11-7-2012

Host and Network Optimizations for Performance Enhancement and Energy Efficiency in Data Center Networks

Hao Jin
Florida International University, hjin001@fiu.edu

DOI: 10.25148/etd.FI12111906

FLORIDA INTERNATIONAL UNIVERSITY

Miami, Florida

HOST AND NETWORK OPTIMIZATIONS FOR PERFORMANCE ENHANCEMENT AND ENERGY EFFICIENCY IN DATA CENTER NETWORKS

A dissertation submitted in partial fulfillment of the

requirements for the degree of

DOCTOR OF PHILOSOPHY

in

ELECTRICAL ENGINEERING

by

Hao Jin

2012

To: Dean Amir Mirmiran
College of Engineering and Computing

This dissertation, written by Hao Jin, and entitled Host and Network Optimizations for Performance Enhancement and Energy Efficiency in Data Center Networks, having been approved in respect to style and intellectual content, is referred to you for judgment.

We have read this dissertation and recommend that it be approved.

Niki Pissinou

Jean Andrian

Jason Liu

Deng Pan, Major Professor

Date of Defense: November 7, 2012

The dissertation of Hao Jin is approved.

Dean Amir Mirmiran
College of Engineering and Computing

Dean Lakshmi N. Reddi
University Graduate School

Florida International University, 2012

DEDICATION

To my wife and my parents.

ACKNOWLEDGMENTS

I would like to express my sincerest gratitude to my major professor Dr. Deng Pan. Without his insightful advice and consistent encouragement throughout my Ph.D. study, this work would not have been possible. I am extremely grateful for his patient, continued guidance at the beginning stage of my Ph.D. research. His hardworking and never-ending pursuit of excellence has been and will always be a tremendous source of inspiration to me.

I would like to thank Dr. Pissinou, Dr. Kang Yen and the Telecommunication and Information Technology Institute (IT^2) for offering me the opportunity to pursue my doctoral degree at FIU. I also want to thank my dissertation committee members, Dr. Jean Andrian and Dr. Jason Liu, for their enlightening comments and suggestions. I sincerely thank the Department of Electrical and Computer Engineering (ECE) and the School of Computing and Information Sciences (SCIS) for the financial supports of my Ph.D. study.

My special thanks go to all the lab mates in IT^2 and SCIS. In particular, I want to thank Dr. Qian Wang, Dr. Kai Chen, Xinyu Jin, Pasd Putthapipat, Yubin Li, Dr. Xiaowen Zhang and Yu Li for their help and friendship. My appreciation also goes to Olga Carbonell, Pat Brammer, Maria Benincasa for their kind administrative support. Thanks to Dr. Geoffrey Smith, Professor of SCIS for developing and sharing the disseration Latex template.

Finally, and most importantly, I want to thank my wife and my parents for their unconditional love. Their company and understanding has been a imence motivation of my Ph.D. life.

ABSTRACT OF THE DISSERTATION

HOST AND NETWORK OPTIMIZATIONS FOR PERFORMANCE ENHANCEMENT AND ENERGY EFFICIENCY IN DATA CENTER NETWORKS

by

Hao Jin

Florida International University, 2012

Miami, Florida

Professor Deng Pan, Major Professor

Modern data centers host hundreds of thousands of servers to achieve economies of scale. Such a huge number of servers create challenges for the data center network (DCN) to provide proportionally large bandwidth. In addition, the deployment of virtual machines (VMs) in data centers raises the requirements for efficient resource allocation and find-grained resource sharing. Further, the large number of servers and switches in the data center consume significant amounts of energy. Even though servers become more energy efficient with various energy saving techniques, DCN still accounts for 20% to 50% of the energy consumed by the entire data center.

The objective of this dissertation is to enhance DCN performance as well as its energy efficiency by conducting optimizations on both host and network sides. First, as the DCN demands huge bisection bandwidth to interconnect all the servers, we propose a parallel packet switch (PPS) architecture that directly processes variable length packets without segmentation-and-reassembly (SAR). The proposed PPS achieves large bandwidth by combining switching capacities of multiple fabrics, and it further improves the switch throughput by avoiding padding bits in SAR. Second, since certain resource demands of the VM are bursty and demonstrate stochastic nature, to satisfy both deterministic and stochastic demands in

VM placement, we propose the Max-Min Multidimensional Stochastic Bin Packing (M^3SBP) algorithm. M^3SBP calculates an equivalent deterministic value for the stochastic demands, and maximizes the minimum resource utilization ratio of each server. Third, to provide necessary traffic isolation for VMs that share the same physical network adapter, we propose the Flow-level Bandwidth Provisioning (FBP) algorithm. By reducing the flow scheduling problem to multiple stages of packet queuing problems, FBP guarantees the provisioned bandwidth and delay performance for each flow. Finally, while DCNs are typically provisioned with full bisection bandwidth, DCN traffic demonstrates fluctuating patterns, we propose a joint host-network optimization scheme to enhance the energy efficiency of DCNs during off-peak traffic hours. The proposed scheme utilizes a unified representation method that converts the VM placement problem to a routing problem and employs depth-first and best-fit search to find efficient paths for flows.

