as a function of clock frequency. Supply voltage was set at the minimum value that guarantees correct operation at chosen frequency.

and frequency pairs and varying another important input parameter, the *load*. This value represents the number of flits each router port receives per clock cycle. A 1.0 value corresponds to a fully loaded router (one flit per port received per clock cycle), whereas a load of 0.0 represents a situation of idle router. It was shown that a router power as a function of the load at its input ports is accurately captured by a linear model [98]. Power expressions in Orion are linear functions of the load parameter. Therefore, to build a model suitable for integration in our algorithms it was sufficient to evaluate power at full load (1.0) and at no load (0.0). Any intermediate value can be obtained as a weighted combination of the two values as follows

$$P(l,f) = P_0(f) + l \cdot \big(P_1(f) - P_0(f)\big). \tag{7.14}$$

Figures 7.8 and 7.9 report power values as a function of clock frequency and consequently also of voltage. Voltage and frequency pairs were chosen according to the curves plotted in Figure 7.7, for the two routers. The two curves in each plot in Figures 7.8 and 7.9 represent power at maximum and at zero load. Values plotted in figures are stored in tables which get looked-up by our algorithms when required for power analysis. The only entry in table is clock frequency, whereas the load parameter, which we derive from the traffic matrix, is inserted in (7.14) for final evaluation.

We included the Orion power model in our two-plane NoC architecture and ran the allocation algorithms. We only show the results for the 48 cores Intel chip in

Fig. 7.10, for the case of hotspot traffic pattern. We did not run the numerical solver for IPC to get the lower bound power since it requires a closed-form expression for the power, which we can not extract from Orion's model. As we could expect, differently from previous results which ignore static power, two planes policies (2P curves) consume more than a single plane when the load is low (XY DVFS curve), because of the doubling of resources. But when the input load is greater than 0.65, the same trend of previous results is observed: load concentration is better than load balancing, which validates our previous findings. We believe that a suitable application of standby or sleep mode strategies, which we do not consider in this work, could help lessen the impact of leakage power.

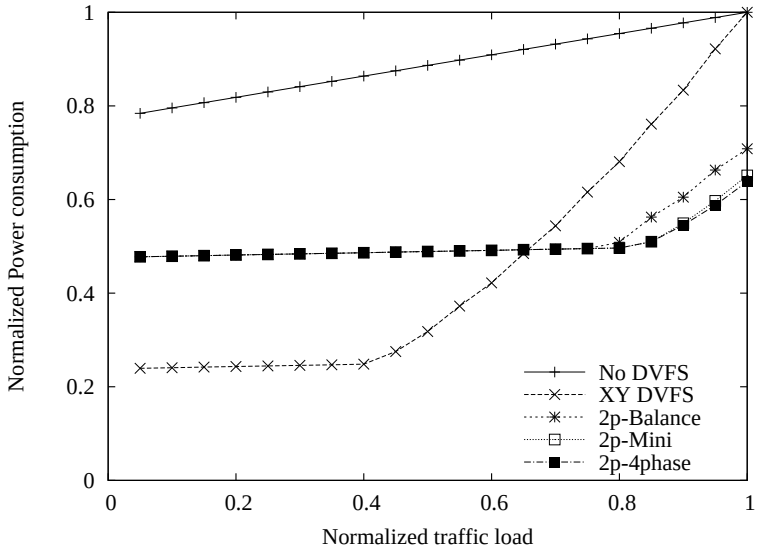


Figure 7.10. Power consumption of the Intel 48 cores NoC under hot-spot traffic pattern and different loads.

7.6 Conclusions

In this chapter we consider a two-planes NoC architecture, in which each plane exploits Dynamic Voltage and Frequency Scaling (DVFS) independently from the other plane. We show how to leverage the spatial diversity provided by the two planes to reduce more power than the one achieved by a naive load-balancing scheme. The main idea is to concentrate the high-traffic flows in one plane and the low-traffic flows in the other plane; in this way, at least one plane can run at a reduced voltage and frequency to better exploit the beneficial effects of DVFS. We propose three traffic allocation algorithms, with a different performance and complexity tradeoff,

and investigate the corresponding power consumptions under different traffic matrices. We compare their performance with respect to single plane architectures and to the optimal allocation.

Our results have been obtained assuming a well known theoretical model for the dynamic power in CMOS hardware, but they have also been validated through a realistic power model, obtained from an accurate hardware simulator for NoCs.

Chapter 8

Energy Efficient Distributed Software Router Design

As we have shown in the first part of the thesis, a multistage software router (MSR) architecture is composed of several personal computers (PCs) to overcome scalability issues of a single stage PC-based software router (SR). Although the architecture scales almost linearly with the number of internal elements, energy consumption could be a threat to scalability features when building a carrier grade router with many internal components.

In this chapter, we assume a known 24 hour traffic load, we propose three new energy efficient MSR design approaches, which enhance the performance of energy saving algorithms that minimize the energy consumption by tailoring the MSR architecture, designed for the worst case traffic scenario, to match the current input traffic demand. We will show in the simulation part that the proposed design approaches reduce the MSR energy consumption by roughly 10% with respect to existing energy saving algorithms for similar costs and up to 20% depending on the initial admissible budget.

8.1 Introduction

Proprietary networking equipments, and routers in particular, have high cost in terms of both CAPEX (investment cost) and OPEX (training and energy consumption). Software Routers (SRs) which are based on Personal Computers (PCs) running open-source network applications like Linux, Click Modular Router or XORP [99, 100], are an appealing alternative to proprietary devices thanks to their low cost, programmability and flexibility.

To scale SR to large size and high performance, distributed architectures composed by several PCs should be sought for. As an example, a multistage software

router (MSR) architecture shown in Fig. 8.1 has been proposed in [101]. The multi-stage architecture exploits classical PCs as elementary switching elements to build a high-performance SR. The proposed architecture has three stages: the layer-2 front-end Load Balancers (LBs) acting as the interfaces to the external networks, the back-end PCs (BEPCs), also named back-end routers, providing layer-3 routing functionality, and an interconnection network based on Ethernet switches to interconnect the two stages.

The key advantages of the MSR architecture are the ability to i) overcome the performance limitation of a single PC-based router by offering multiple, parallel forwarding paths, ii) scale the number of interfaces, iii) improve router performance by incrementally adding/upgrading internal elements.

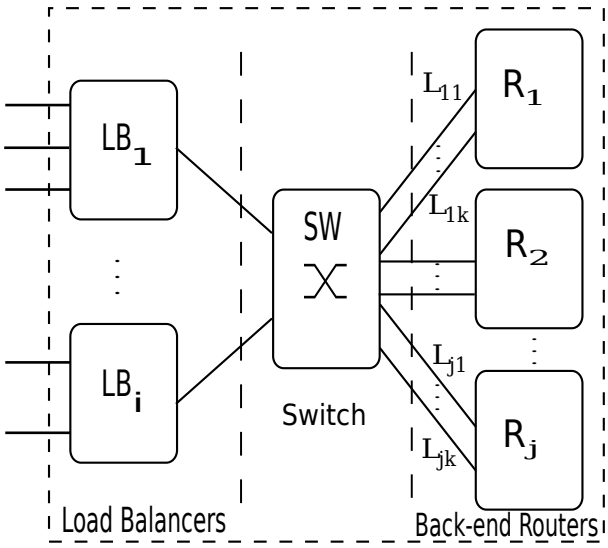


Figure 8.1. MSR Architecture: the load balancers (first stage), the switch (second stage) and the back-end routers (third stage)

LBs distribute IP packets to BEPCs, which share the same routing table providing several parallel forwarding paths in the architecture. Increasing the number of BEPCs enhances MSR routing performance. However, this performance scaling feature implies a high level of redundancy at this stage for period of low traffic load, which translates into high and un-needed energy consumption. To enhance energy efficiency, unused BEPCs must be switched off during low load periods, because the routing task can be handled by a subset of BEPCs. Note that while BEPCs are redundant during low traffic periods, LBs and switches are not, acting respectively as external interfaces, which must stay active to guarantee external connectivity and may be switched off only with a network-wide view, and internal interconnection network which must guarantee internal connectivity. Thus we focus only on optimizing the number of active BEPCs depending on the MSR traffic load.

We [102, 103] proposed on-line and off-line energy saving schemes in MSR architecture to adapt the capacity of the back-end stage to a given known traffic demand. More precisely, in the off-line case the MSR architecture is optimized at a given time instant for a known traffic demand. If running this algorithm at different times for a variable traffic profile, the MSR architecture configurations obtained may be composed by a different set of PCs. In the on-line case a differential approach is sought for: a new configuration is obtained by updating the configuration defined at the previous time by minimizing the number of PCs that should be switched on/off. No attempt is made to globally optimize the MSR architecture taking into account the long-term (e.g., 24 hours) traffic profile.

In [102, 103] it was shown that the proposed algorithms can save up to 60% of energy when compared to architectures sized for peak traffic loads. However, the achievable savings depend on the back-end routers configuration. For instance, if the back-end stage is built up of PCs with coarse capacity granularity, it is difficult to resize the configuration in period of low traffic load, and the installed un-needed capacity translates into energy wastage. On the other hand, a back-end stage composed of PCs with smaller capacity granularity is more flexible for reconfiguration. Indeed, the smaller capacity means a larger number of PCs to handle a given traffic demand, which requires more energy.

The goal of this chapter is to define the back-end routers configuration that minimizes the energy consumption over a given period, under the assumption that once the MSR configuration has been optimally defined, energy saving algorithms similar to those presented in [102, 103] are used to adapt the back-end stage configuration to the input traffic demand.

8.2 Energy Efficient Back-end Routers Design

Capacity to handle traffic is the basic requirement when designing a high performance MSR to satisfy the peak load demand. However, the approach of sizing back-end routers on the peak demand does not translate in energy efficient configuration. Given the increasing importance of energy saving techniques in networks, back-end routers design should consider energy consumption in addition to peak load capacity requirement. To achieve this goal, we propose three different back-end routers design approaches: a goal programming based methodology, a heuristic and a locally optimal approach, described in detail in the following subsections. All design approaches assumes the following input parameters:

- input traffic $T_t \in \mathbb{R}$: an average traffic profile, derived by estimates or measures, and sampled every time t;
- set of available PCs to be used as back-end routers. Let S be a set of groups

of PCs of different types. Each PC in the same group is characterized by the same power consumption $P_k \in \mathbb{R}$, routing capacity $R_k \in \mathbb{R}$,

and hardware cost $C_k \in \mathbb{R}$. In the design phase, in each group $k \in S$ PCs are assumed to be available in infinite number.

8.2.1 Goal Programming Design Approach

The first approach models the energy efficient MSR design as a preemptive goal programming problem. The model has two objectives: energy minimization and cost minimization. The primary objective is to minimize the energy consumption of the back-end routers over the traffic sampling duration. This defines the N_k PCs from each group k used to design the MSR architecture. To keep cost under control, a maximum admissible budget I is assumed. The problem is formulated as follows:

minimize

$$O(t,k) = \sum_t \sum_k P_k N_k \alpha_t \tag{8.1}$$

subject to

$$\sum_k R_k N_k \alpha_t \geq T_t \quad \forall t \tag{8.2}$$

$$\sum_k C_k N_k \leq I \quad \forall k \in S \tag{8.3}$$

$$0 \leq N_k \alpha_t \leq N_k \quad \forall k \in S, \forall t, \tag{8.4}$$

$$N_k, N_k \alpha_t \in \mathbb{Z}, \alpha_t \in [0,1]$$

The solution to the optimization problem (8.1) – (8.4) is the number of PCs N_k from each group k to be used to build the back-end routers configuration. (8.2) adapts N_k by $\alpha_t \in [0,1]$ defining the suitable N_k composition that satisfies T_t at each sampling instance t. It takes into account the energy saving algorithms running after the initial design phase to adapt the designed configuration to the input traffic demand T_t. (8.3) ensures that the hardware cost of the selected PCs should not exceed the maximum cost I and (8.4) bounds the number of PCs needed at each sampling time t within N_k.

The objective function, $O(t,k)$, minimizes the sum of each sampling instance active configuration power dissipation. This is equivalent to minimize the energy consumption over a specified period. Note that the active configuration at time instance t consists of only $N_k \alpha_t$ from each group k. Hence, the energy consumption of a MSR configuration varies for each sampling time t.

If the maximum admissible budget I, which is difficult to set up a priori, is too large with respect to traffic needs, the first design phase could result in a costly configuration because the optimization target is the energy consumption. Thus, we define a second step that optimize the cost of the back-end routers by tailoring any possibly over-sized configuration obtained in the first step. To maintain the primary

objective, we enforce the energy cost obtained from the energy optimization problem as an equality constraint and build the budget optimization model as follows:

minimize

$$O(k) = \sum_k C_k N_k \tag{8.5}$$

subject to

$$\sum_k R_k N_k \alpha_t \geq T_t \quad \forall t \tag{8.6}$$

$$0 \leq N_k \alpha_t \leq N_k \quad \forall t, \forall k \in S \tag{8.7}$$

$$\sum_t \sum_k P_k N_k \alpha_t = O(t,k) \tag{8.8}$$

Constraints (8.6) and (8.7) have the same meaning as in (8.2) and (8.4) respectively. (8.8) guarantees that the new back-end routers configuration dissipates the same power as in the previous phase.

The solution is a back-end PCs configuration with optimized cost that satisfies the traffic demand and maintains the primary objective of back-end routers energy consumption. Thus equations (8.1) – (8.8) define a preemptive goal programming model that minimizes the aggregate energy consumption of back-end routers over a specified period with minimum cost.