TABLE OF CONTENTS

CHAPTER PAGE

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 Overview

Data centers, which provide enormous computing power and data storage space [NUM], have become more and more important for today's economy [MY10]. During the last decade, large corporations, like Microsoft [GHJ+09] and Google [AMW+10], have kept expending the scale of their data centers to accommodate the ever increasing needs for more computational power. Large data centers, such as Microsoft's Chicago data center, are reported to hold over 300,000 servers. Such fast expansion creates serious performance and efficiency challenges for the data center network (DCN) to provide sufficient and efficient network connectivity for the servers [AFLV08] [GLM+08]. For example, the massive deployment of bandwidth intensive applications and services, such as MapReduce and Internet video streaming, is demanding for larger and more cost-efficient DCN bandwidth. Further, virtual machines (VMs) significantly improve the efficiency of data centers by sharing resources of physical servers. However, such benefit highly relies on efficient resource allocation and find-grained resource sharing. In addition, data centers can be easily listed as one of the top local electricity consumers, and DCN accounts for a notable portion of the data center's total energy consumption. Thus, to increase DCN's energy efficiency is critical for the current and future data center designs. This dissertation aims to address these challenges by enhancing the DCN's performance and energy efficiency, so that the DCN can adapt to the rapid growth of the data center scale.

1.2 Motivations

In order to provide high performance network services, data centers gather large numbers of servers together at a central location and interconnect them with high

speed switches. As the size of the data center has grown dramatically, the DCN faces several performance and efficiency challenges. Bandwidth intensive applications, such as search engine and cloud data storage, are more and more popular in data centers. As a result, DCN needs to expand its network capacity continuously to keep pace with the data center scale. Another recent development of data centers is the wide deployment of VMs. Because multiple VMs can share the resources of the same physical server, the server utilization of data centers can be greatly improved. However, as the number of VMs getting larger, the resource allocation task becomes more complicated. It is impossible for the DCN operators to manually allocate resources for tens of thousand VMs, and at the same time optimize the overall server utilization. Finally, due to the huge energy consumption of data centers, the DCN needs to be more energy efficient so that the data centers can be both cost-effective and environmentally sustainable.

1.2.1 Needs for Larger Network Capacity

The bandwidth demands in DCNs are dramatically increasing. There are several factors that have contributed to such increase. First, bandwidth intensive applications become more and more popular in people's daily life. For example, as the quality of the online video getting better and the price getting lower, more people choose to watch videos online rather than renting discs from local video stores. Another example is the cloud based computing, in which clients rely on adequate network bandwidth to work properly. Second, the wide deployment of VMs intensifies the network usage of DCNs. As the server capacity and resource separation technique greatly improved, it is not difficult to host tens even hundreds of VMs on a single server. As a result, VMs may experience constant network congestions if the servers fall short on bandwidth resources. In addition, many distributed computing appli-

cations, such as MapReduce, may create significant amount of inter-communication traffics among servers and VMs, which further aggravate DCN's bandwidth pressure.

Traditional single switching fabric based switches are more and more difficult to meet such rapid growing bandwidth demands. This is mainly because of the slower growth pace of the switch memory access speed when comparing to the growth pace of the switch line speed. It is found that the speed of DRAM increase 10% every 18 months while the switch line speed increases 100% every 7 months [AC04]. Switches use DRAM to implement most of their large buffers in order to be cost-efficient. As a result, if the line speed keeps increasing, the switch may not be able to buffer packets as fast as they arrive and depart.

A popular solution to this challenge is the Parallel Packet Switch (PPS) [IM03] architecture, which combines several lower-speed switching fabrics or center stage switches to provide huge aggregate bandwidth. Since it uses only inexpensive commodity switches, PPS can greatly reduce the cost of the DCN while providing sufficient bandwidth. Most existing PPSs handle only fixed length packets, also called cells. Since packets in data centers are of variable length, existing PPSs need segmentation-and-reassembly (SAR) to process such packets, which will introduce padding bits and waste precious bandwidth. Therefore, it becomes highly necessary to design a PPS architecture that directly supports variable length packet.

1.2.2 Needs for Effective Resource Allocation and Management

VMs are attractive to modern data centers because they may significantly promote the efficiency and flexibility [BKB07]. However, such an incentive highly relies on an effective VM management scheme [SMLF09]. This is because an ineffective VM management scheme may result in lower resource utilization and thus needs more physical servers, which will further lead to not only higher capital investments on

equipment and facilities, but also increased operational expenditures on energy and labor. In addition, since the number of VMs in the data center increase significantly, manual VM management is no longer feasible. The key of an effective VM management is how to efficiently allocate physical resource to VMs so that each VM obtains its required share, and meanwhile, the overall server utilization is maximized. VM placement, which assigns the host server for each VM, is the primary stage where the physical resource allocations take place. Therefore, a well-designed VM placement scheme is critical for DCN to effectively manage a large number of VMs.