8.2.2 Heuristic Design Approach

The heuristic approach defines the back-end routers configuration by greedily choosing the most efficient PCs until the traffic requirement is satisfied. The PCs efficiency is defined as performance per unit watt. Since we assumed infinite PCs for each group, only the most efficient group k is used to build the back-end routers cluster. Thus, the number of PCs, N_k, required to handle a peak load $T_{t,max}$ is simply computed as:

$$N_k = \left\lceil \frac{T_{t,max}}{R_k} \right\rceil \tag{8.9}$$

Similar cluster design approaches exist in the literature. The Google cluster architecture [104] considers performance per unit of price as PCs selection criterion to setup a cluster configuration. This approach gives priority to commodity-class PCs to high-end multi-processor servers because of their cost advantage. The most common way of resizing a cluster capacity is to use servers with best absolute performance such that the cluster handles peak load with fewer reliable computers [105, 106]. However, such cluster is far more expensive due to the higher interconnect bandwidth and reliability of the servers [104]. The clusters designed by these approaches have not been analyzed for power consumption. In the design validation section we will compare their energy costs with those of our proposed design techniques.

8.2.3 Locally Optimal Design Approach

This optimal design approach defines the optimal back-end routers configuration considering only the peak load. This configuration is used as a basis for the other sampling time t, assuming that energy saving algorithms deployed after the design stage adjust this base configuration to the traffic demand T_t at each sampling instance. Of course this design is not globally optimal. The problem formulation is the same as (8.1) – (8.8) with a single sampling time, i.e. the peak load sampling time.

8.3 Design Validation

In this section we discuss the performance of the proposed approaches through experimental results. First we discuss inputs to the design problem: The traffic traces and the experimental scenario. Then, we discuss the main results in the following subsections.

8.3.1 Traffic Traces

Instead of defining a synthetic traffic load with a typical day and nigh behaviour, we captured traffic from a university core router in Twente to derive the traffic load. To build large MSR, traffic was scaled up while keeping the ratio among traffic loads at different sampling time. We used Perl scripts to aggregate the traces into 5min, 15min, 30min and 60min time interval. Then, we averaged the traffic volume over a week to get a per day volume statistics. Fig. 8.2 shows the 5min and 60min volume traces. We did not include the 15 and 30min curves for the sake of clarity, but they show a similar behavior.

8.3.2 Experimental Setup

The following four groups of PCs are used as input to the model [107, 108, 109]:
Group I

- Routing capacity, $R_I = 6500$ Mbps;
- PC power consumption, $P_I = 75$ W;
- PC cost, $C_I = \$250$

Group II

- Router routing capacity, $R_{II} = 8700$ Mbps;

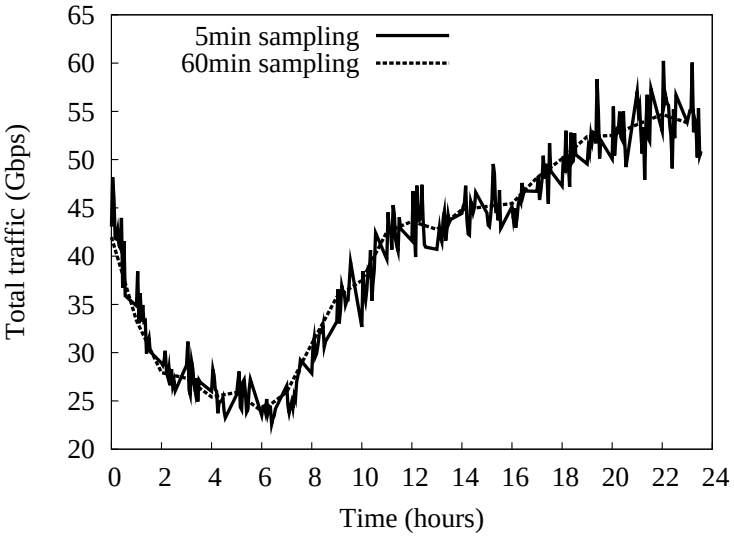


Figure 8.2. Input traffic trace used in the experiment

- PC power consumption, $P_{II} = 95$ W;
- PC cost. $C_{II} = \$400$

Group III

- Router routing capacity, $R_{III} = 10000$ Mbps;
- PC power consumption, $P_{III} = 120$ W;
- PC cost, $C_{III} = \$500$

Group IV

- Router routing capacity, $R_{IV} = 8500$ Mbps;
- PC power consumption, $P_{IV} = 92$ W;
- PC cost, $C_{IV} = \$400$

An infinite number of PCs in each group is available in the design phase. Based on this setup, we compare the different design approaches in terms of energy consumption and cost of back-end routers configuration.

- *Design-I*: build a back-end routers cluster by choosing PCs with highest performance/price ratio

- *Design-II*: build a back-end routers cluster by choosing PCs with the highest performance (i.e, routing capacity)
- *Design-III*: build a back-end routers cluster by choosing PCs with the highest performance/power ratio
- *Design-IV*: design based on optimal back-end routers configuration for the peak load
- *Design-V,I_x*: solves the optimization problem defined in (8.1) – (8.8) for different budget constraints I_x where x = 1,2,...n and $\mathrm{I}_1 < \mathrm{I}_2 < ... < I_n$.

Design-I and *Design-II* are existing cluster design approaches [104, 105, 106]. The former gives priority to commodity-class PCs in terms of cost; on the contrary, high performance servers are preferred in the latter approach to ensure hardware reliability. The last three are our proposed design approaches (See section 8.2). We derive the energy consumption over a period of 24 hours and compare the results with the optimal solution which is obtained by solving the following ILP model:

minimize

$$O_{optimal} = \sum_t \sum_k P_k N_k \tag{8.10}$$

subject to

$$\sum_k R_k N_k \geq T_t \tag{8.11}$$

Note that we do not have any constraint in this model, except to guarantee the capacity for the traffic load.

We also assume that algorithms similar to those presented in [102] will be used to turn on/off PCs at each traffic sampling time to adapt the back-end routers size to the traffic demand. We rely on CPLEX [110] to collect the results of the optimization models.

8.3.3 Results

In this subsection we compare the back-end routers selected by the different design approaches from the energy consumption and cost perspective. We also discuss the impact of traffic sampling interval on each design approaches.

Energy consumption

Fig. 8.3 compares the power dissipation of the different configurations. The result is based on the 60 minutes input traffic sampling. The energy consumption

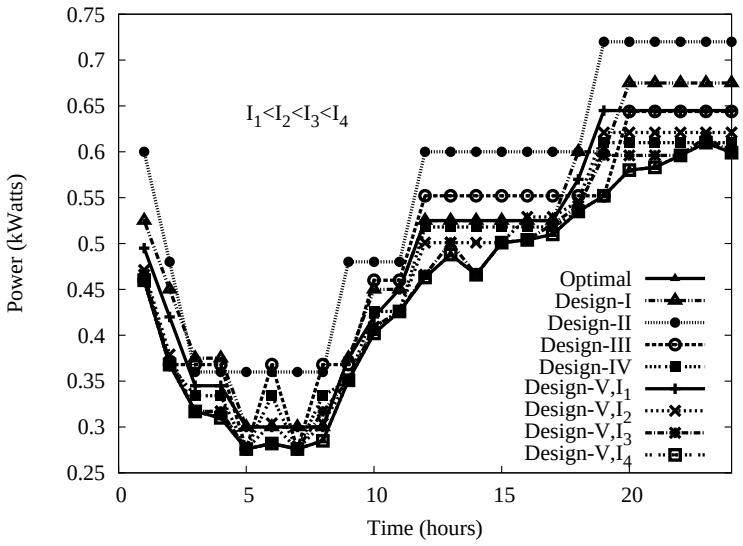


Figure 8.3. MSR power dissipation of different design approaches based on 60min sampling

of the back-end routers over a given period can be computed from the power curves as:

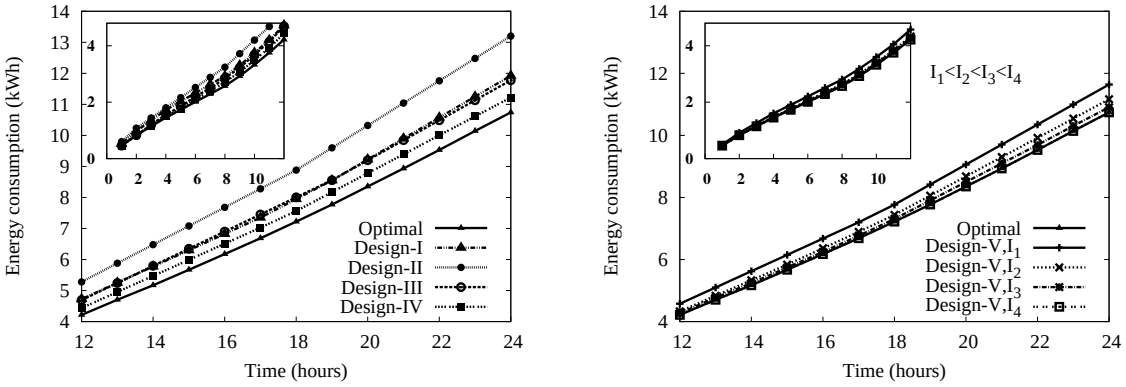
$$Energy = power \times time \tag{8.12}$$

This is equivalent to compute the area under each power curve. The result is shown for a 24 hours traffic pattern in Fig. 8.4. For the sake of clarity we split the results into two graphs (Fig. 8.4(a),Fig. 8.4(b)). The optimal solution obtained by solving the ILP model (8.10) – (8.11) is included in both graphs as a reference.

Results show that *Design-II* consumes the highest energy in the specified period. This design approach gives priority to high performance devices to setup the back-end routers cluster. Since the high-end servers have large capacity granularity, resizing the configuration to the input traffic mostly ends up in wasting some capacity. Thus, less loaded PCs in the back-end are sources of energy inefficiency and the energy required by this approach is roughly 2kWh larger than the one required by the optimal algorithm.

The design principle that configures back-end routers based on performance/price ratio, *Design-I*, has better tailoring properties as it give priority to mid-range PCs that have relatively smaller capacity and consume less energy, making it easier to match input traffic. This yields roughly 1kWh less energy consumption than *Design-II*, although it is fairly far from the optimal.

The curves labeled *Design-III*, *Design-IV*, and *Design-V,*I_x are the energy consumption of our proposed back-end routers cluster design approaches. Goal programming technique outperforms the other design approaches if not under very tight



(a) Energy consumption: Design-I, Design-II, Design-III, and Design-IV

(b) Energy consumption: Design-V

Figure 8.4. Energy consumption of back-end routers selected by different design approaches (based on 60 min traffic sampling)

budget constraint (see label *Design-V,*I_1) where the energy consumption is similar to *Design-III* and *Design-I*. However, with less strict budget requirements, performance is very close to the optimal solution (see label *Design-V,*I_3 and *Design-V,*I_4), saving up to 1-2 kWh every day if compared to other design approaches. The goal programming approach is more efficient because it selects PCs from heterogeneous groups giving more flexibility to resize the back-end PCs to match input traffic.

Design-IV is also competitive with respect to the goal programming approach, being roughly 0.25kWh less efficient than the optimal solution over 24hours. *Design-III*, on the other hand, performs worse than the two other proposed approaches.

The proposed design approaches, namely *Design-III*, *Design-IV* and *Design-V,*I_x, save roughly 10% of energy when compared to existing design techniques.

Cost

Since cost is always a key consideration, it is important to compare the cost of back-end routers defined by different design approaches, as shown in Fig. 8.5. As expected, *Design-I* has the least price because it is based on the performance/price ratio selection criterion which tries to minimize price while increasing performance. *Design-IV,*I_1 has similar costs mainly because of the tight initial investment constraint. It is also interesting to note that the two approaches have similar energy efficiency, as shown in Fig. 8.4. However, the cost of *Design-IV,*I_x increases for increasing budget constraints while the energy consumption decreases. The almost optimal solution, *Design-IV,*I_4, has cost $\sim$1.5-2 times larger than any of the other approaches. *Design-IV,*I_3 has similar costs with the other design approaches but

it achieves near optimal energy efficiency. There is a trade-off between energy efficiency and cost in goal programming back-end routers design approach which can be controlled by selecting a proper budget constraint.

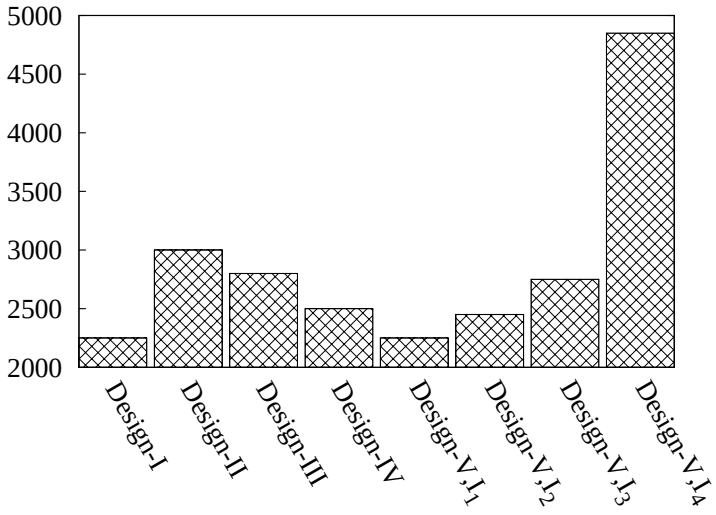


Figure 8.5. Cluster cost for different design approaches

Except the best goal programming approach, all other design approaches have similar cost. At least with the considered set of PCs, which are those available on the market today, energy efficiency seems to be the most important metric in selecting the proper set of back-end routers.

Sampling interval impact

We also analyzed the impact of traffic sampling intervals on energy efficiency. We analyzed the energy consumption of the back-end PCs using traffic traces sampled every 5min, 15min, 30min and 60min. The results, not reported due to lack of space, show that energy consumption depends only slightly on traffic sampling interval. However, if the back-end routers cluster is analyzed under a sampling interval for which it was not designed, the results are different as shown in Fig. 8.6. We considered a cluster designed on 60min sampling, later analyzed for its energy consumption under the same traffic trace but for a 5min sampling interval. The curve labeled "Design-X/Sampling-Y" represents a back-end PCs designed under "X min" traffic sampling but analyzed for energy consumption under "Y min" sampling. The figure shows the difference in energy consumption for a MSR designed for 60min and analyzed for 60min and 5min traffic sampling time. The mismatch

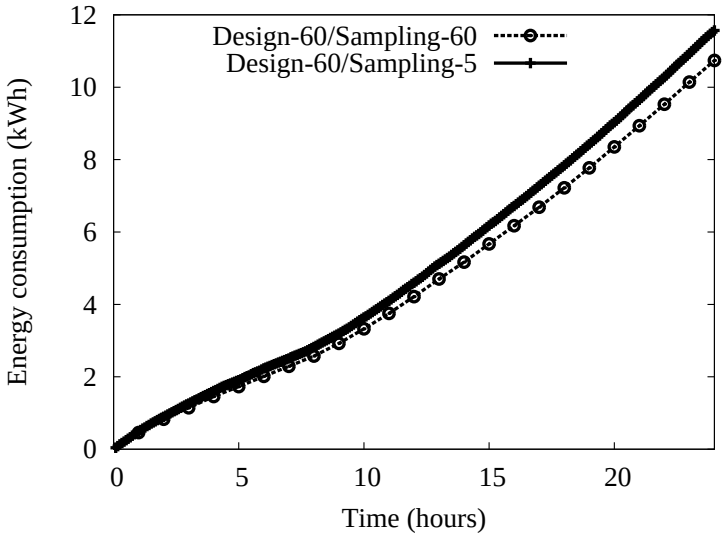


Figure 8.6. Design and sampling mismatch effect

accounts for 0.8 kWh energy wastage per day. This implies that algorithms responsible to resize the back-end routers configuration after initial deployment should stick to the sampling time used during the design phase to reduce energy inefficiency.

8.4 Conclusions

In this chapter we proposed three different energy efficient back-end routers design approaches that make energy saving algorithms more efficient. They permit to save up to 10% of energy with respect to existing design approaches with similar budget. Savings could raise to 20% for the goal programming design approach if relaxing the budget constraint.

The traffic sampling interval is not so fundamental in designing energy efficient back-end routers. Furthermore, if back-end PCs are designed for a given sampling time, algorithms responsible to resize the configuration for variable traffic load should stick to the sampling time used during the design stage to improve efficiency.

Although we considered the specific design scenario of a MSR architecture, the proposed design approaches can be used in any cluster design that involve PCs as the basic computational resources.

As future research activity, we would like to include quality of service issues as an additional design requirement.

Chapter 9

Conclusion

In this thesis work, we focused on two main problems in switching, the virtualization and energy saving. In the first part we showed that virtualization could be valuable to build flexible software routers. We implement the Multistage Software Router (MSR) into different virtual infrastructures including VMware ESXi, KVM, XEN and we evaluated their performance for the single component/stage and the whole architecture. The results showed that virtualization for software router could impose overhead and reduce the performance for around 40 % compared with the physical software router. As such, we worked on two directions for increasing the performance, namely from the architecture point of view and from the single component tuning point of view. In the first aspect, we experimentally evaluated the allocation of different virtual units (i.e, load balancer, back-end router) into physical hardware and we discovered that the performance is quit depending on the allocation, especially when precious hardware resource are contended by more units which are i/o intensive. Thus we argue that it is important to separate the VMs with the same functionality and group the ones with different functionalities, i.e, cpu intensive and i/o intensive could be together. Next, we measured the performance of mapping different virtual CPUs into physical cores without considering the location of VMs. The results showed that the cpu cycles are shared among the number of virtual CPUs bounding to it and the performance is influenced by the workload of each virtual CPU. Regarding the virtual machine tuning work, we carefully created the software router with minimum possible thread and bound each thread to a specific CPU core and we gain dramatically performance. We have also demonstrated that the thread priority, the internetworking methodology and the number of hardware CPU cores are critical to the performance. We presented some initial results on how to optimally configure all these parameters to obtain the throughput close to hardware software router, yet some more systematical measurement is still missing for obtaining the general rules on how to exploit all the resource efficiently for virtual software router.

In the second part of the thesis, we considered the energy saving problem for two main different switching architecture, namely network on chip and multistage software router. In network on chip, we noticed that the chip capacity is usually built to match with the peak traffic load and this could waste energy when the traffic load is low. Thus we proposed to combine dynamic voltage and frequency scaling with load balancing algorithm to reduce the chip power when possible. In specifically, we considered a common mesh chip architecture with realistic constraints (i.e, all links work at the same frequency, to minimize the voltage/frequency transition synchronization overhead) and we evaluated the chip power cost with different traffic load when running our load balancing algorithms. Our results have been compared with the optimal value obtained by modeling and solving the power saving problem formally with optimization technique. The results showed that in the single plane mesh NoC architecture, it is usually enough to consider two path load balancing for power saving and the results could approach the optimal value very close. Furthermore, we noticed that the DVFS effectiveness is restricted by the bottleneck link, to solve this issue, we proposed to build the NoC architecture with multiplan and we assume each plane can work on its own frequency and voltage for all links. We found that the DVFS could be more effective if we concentrate more traffic on few planes which run fast enough to accommodate the high traffic load and leaving all other planes working at the minimum possible voltage/frequency with few flows. Our results have been validated not only by the empirical cubic power model but also by a realistic power model extracted from Orion 2.0 NoC power simulator. Regarding the multistage software router, we proposed a novel design method called the goal programming approach to find the best compositions of the MSR architecture when providing the traffic trace, budget and server type. Our main results show that with the similar investment, our approach could save 10 % $\sim$ 20 % more energy compare with the energy-blind MSR architecture, pushing our MSR solution a step more close to the production network.

Bibliography

[1] P. Bogdan and R. Marculescu, "Workload characterization and its impact on multicore platform design," in *Hardware/Software Codesign and System Synthesis (CODES+ISSS), 2010 IEEE/ACM/IFIP International Conference on*, oct. 2010, pp. 231–240.

[2] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek, "The Click modular router," *ACM Trans. Comput. Syst.*, vol. 18, no. 3, pp. 263–297, 2000.

[3] M. Handley, O. Hodson, and E. Kohler, "XORP: an open platform for network research," *SIGCOMM Comput. Commun. Rev.*, vol. 33, no. 1, pp. 53–57, 2003.

[4] "Quagga routing suite," http://www.quagga.net.

[5] S. Han, K. Jang, K. Park, and S. Moon, "Building a single-box 100 gbps software router," in *Local and Metropolitan Area Networks (LANMAN), 2010 17th IEEE Workshop on*, may 2010, pp. 1–4.

[6] ——, "Packetshader: a gpu-accelerated software router," *SIGCOMM Comput. Commun. Rev.*, vol. 40, no. 4, pp. 195–206, Aug. 2010.

[7] R. Bolla and R. Bruschi, "RFC 2544 performance evaluation and internal measurements for a Linux based open router," in *HPSR*, Poznan, Poland, Jun. 2006.

[8] ——, "PC-based software routers: high performance and application service support," in *PRESTO*, Seattle, WA, USA, Aug. 2008.

[9] ——, "An effective forwarding architecture for SMP Linux routers," in *IT-NEWS*, Venice, Italy, Feb. 2008.