Multiple VM characteristics need to be considered in finding the VM placement. First, each VM may have demands on various server resources. CPU and memory are traditionally the two major criteria. More recently, due to the increasing concerns on data center energy [AMW+10] and emerging bandwidth intensive applications [BAM10], VMs' power consumption and bandwidth requirement are also taken into account when computing the placement. Second, some of the VM's resource demands may be highly bursty and time varying. The real demands of these resources are fluctuating, and it is difficult to obtain an accurate fixed-value measure. One such example is the network bandwidth. It is shown that such bursty bandwidth demands of VMs in data centers can be approximated by certain stochastic processes [KSG+09]. As a result, besides supplying fixed-value resources requested by VMs, servers need to provide an availability guarantee for such *stochastic* resources in the form of a violation probability threshold, specified in the data center's service level agreement (SLA). The threshold gives the worst-case likelihood that a server cannot satisfy the dynamical demands of a VM for stochastic resources.

1.2.3 Needs for Fine Granularity Performance Guarantee

In data centers, virtualization technology is heavily deployed. It requires resources, such as network bandwidth, to be allocated and shared at fine granularity [CIM05]. In particular, bandwidth allocation or bandwidth provisioning on switches can be on different levels, namely port level and flow level. Port level bandwidth provisioning assures bandwidth for the traffic from an input port to an output port, by which switches can support traffic isolation between VLANs. Since a switch port may belong to a single or multiple VLANs, bandwidth provisioning at the port level ensures the bandwidth of each VLAN and makes one VLAN transparent to the other. On the other hand, flow level bandwidth provisioning ensures bandwidth for an individual flow, which may be a subset of the traffic from the input port to the output port. A flow may be the sequence of packets generated by a specific application or departing from an IP address, and in general can be flexibly defined by a combination of the twelve packet header fields [MSA+06].

Flow level bandwidth provisioning is particularly necessary and important for virtualization based DCNs, as it differentiates traffic at sufficiently fine granularity [GKP+08]. In such a DCN, multiple VMs reside in a single physical server, and their traffic shares the same physical network adapter and is correspondingly fed into the same switch port. Since flows from different VMs may require different bandwidth allocations and delay guarantees, port level bandwidth provisioning is no longer sufficient [CIM05]. In contrast, flow level bandwidth provisioning is able to allocate bandwidth resource on a per flow basis, so that each flow can have its guaranteed bandwidth and thus guaranteed delay performance.

1.2.4 Needs for Better Energy Efficiency

As the data center size increases dramatically, energy efficiency becomes a critical metric in the DCN design [dat]. It is estimated that total 44 million servers of data centers all over the world consume 0.5% of the total electricity [Kat09], and 20% to 50% of the total data center electricity is consumed by the DCN [SLX10] [AMW+10]. However, nowadays, DCNs suffer from low energy efficiency, especially during off-peak hours. With the huge number of servers in a data center, the DCN needs proportionally large bandwidth to interconnect the servers. In addition, a DCN is typically provisioned with full bisection bandwidth [GWT+08] to support burst all-to-all communication. However, since DCN traffic demonstrates fluctuating patterns, the fully provisioned bandwidth cannot be always well utilized during off-peak hours, resulting in resource underutilization and energy waste. Specifically, due to the non-linearity relationship between the load and power consumption of the devices, including servers and switches, idle devices still consume as much as 60% of their peak power [Kat09]. Thus, an energy efficient network design for DCNs is highly desirable. Such design should consolidate the services and traffics during the off-peak period to avoid the huge energy overhead.

1.3 Research Objectives

The primary objective of this dissertation is to design a performance and energy efficiency enhancement scheme for the DCN, so that as the data center size getting larger and larger, the DCN can still provide sufficient bandwidth, manage the resources effectively and achieve high energy efficiency.

1. Traditional single switching fabric based switches are more and more difficult to meet the increasing bandwidth demand in the DCN. Thus, our first goal is to design a PPS architecture that combines multiple low-speed switches to

provide larger aggregate bandwidth. In addition, the parallel packet switch should also be able to directly process variable length packets to avoid bandwidth wastes on the padding bits of the fixed length cells.

2. VMs in data centers may demands multiple types of physical resources, and some of the resource demands can be bursty and be modeled as stochastic processes. Thus, our second goal is to design an efficient VM placement scheme which achieves better server utilization while satisfying multiple deterministic and stochastic demands.

3. Since multiple VMs may share the bandwidth of a single physical network adapter, port level traffic isolation is insufficient to guarantee performance for each VM. Thus, the third goal of this dissertation is to design a practical flow level performance guarantee scheme which ensures the provisioned bandwidth and delay performance for each individual flow in the DCN.

4. During the off-peak traffic hours, DCNs suffers from low energy efficiency and thus huge energy waste. Thus, the last goal of this dissertation is to design an optimization scheme to improve the energy efficiency of the DCN during off-peak traffic time.