[10] K. Argyraki, S. Baset, B.-G. Chun, K. Fall, G. Iannaccone, A. Knies, E. Kohler, M. Manesh, S. Nedevschi, and S. Ratnasamy, "Can software routers scale?" in *PRESTO*, Seattle, WA, USA, Aug. 2008.

[11] IETF, "Forwarding and control element separation working group (ForCES)," http://tools.ietf.org/wg/forces/.

[12] W. Wang, L. Dong, B. Zhuge, M. Gao, F. Jia, R. Jin, J. Yu, and X. Wu, "Design and implementation of an open programmable router compliant to IETF ForCES specifications," in *ICN*, Sainte-Luce, Martinique, Apr. 2007.

[13] M. Dobrescu, N. Egi, K. Argyraki, B.-G. Chun, K. Fall, G. Iannaccone,

A. Knies, M. Manesh, and S. Ratnasamy, "RouteBricks: exploiting parallelism to scale software routers," in *SOSP*, Big Sky, MT, USA, Oct. 2009.

[14] A. Bianco, J. M. Finochietto, M. Mellia, F. Neri, and G. Galante, "Multistage switching architectures for software routers," *IEEE Network*, vol. 21, no. 4, pp. 15–21, Jul.-Aug. 2007.

[15] A. Bianco, R. Birke, J. M. Finochietto, L. Giraudo, F. Marenco, M. Mellia, A. Khan, and D. Manjunath, "Control and management plane in a multi-stage software router architecture," in *HPSR*, Shanghai, China, May 2008.

[16] A. Khan, R. Birke, D. Manjunath, A. Sahoo, and A. Bianco, "Distributed PC based routers: bottleneck analysis and architecture proposal," in *HPSR*, Shanghai, China, May 2008.

[17] A. Bianco, J. M. Finochietto, G. Galante, M. Mellia, D. Mazzucchi, and F. Neri, "Scalable layer-2/layer-3 multistage switching architectures for software routers," in *IEEE GLOBECOM*, San Francisco, CA, USA, Dec. 2006.

[18] A. Bianco, F. G. Debele, and L. Giraudo, "Energy saving in distributed router architectures," in *IEEE ICC*, Ottawa, Canada, June 2012.

[19] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: enabling innovation in campus networks," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 69–74, 2008.

[20] P. Szegedi, J. Riera, J. Garcia-Espin, M. Hidell, P. Sjodin, P. Soderman, M. Ruffini, D. O'Mahony, A. Bianco, L. Giraudo, M. Ponce de Leon, G. Power, C. Cervello-Pastor, V. Lopez, and S. Naegele-Jackson, "Enabling future internet research: the FEDERICA case," *IEEE Communic. Mag.*, vol. 49, no. 7, pp. 54–61, Jul. 2011.

[21] M. B. Anwer and N. Feamster, "Building a fast, virtualized data plane with programmable hardware," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 40, no. 1, pp. 75–82, 2010.

[22] M. Caesar and J. Rexford, "Building bug-tolerant routers with virtualization," in *PRESTO*, Aug. 2008.

[23] N. Egi, A. Greenhalgh, M. Handley, M. Hoerdt, L. Mathy, and T. Schooley, "Evaluating Xen for router virtualization," in *ICCCN*, Honolulu, HI, USA, Aug. 2007.

[24] "VMware," http://www.vmware.com.

[25] "Windows server hyper-v," http://www.microsoft.com/hyper-v-server/en/us/default.aspx.

[26] "IBM system z," http://www-03.ibm.com/systems/z/.

[27] K. P. Lawton, "Bochs: A portable PC emulator for Unix/X," Linux Jounal, Tech. Rep., Sep. 1996.

[28] F. Bellard, "QEMU, a Fast and Portable Dynamic Translator," in *USENIX*, Anaheim, CA, USA, Apr. 2005.

[29] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebar, I. Pratt, and A. Warfield, "Xen and the art of virtualization," in *Proceedings of the nineteenth ACM symposium on Operating systems principles (SOSP03)*, 2003.

[30] A. Kivity, Y. Kamay, D. Laor, U. Lublin, and A. Liguori, "KVM: the linux virtual machine monitor," in *Proceedings of the Linux Symposium*, Ottawa, Ontario, Canada, 2007.

[31] "OpenVZ project," http://www.openvz.org/.

[32] "Linux Vserver," http://wiki.linux-vserver.org/.

[33] N. M. M. K. Chowdhury and R. Boutaba, "Network virtualization: state of the art and research challenges," *IEEE Communic. Mag.*, vol. 47, no. 7, pp. 20–26, Jul. 2009.

[34] T. Anderson, L. Peterson, S. Shenker, and J. Turner, "Overcoming the internet impasse through virtualization," *Computer*, vol. 38, no. 4, pp. 34–41, Apr. 2005.

[35] "NSF GENI project," http://www.geni.net/.

[36] "The FEDERICA project," http://www.fp7-federica.eu/.

[37] F. Anhalt and P. Primet, "Analysis and experimental evaluation of data plane virtualization with Xen," in *ICNS*, Valencia, Spain, Apr. 2009.

[38] K. K. Ram, J. R. Santos, Y. Turner, A. L. Cox, and S. Rixner, "Achieving 10 gb/s using safe and transparent network interface virtualization," in *Proceedings of the 2009 ACM SIGPLAN/SIGOPS international conference on Virtual execution environments*, ser. VEE '09. New York, NY, USA: ACM, 2009, pp. 61–70. [Online]. Available: http://doi.acm.org/10.1145/1508293.1508303

[39] N. Egi, A. Greenhalgh, M. Handley, M. Hoerdt, F. Huici, and L. Mathy, "Towards high performance virtual routers on commodity hardware," in *Proceedings of the 2008 ACM CoNEXT Conference*, ser. CoNEXT '08. New York, NY, USA: ACM, 2008, pp. 20:1–20:12. [Online]. Available: http://doi.acm.org/10.1145/1544012.1544032

[40] S. Rathore, M. Hidell, and P. Sjodin, "Performance evaluation of open virtual routers," in *IEEE GLOBECOM*, Miami, FL, USA, Dec. 2010.

[41] A. Bavier, N. Feamster, M. Huang, L. Peterson, and J. Rexford, "In VINI veritas: realistic and controlled network experimentation," *SIGCOMM Comp. Communic. Rev.*, vol. 36, no. 4, pp. 3–14, 2006.

[42] Microsoft, "Enabling a dynamic datacenter with microsoft virtualization," Microsoft, Tech. Rep., 2008-2009.

[43] M. Dobrescu, N. Egi, K. Argyraki, B.-G. Chun, K. Fall, G. Iannaccone, A. Knies, M. Manesh, and S. Ratnasamy, "Routebricks: exploiting parallelism to scale software routers," in *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*, ser. SOSP '09. New York,

NY, USA: ACM, 2009, pp. 15–28. [Online]. Available: http://doi.acm.org/10.1145/1629575.1629578

[44] S. Soltesz, H. Poltz, M. Fiuczynski, A. Bavier, and L. Patersson, "Container-based operating system virtualization: A scalable, high-performance alternative to hypervisors," in *EuroSys*, Mar. 2007.