1.4 Our Contributions

Following the research goals, we propose a joint host-network optimization scheme to enhance performance and energy efficiency of the DCN. We have evaluated the performance of the proposed scheme with theoretical analysis, program simulations and realistic prototypes. In specific, we have made the following contributions.

1. Variable Length Parallel Packet Switch to Increase Switch Bandwidth [JPP11]

To effectively and cost-efficiently increase the DCN's network capacity, we propose a variable length PPS (vPPS) architecture that directly handles variable-length packets without SAR and with low hardware cost.

- We investigate a simplified 1×1 vPPS which is similar to the traditional inverse multiplexing system. We show that two additional buffers, namely input conversion buffer (ICB) and output conversion buffer (OCB), are required to accommodate the rate difference between the input/output line and the center stage switch (CSS).
- We design two different scheduling policies to limit the size of the ICB and OCB for the simplified 1×1 vPPS, respectively. We show that both ICB size and OCB size can be bounded by $2L$, where L is the maximum packet length. Moreover, we prove that the second policy enables the switch to emulate an FIFO OQ switch.
- We investigate the general $N \times N$ vPPS by expanding the 1×1 switch structure and combining its two scheduling policies. We design a scheduling policy based on the policies from the simplified 1×1 vPPS case to limit the size of ICB and OCB, respectively. We prove that the presented vPPS architecture with the proposed scheduling policy can emulate an FIFO OQ switch with speedup of 2.

2. Multidimensional Stochastic VM Placement to Improve Resource Utilization [JPXP12]

To improve resource utilizations in DCNs with various deterministic and stochastic resources, we propose a multidimensional stochastic VM placement scheme.

- We model the VM placement with multiple deterministic and stochastic resources as a Multidimensional Stochastic VM Placement (MSVP) problem, with the objective to minimize the number of required servers while satisfy the service level agreement (SLA) availability guarantee. We prove that this problem is NP-hard.
- We propose a polynomial time algorithm named Max-Min Multidimensional Stochastic Bin Packing (M^3SBP) to solve this problem. The basic idea is to maximize the minimum utilization ratio of all the resources of a server, while satisfying the VMs' demands for both deterministic and stochastic resources.
- We demonstrate by extensive simulations that that M^3SBP guarantees the availability requirement for the stochastic resource while employing fewer servers than other benchmark algorithms do. We also show that, compared to the modified deterministic algorithms that simply implement over-provisioning for stochastic resources, M^3SBP obtains more efficient placement schemes.

3. Flow Level Network Performance Guarantee [JPLP12]

To provide the fine-grained performance assurance in the DCN, we propose a flow level traffic scheduling technique to provision bandwidth for each individual flow.

- We propose the Flow-level Bandwidth Provisioning (FBP) algorithm, which assures the provisioned bandwidth and thus delay guarantees for each individual flows in the DCN. FBP reduces the switch scheduling problem to multiple instances of fair queuing problems, each utilizing a well studied fair queuing algorithm. As a result, FBP can closely emulate the ideal Generalized Processor Sharing (GPS) model and accurately guarantee the provisioned bandwidth.
- We theoretically analyze the performance of FBP, and prove that it achieves constant service guarantees and tight delay guarantees. We prove that FBP is

economical to implement with bounded crosspoint buffer sizes and no speedup requirement, and is fast with low time complexity and distributed scheduling.

- We implement FBP in the OpenFlow software switch [MSA+06] and integrate FBP with the NOX controller as one of its components. We validate the constant service guarantees and tight delay guarantees by the empirical simulation results. We demonstrate by prototype experiment results that our prototype can accurately provision bandwidth at the flow level and is practical to implement in the DCN.

4. Joint Host-Network Optimization to Enhance DCN's Energy Efficiency [JCL+ed]

To improve the energy efficiency of the DCN during off-peak traffic time, we investigate the the optimization scheme which reduces the number of active devices in the network while maintaining the required network services. We propose a host-network energy efficiency co-optimization scheme for DCN that combines VM placement and flow routing optimization, so that the energy efficiency can be improved on both sides.

- We develop a unified representation method which transforms the VM placement problem to adapt the flow routing problem. We develop a topology-aware recursive multi-path routing algorithm which utilizes the depth first search algorithm to traverse the hierarchies of the DCN, and utilizes the best fit to find the most proper flow routing path. We introduce a parallel processing approach which divides the target data center into clusters and optimizes the clusters simultaneously.
- We build a 4-pod and 16-host, HP ProCurve OpenFlow switches and VMware vSphere based prototype based on the proposed optimization scheme. We

develop an Equinox framework bundle in the original Beacon controller to implement our optimization algorithm. We demonstrate the effectiveness of the proposed optimization scheme by empirical experiment results.