[45] "Virtual linux from ibm," http://www.ibm.com/developerworks/linux/library/l-linuxvirt/.

[46] "XEN networking," http://wiki.xensource.com/xenwiki/XenNetworking.

[47] "Agilent N2X router tester," http://advanced.comms.agilent.com/n2x/.

[48] VMware, "Understanding full virtualization, paravirtualization, and hardware assist," VMware Inc., Tech. Rep., 2007.

[49] "The Click modular router web-site." [Online]. Available: http://www.read.cs.ucla.edu/click/click

[50] A. Bianco, R. Birke, D. Bolognesi, J. M. Finochietto, G. Galante, and M. Mellia, "Click vs. Linux: Two efficient open-source IP network stacks for software routers," in *HPSR*, Hong Kong, China, May 2005.

[51] "Iperf." [Online]. Available: http://iperf.sourceforge.net/

[52] N. Egi, A. Greenhalgh, M. Handley, M. Hoerdt, F. Huici, and L. Mathy, "Towards high performance virtual routers on commodity hardware," in *Proceedings of the 2008 ACM CoNEXT Conference*, ser. CoNEXT '08. New York, NY, USA: ACM, 2008, pp. 20:1–20:12. [Online]. Available: http://doi.acm.org/10.1145/1544012.1544032

[53] A. Whitaker, M. Shaw, and S. D. Gribble, "Scale and performance in the Denali isolation kernel," *SIGOPS Oper. Syst. Rev.*, vol. 36, no. SI, pp. 195–209, 2002.

[54] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek, "The Click modular router," *ACM Trans. Comput. Syst.*, vol. 18, no. 3, pp. 263–297, 2000.

[55] L. Rizzo, "Revisiting network i/o apis: The netmap framework," *Queue*, vol. 10, no. 1, pp. 30:30–30:39, Jan. 2012. [Online]. Available: http://doi.acm.org/10.1145/2090147.2103536

[56] R. Bolla, R. Bruschi, G. Lamanna, and A. Ranieri, "Drop: An open-source project towards distributed sw router architectures," in *Global Telecommunications Conference, 2009. GLOBECOM 2009. IEEE*, 30 2009-dec. 4 2009, pp. 1–6.

[57] A. Bianco, R. Birke, L. Giraudo, and N. Li, "Multistage software routers in a virtual environment," in *Proceedings of the IEEE Global Telecommunications Conference (GLOBECOM 2010)*, Miami, Florida, December 2010.

[58] F. Bellard, "Qemu, a fast and portable dynamic translator," in *Proceedings of the 2005 USENIX Annual Technical Conference*, Anaheim, CA, April 2005.

[59] R. Russel, "virtio: Towards a de-facto standard for virtual i/o devices," *ACM SIGOPS Operating Systems Review*, vol. 42, no. 5, 2008.

[60] L. Abeni and G. Buttazzo, "Integrating multimedia applications in hard real-time systems," in *Proceedings of the IEEE Real-Time Systems Symposium*, Madrid, Spain, December 1998.

[61] D. Faggioli, F. Checconi, M. Trimarchi, and C. Scordino, "An EDF scheduling class for the Linux kernel," in *Proceedings of the Eleventh Real-Time Linux Workshop*, Dresden, Germany, September 2009.

[62] L. Rizzo and G. Lettieri, "Vale, a switched ethernet for virtual machines," in *Proceedings of the 8th international conference on Emerging networking experiments and technologies*, ser. CoNEXT '12. New York, NY, USA: ACM, 2012, pp. 61–72. [Online]. Available: http://doi.acm.org/10.1145/2413176.2413185

[63] A. Bianzino, L. Chiaraviglio, and M. Mellia, "Grida: A green distributed algorithm for backbone networks," in *Online Conference on Green Communications (GreenCom), 2011 IEEE*, sept. 2011, pp. 113–119.

[64] T. C. Group, "Smart 2020: Enabling the low carbon economy in the information age," *M. Webb*, June 2008.

[65] G. A. Plan, "An inefficient truth," Global Action Plan Report, Tech. Rep., December 2007. [Online]. Available: http://globalactionplan.org.uk

[66] R. Bolla, R. Bruschi, F. Davoli, and F. Cucchietti, "Energy efficiency in the future internet: A survey of existing approaches and trends in energy-aware fixed network infrastructures," *Communications Surveys Tutorials, IEEE*, vol. 13, no. 2, pp. 223–244, quarter 2011.

[67] A. Adelin, P. Owezarski, and T. Gayraud, "On the impact of monitoring router energy consumption for greening the internet," in *Grid Computing (GRID), 2010 11th IEEE/ACM International Conference on*, oct. 2010, pp. 298–304.

[68] A. Bianzino, C. Chaudet, D. Rossi, and J.-L. Rougier, "A survey of green networking research," *Communications Surveys Tutorials, IEEE*, vol. 14, no. 1, pp. 3–20, quarter 2012.

[69] I. Cidon and I. Keidar, "Zooming in on network-on-chip architectures," *Electrical Engineering*, vol. 565, p. 10, 2009. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.66.9010&rep=rep1&type=pdf

[70] L. Benini and G. D. Micheli, "Networks on Chips: A new SoC paradigm," *IEEE Computer*, vol. 35, no. 1, pp. 70–78, 2002.

[71] T. Bjerregaard and S. Mahadevan, "A survey of research and practices of network-on-chip," *ACM Computing Surveys*, vol. 38, no. 1, 2006.

[72] S. Vangal, J. Howard, G. Ruhl, S. Dighe, H. Wilson, J. Tschanz, D. Finan, A. Singh, T. Jacob, S. Jain, V. Erraguntla, C. Roberts, Y. Hoskote, N. Borkar, and S. Borkar, "An 80-tile sub-100-w teraflops processor in 65-nm cmos," *Solid-State Circuits, IEEE Journal of*, vol. 43, no. 1, pp. 29–41, jan. 2008.

[73] J. Stine, N. Carter, and J. Flich, "Comparing adaptive routing and dynamic voltage scaling for link power reduction," *Computer Architecture Letters*, vol. 3, no. 1, p. 4, january-december 2004.

[74] K. Choi, R. Soma, and M. Pedram, "Fine-grained dynamic voltage and frequency scaling for precise energy and performance tradeoff based on the ratio of off-chip access to on-chip computation times," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 24, no. 1, pp. 18–28, 2005.

[75] R. Jejurikar, C. Pereira, and R. Gupta, "Leakage aware dynamic voltage scaling for real-time embedded systems," in *Proceedings of the 41st annual Design Automation Conference*, New York, USA, 2004.

[76] C. hsing Hsu and W. chun Feng, "Effective dynamic voltage scaling through CPU-boundedness detection," in *Power-Aware Computer Systems*, 2004, pp. 135–149.

[77] L. Shang, L.-S. Peh, and N. K. Jha, "Dynamic voltage scaling with links for power optimization of interconnection networks," in *9th International Symposium on High-Performance Computer Architecture*, Washington, DC, USA, 2003.

[78] J. Kim, M. A. Horowitz, F. Vdd, F. Vdd, and V. Vctrl, "Adaptive supply serial links with sub-1v operation and per-pin clock recovery," in *Proc. International Solid-State Circuits Conference*, San Francisco, USA, 2002.

[79] G. yeon Wei, J. Kim, D. Liu, S. Sidiropoulos, and M. A. Horowitz, "A variable-frequency parallel i/o interface with adaptive power-supply regulation," *IEEE J. Solid-State Circuits*, vol. 35, no. 11, pp. 1600–1610, 2000.

[80] P. Ghosh, A. Sen, and A. Hall, "Energy efficient application mapping to noc processing elements operating at multiple voltage levels," in *Proceedings of the 2009 3rd ACM/IEEE International Symposium on Networks-on-Chip*, Washington, DC, USA, 2009.

[81] U. Y. Ogras, R. Marculescu, P. Choudhary, and D. Marculescu, "Voltage-frequency island partitioning for gals-based networks-on-chip," in *Design Automation Conference*, 2007.

[82] F. G. Moraes, N. Calazans, A. Mello, L. Moller, and L. Ost, "Hermes: an infrastructure for low area overhead packet-switching networks on chip," *Integration*, vol. 38, pp. 69–93, 2004.

[83] M. Mirza-Aghatabar, S. Koohi, S. Hessabi, and M. Pedram, "An empirical investigation of mesh and torus noc topologies under different routing algorithms and traffic models," in *10th Euromicro Conference on Digital System Design Architectures, Methods and Tools*, Washington, DC, USA, Aug. 2007.

[84] J. Hu and R. Marculescu, "Energy- and performance-aware mapping for regular NoC architectures," *IEEE Transactions on computer-aided design of intergrated circuits and systems*, vol. 24, no. 4, pp. 551–562, 2005.

[85] S. Bhat, "Energy models for network on chip components," Ph.D. dissertation, Technische Universiteit Eindhoven, 2005.

[86] B. Zhai, D. Blaauw, D. Sylvester, and K. Flautner, "The limit of dynamic voltage scaling and insomniac dynamic voltage scaling," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 13, no. 11, pp. 1239–1252, 2005.

[87] F. G. Moraes, N. L. V. Calazans, A. V. de Mello, and L. C. Ost, "Evaluation of routing algorithms on mesh based NoCs," Faculdade de informatica pucrs - Brazil, Tech. Rep., 2004.

[88] W. J. Dally and B. Towles, *Principles and practices of interconnection networks*, D. E.M.Penrose, Ed. Morgan Kaufmann, 2003.

[89] M. Rahman, Y. Sato, and Y. Inoguchi, "High performance hierarchical torus network under adverse traffic patterns," in *Computer and Information Technology (ICCIT), 2010 13th International Conference on*, dec. 2010, pp. 210–215.

[90] [Online]. Available: http://www.gnu.org/software/glpk/

[91] [Online]. Available: http://www.systemc.org/downloads/standards/

[92] A. Bianco, P. Giaccone, and N. Li, "Exploiting dynamic voltage and frequency scaling in networks on chip," in *IEEE 13th International Conference on High Performance Switching and Routing (HPSR)*, June 2012.

[93] S. Noh, V.-D. Ngo, H. Jao, and H.-W. Choi, "Multiplane virtual channel router for network-on-chip design," in *First International Conference on Communications and Electronics (ICCE'06)*, Oct. 2006, pp. 348–351.

[94] P. Salihundam, S. Jain, T. Jacob, S. Kumar, V. Erraguntla, Y. Hoskote, S. Vangal, G. Ruhl, and N. Borkar, "A 2 tb/s 6 4 mesh network for a single-chip cloud computer with dvfs in 45 nm cmos," *Solid-State Circuits, IEEE Journal of*, vol. 46, no. 4, pp. 757–766, april 2011.

[95] A. Kahng, B. Li, L.-S. Peh, and K. Samadi, "Orion 2.0: A fast and accurate noc power and area model for early-stage design space exploration," in *Design, Automation Test in Europe Conference Exhibition, 2009. DATE '09.*, april 2009, pp. 423–428.

[96] S. Vangal, A. Singh, J. Howard, S. Dighe, N. Borkar, and A. Alvandpour, "A 5.1GHz 0.34mm2 router for network-on-chip applications," in *IEEE Symposium on VLSI Circuits*, June 2007, pp. 42–43.

[97] (2009) International Technology Roadmap for Semiconductors (ITRS), Process Integration, Devices, and Structures Chapter. [Online]. Available: http://www.itrs.net

[98] N. Eisley and L.-S. Peh, "High-level power analysis for on-chip networks," in *Proceedings of CASES 2004*. New York, NY, USA: ACM, 2004, pp. 104–115.

[99] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek, "The Click modular router," *ACM Trans. Comput. Syst.*, vol. 18, pp. 263–297, August

2000.

[100] M. Handley, O. Hodson, and E. Kohler, "XORP: an open platform for network research," *SIGCOMM Comput. Commun. Rev.*, vol. 33, pp. 53–57, January 2003.

[101] A. Bianco, J. Finochietto, M. Mellia, F. Neri, and G. Galante, "Multistage switching architectures for software routers," *Network, IEEE*, vol. 21, no. 4, pp. 15–21, July–August 2007.

[102] A. Bianco, F. G. Debele, and L. Giraudo, "On-line energy saving in a distributed multistage router architecture," in *Communications (GCC), 2012 IEEE International Conference on.* IEEE, December 2012, pp. 1–6.

[103] ——, "Energy saving in distributed router architectures," IEEE, pp. 1–5, June 2012.

[104] L. Barroso, J. Dean, and U. Holzle, "Web search for a planet: The Google cluster architecture," *Micro, IEEE*, vol. 23, no. 2, pp. 22–28, March-April 2003.

[105] "Windows clustering." [Online]. Available: http://technet.microsoft.com/en-us/library/cc757731(v=ws.10)

[106] "Domino 8.0 administration." [Online]. Available: http://publib.boulder.ibm.com/infocenter/domhelp/v8r0/index.jsp

[107] A. Bianco, R. Birke, D. Bolognesi, J. Finochietto, G. Galante, M. Mellia, M. Prashant, and F. Neri, "Click vs. Linux: two efficient open-source IP network stacks for software routers," in *High Performance Switching and Routing, 2005. HPSR. 2005 Workshop on*, May 2005, pp. 18–23.

[108] M. Dobrescu, N. Egi, K. Argyraki, B. Chun, K. Fall, G. Iannaccone, A. Knies, M. Manesh, and S. Ratnasamy, "RouteBricks: Exploiting parallelism to scale software routers," in *ACM SOSP*, 2009, pp. 15–28.

[109] "Approximate desktop, notebook, & netbook power usage." [Online]. Available: http://www.upenn.edu/computing/provider/docs/hardware/powerusage.html

[110] "IBM ILOG CPLEX Optimization Studio." [Online]. Available: http://www-01.ibm.com/software/integration/optimization/cplex-optimization-studio/