1.5 Dissertation Outline

The rest of the dissertation is organized as follows. Chapter 2 describes the background and related works. Chapter 3 presents the $N \times N$ vPPS architecture that directly handles variable-length packets without SAR and with low hardware cost. Chapter 4 investigates efficient VM placement problem in data centers with multiple deterministic and stochastic resources. In addition, Chapter 4 proposes a polynomial time algorithm to solve the problem, and evaluates the performance by theoretical analysis and prototype experiments. Chapter 5 studies the flow-level bandwidth provisioning problem for data centers with OpenFlow context. Chapter 5 also proposes an efficient flow-level bandwidth provisioning algorithm with constant service guarantees, and to experimentally demonstrate a practical flow-level bandwidth provisioning solution based on the OpenFlow protocol. Chapter 6 studies the energy efficiency optimization problem in data centers, and proposes a host-network energy efficiency co-optimization scheme which considers the VM and flow consolidation simultaneously. Further, Chapter 6 evaluates the performance of the proposed scheme via a OpenFlow hardware switch based prototype. Finally, Chapter 7 concludes this dissertation and provides directions for future works.

CHAPTER 2

BACKGROUND AND RELATED WORKS

In this chapter, we describe existing solutions that address the DCN challenges highlighted in the previous chapter.

2.1 Parallel Packet Switch Architecture

Parallel Packet Switch (PPS) architecture combines multiple parallel switching fabrics and provides huge aggregate bandwidth. In a PPS, when packets arrive at the input ports, they will be first distributed by a demultiplexor to one of the internal low-speed switches. The low-speed switch is selected according to the PPS's packet distribution policy whose major goal is to achieve PPS work conserving. After switching at the internal switch stages, packets will be aggregated by a multiplexor and forwarded to the output ports of the PPS. Similarly to the demulitplexor, the multiplexor employs a packet collection policy, which controls the packet collection order and time, to assure the work conserving property of the PPS.

Existing PPS designs in literatures can be divided into two categories, bufferless PPS and buffered PPS. A bufferless PPS uses no high-speed buffer at the demultiplexor or multiplexor. In this way, the PPS can greatly lower the hardware cost. This is because that the high-speed buffers are usually implemented by using Static Random Access Memory (SRAM), which is significantly expensive comparing to normal switch memories. In order to achieve work conserving, current bufferless PPS designs have to use speedup at the internal switches as a trade-off. A speedup of n means that the internal switch's line speed needs to be n times faster than the internal switch's switching fabric speed. [IAM00] proposes a bufferless PPS by using k OQ switches as internal switches, where k is the ratio between the external line rate and internal line rate. Each arriving packet is divided into fixed length

cells first, and then sent into the switch. [IAM00] also shows that, such a PPS can emulate a FIFO OQ switch with speedup of 2 [KK01] studies the minimum requirement for a general bufferless OQ PPS to work conserving. It obtains a tight lower bound for the number of internal OQ switches, k, and for the speedup s. [KK01] proves that $k \geq 2\lceil R/r \rceil - 1$ is necessary and sufficient for a bufferless PPS with port speed of R and internal switch speed of r to work conserving. In addition, [KK01] proves that to work conserving, the speedup s of a bufferless PPS should satisfy: $s \geq k/\lceil k/2 \rceil$.

To eliminate the speedup, buffered PPS architectures are studied. [IM03] shows that with extra buffer at each input and output port, the PPS eliminates the need for speedup when emulating the FIFO OQ switch. [AC04] employs virtual input queues (VIQ) at multiplexers and proposes corresponding cell dispatch-and-reassembly algorithms to eliminate the speedup. It also can achieve load balancing and in-order cell delivery. [LS06] proposes the multiple input-output-queued (MIOQ) switch that has two parallel center stage switches. It is showed that MIOQ switch can emulate an OQ switch with no speedup of any component.

2.2 Virtual Machine Placement Scheme

Several models and heuristics are proposed in recent literatures to improve DCN's resource utilization with effective virtual machine (VM) placement. Those proposed solutions can be divided into two categories, multiple resource demands VM placement and stochastic resource demands VM placement. The former takes account multi-type resource constraints in calculating VM placement, while the latter recognizes the bursty nature of certain resource demands and considers it in the placement.

2.2.1 Multiple Resource Demands

VM placement problem with multiple resource demands can be modeled as a variant of Multidimensional Bin Packing (MBP) problem. The classic MBP problem is to determine the way to pack items into the least number of size-fixed bins. There are extensive studies focusing on finding a fast and effective solution to this NP-hard problem. [CGJ97] provides a detailed survey. In VM placement problem, some of the resource requirements are hard-constraints while others are soft-constraints. To reflect such characteristic, [XF10] models VM placement as a multi-objective optimization problem. A fuzzy multi-objective evaluation aided genetic algorithm is proposed to search large solution space for large scale data centers. While [XF10] considers CPU, memory, power consumption and thermal dissipation as its placement criteria, [MPZ10] focuses their attention on network bandwidth. Their goal is to find an optimized VM placement scheme to improve the network scalability for traffic-intensive data centers. An approximation algorithm aiming to reduce the average traffic latency is proposed. The algorithm takes a two-tier approach that first divides VMs and servers into clusters respectively, and then matches VMs and servers at the cluster and server levels consequently.

2.2.2 Stochastic Resource Demands

An effective approach to deal with stochastic resource demands is to calculate an *equivalent bandwidth* to represent the the stochastic demands as closely as possible. [KRT00] focuses on allocating bandwidth for busty connections. [KRT00] models the stochastic bandwidth demands as Bernoulli type distributions and applies an modified bin packing algorithm to reduce the number of connection and to achieve load balancing. [GI99] proposes polynomial time approximation algorithms to solve the stochastic bin packing and load balancing problem for stochastic demands with Pois-

son or exponential distributions. [GI99] also proposes a quasi-polynomial approximation scheme for Bernoulli-distributed demands. [WMZ11] proposes a method that first calculates a total equivalent demand for all VMs that are hosted by the same server, and then compares the total equivalent demand with the server's capacity to determine whether a placement is valid. In [WMZ11], bandwidth demands are considered to follow Poisson distribution

2.3 Bandwidth Provisioning in DCNs

The current main approach of bandwidth provisioning on switches is to emulate PIFO OQ switches. In a PIFO OQ switch, all packets are buffered at output ports, either on a per input port or per flow basis. Each output port runs a fair queuing algorithm [DKS89] to emulate the ideal GPS model and provide guaranteed bandwidth for each output queue. OQ switches achieve the optimal performance but are not practical because they need speedup of N [PY09]. Based on the granularity level, existing bandwidth provisioning solutions can be divided into two categories, port-level and flow-level.

2.3.1 Port-level Bandwidth Provisioning

The following solutions provide port-level bandwidth provisioning. [MRS03] shows that a buffered crossbar switch with speedup of two satisfying non-negative slackness insertion, lowest-time-to-live blocking, and lowest-time-to-live fabric scheduling can exactly emulate a PIFO OQ switch. [MH03] proposes the *Modified Current Arrival First - Lowest Time To Leave First* scheduling algorithm, for a one-cell buffered crossbar switch with speedup of two to emulate a PIFO OQ switch without time stamps. [CIM05] shows that with speedup of two, a buffered crossbar switch can mimic a PIFO OQ switch with the restriction that the cells of an input-output pair depart in the same order as they arrive. [HSG+08] proposes the rate based *Smooth*

Multiplexing algorithm for a CICQ switch with a two-cell buffered crossbar, and shows that the algorithm provides bandwidth and throughput guarantees. [Tur09] presents the *Packet Group by Virtual Output Queue* and *Packet Least Occupied Output First* scheduling algorithms for buffered crossbar switches, and shows that they can emulate PIFO OQ switches with speedup of two or more. [SZ98] proposes the *Joined Preferred Matching* algorithm for CIOQ switches, and proves that the algorithm can emulate a general class of OQ service disciplines. [AHK10] proposes frame-based schedulers for Combined Input Output Queued (CIOQ) switches handling variable length packets to mimic an ideal OQ switch with bounded delay, and demonstrates a trade-off between the switch speedup and the relative queuing delay. [WYHL09] considers high-speed packet switches with optical fabrics, and proposes scheduling algorithms to provide performance guaranteed switching. [KHK09] introduces the Crosspoint Queued switch with large crosspoint buffers and no input queues, and proposes scheduling algorithms for it to emulate an ideal OQ switch.

2.3.2 Flow-level Bandwidth Provisioning

The following solutions provide flow-level bandwidth provisioning. [CIM05] shows that in order for a buffered crossbar switch with speedup of two to provide flow-level bandwidth provisioning, a separate crosspoint buffer must be available for each flow. Alternatively, the switch structure must first be modified with a more complicated buffering scheme (similar to that of OQ switches) and then a total of N^3 crosspoint buffers must be provided. Unfortunately, both schemes greatly increase the total number of crosspoint buffers and are not scalable. Another option is to increase the speedup of the crossbar to three, which will drop the maximum throughput of the switch by one third. The additional speedup of one is used to eliminate the crosspoint blocking. [CGMP99] proposes several algorithms for CIOQ switches with

speedup of two to emulate PIFO OQ switches. The *Critical Cell First* (CCF) algorithm needs N^2 iterations and global information. The *Delay Till Critical* (DTC) algorithm reduces the iteration number to N, but still needs global information. On the other hand, the *Group by Virtual Output Queue* (GBVOQ) algorithm does not need global information, but its iteration number is unbounded. [PY08] presents a scheme to achieve trade-offs between those in [CIM05] and [CGMP99]. It conducts distributed scheduling in the average case, but still needs speedup of two and N iterations in the worst case.

2.4 Energy Conservation Optimization

Existing energy saving solutions for DCNs can be divided into two broad categories: network-side optimization and host-side optimization. All solutions only focus on saving DCN energy from the perspective of their own side. As a result, the energy consumption of the other side of the DCN may increase significantly.

2.4.1 Network-Side Optimization

ElasticTree [HSM+10] proposes a DCN-wide power efficiency controller. The controller finds switch groups and flow routing paths to satisfy all of the network service requirements and minimize the overall power consumption. In order to improve the robustness of the DCN after the network optimization, ElasticTree leaves certain safety margin for each network link. It also builds a prototype to demonstrate the feasibility and scalability of the power efficiency controller. To save energy in wide area networks, GreeTE [ZYLZ10] proposes a flow routing path management scheme which utilizes the least number of switches to meet the performance demands, such as bandwidth and packet delay. When the flows are moved onto fewer numbers of switches, those idle switches are then powered off to save energy. [FSR10] studies the energy efficiency problem inside the bundle of links which usually interconnects

core layer switches. The link bundles are designed to provide high bandwidth capacity or to provide redundant routing path options. Based on the network traffic conditions, [FSR10] proposed a optimization scheme that reorganizes the flow routes and consolidates flows onto fewer number of links. Then [FSR10] shuts down the idle links and their associate line cards to save energy.

2.4.2 Host-Side Optimization

On the host side, the major energy saving approach is to increase the server utilization by consolidating VMs onto less number of physical servers. Thus those idle servers can be powered off to save energy. The VM consolidation is usually carried out by VM placement [MPZ10] [WMZ11] and VM live migration [CFH+05]. VM placement calculates the optimum VM locations that have the highest overall energy efficiency, and VM live migration moves the VMs to their new locations without interrupting their services. [MPZ10] identifies large traffic flows and then localize them to lower layer switches so that the switches on higher layers of the network may have less load and thus less energy consumption. In addition, [MPZ10] powers off idle switches and servers to further reduce the energy consumption. [WMZ11] shows that because of the burst characteristic of the network traffics, the VM placement problem in DCN can be modeled as stochastic bin packing problem. [WMZ11] proves the NP-hardness of the problem and proposes an approximation algorithm to find the placement result in polynomial time. Idle servers are shut down to save energy. Another approach to save energy on the host side for DCNs is to improve the hosts energy proportionality. PowerNap[MGW09] proposes a server energy saving scheme which quickly lowers the servers power when its traffic is low.

CHAPTER 3

PARALLEL PACKET SWITCH WITHOUT SEGMENTATION-AND-REASSEMBLY

This chapter investigates the challege of the DCN network capacity, to make it meet the ever increasing demands for more bandwidth at DCN's core switches and routers. We study the parallel packet switch (PPS) architecture, which combines multiple parallel switching fabrics and provide huge aggregate bandwidth. Most existing PPSs handle only fixed length packets, also called cells, mainly because traditional switching fabrics can process only cells. Since packets in the data centers are of variable length, existing PPSs need segmentation-and-reassembly (SAR) to process such packets, which will introduce padding bits and waste precious bandwidth. To tackle this bandwidth waste issue, we propose a PPS to directly handle variable-length packets without SAR.

First, We present a simplified 1×1 variable-length PPS (vPPS). We design the packet distribution and collection algorithms, and show that input and output conversion buffers are bounded by $2L$, where L is the maximum packet length. Next, we present a general $N \times N$ vPPS, and propose a corresponding packet scheduling algorithm. vPPS controls each packet's in- and out-queue time, and the in- and out-queue orders, so that every packet can be forwarded to its destination output port within a guaranteed maximum time. We prove that vPPS can emulate a first-in-first-out (FIFO) output queued (OQ) switch with speedup of two and with small, bounded extra packet buffers. In other words, vPPS achieves full combined switching speed with low hardware cost.

3.1 Introduction

With the booming of broadband multimedia applications, there is an ever increasing demand for more bandwidth at core routers. However, traditional single switching fabric based switches are more and more difficult to meet this bandwidth demand both technically and financially. A popular solution to this challenge is the PPS [IM03], which combines several lower-speed switching fabrics or center stage switches to provide huge aggregate bandwidth.

In Chapter 2, we have describe several exiting PPS designs. However, all of them are based on the assumption that packets are segmented into fixed length cells at the input and then reassembled back at the output. This SAR [McK99] process simplifies the switch design [MSS02] [LK10], but may significantly affect the switch performance [Tur09].

Recent advances in switching techniques have made it possible to directly process variable-length packets without SAR [Tur09]. Variable-length packet switches (or packet switches for short) have some unique advantages. First, packet switches can better utilize available bandwidth and achieve higher throughput. Cell switches may waste significant bandwidth on extra traffic including cell overheads and cell padding. In contrast, packet switches do not have such bandwidth waste. Second, since there is no segmentation and reassembly in packet switches, packets have shorter queuing delay than in cell switches. Therefore, packet switches reduce the latency that a packet experiences. Finally, no extra buffer space is needed at the input port or output port to segment and reassemble packets, which lowers hardware cost.

The goal of this chapter is to extend the packet switch design from single-fabric switches to PPSs, so that PPSs can utilize the advantages and achieve better performance with lower cost. Although there are a few studies on packet based PPSs, they

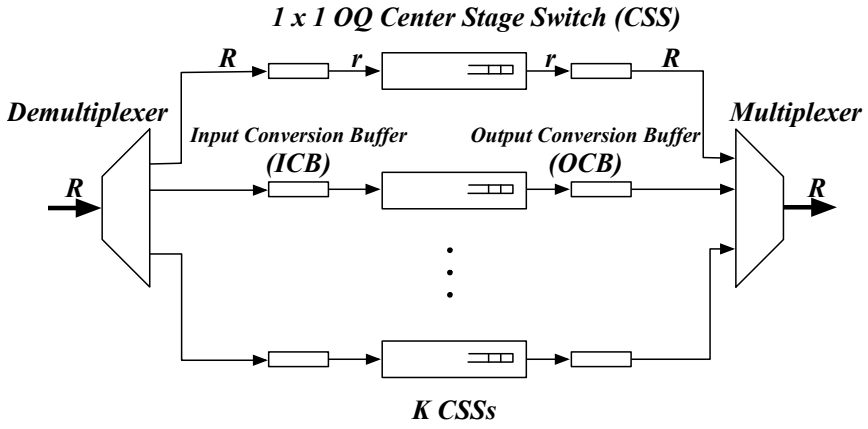


Figure 3.1: 1×1 Variable-Length Parallel Packet Switch

are not able to process arbitrary variable-length packets. [ZXZ05] handles variable-length packets by sending logical cells that belong to the same packet through the same switching plane. Although less overhead is needed, extra padding bits, which lower the overall throughput, are still required when the packet size is smaller than the cell size. [SXL07] proposes the Flow-Mapping PPS (FM-PPS) with flow level load balancing. It guarantees the packet order of each micro-flow and thus eliminates the costly resequencing. However, FM-PPS works only when packets can be organized as micro flows and it needs k buffers at each demultiplexer which increases hardware cost.

3.2 1×1 Variable-Length Parallel Packet Switch

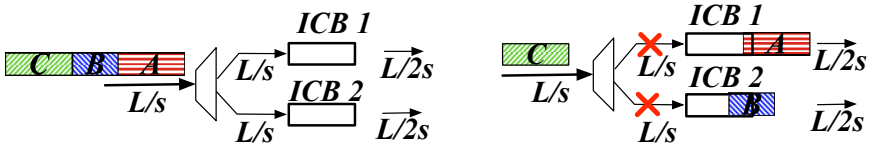
Before presenting the general vPPS, we first describe a simplified vPPS with one input and one output, which will be the basis to design the scheduling algorithms for the general case.

3.2.1 Switch Structure

A 1×1 vPPS, as shown in Figure 3.1, consists of a demultiplexer with bandwidth R, K 1×1 CSSs each with bandwidth $r = R/K$, and a multiplexer with bandwidth

R. The demultiplexer distributes variable-length packets to CSSs, from where the packets are collected by the multiplexer. Since a 1×1 switch needs no switching, the CSS in this case is simply a queue. The demultiplexer and the CSS have different bandwidth, and therefore each CSS needs an ICB to accommodate the speed difference. When the demultiplexer dispatches a packet to a CSS, it first sends the packet to the corresponding ICB, from where it will be retrieved by the CSS. If there are multiple packets in the ICB, they are stored as a first-in-first-out queue. Similarly, each CSS also has an OCB for the speed difference with the multiplexer. Note that the demultiplexer has no high-speed buffer to buffer the arriving packets, and similarly the multiplexer has no high-speed buffer to store the outgoing packets.

We are interested in the scheduling policies and conversion buffer sizes for the demultiplexer and multiplexer to be work conserving, i.e. keeping busy if there are packets to transmit. This seems trivial for fixed length cells, which can be easily accomplished with a round-robin packet distribution policy and L buffer space at each ICB and OCB, where L is the maximum packet length. However, with variable length packets, the problem becomes more challenging, which we will illustrate using the following example. Consider a 1×1 vPPS with two CSSs. The bandwidth of the demultiplexer and multiplexer is L/s, and that of the CSS is $L/2s$. Each ICB has L buffer space. Three packets A, B, and C with length of $L, 2L/3$, and L, respectively, arrive at the demultiplexer back to back, as shown in Figure 3.2(a). Without loss of generality, assume that at time $0s$ the demultiplexer start dispatching the first packet A to the first ICB, and the dispatch will finish at $1s$. Next, during $[1s, 5/3s]$, the demultiplexer dispatches the second packet B to the second ICB. Note that the first and second ICBs will not become empty earlier than $2s$ and $7/3s$, respectively. However, as shown in Figure 3.2(b), the demultiplexer finished dispatching packet B and should start dispatching the third one C at $5/3s$. Therefore, although the



(a) Time 0s: packet A arrives (b) Time 5/3s: packet C arrives

Figure 3.2: A 1×1 vPPS with ICBs of size L is not work conserving.

total bandwidth of the two CSSs is the same as that of the demultiplexer, but the demultiplexer cannot be work conserving with L ICB space.

In the rest of this section, we propose two scheduling policies for packet distribution and collection, and analyze the conversion buffer size bounds.

3.2.2 Policy A: ICB based Packet Distribution

This policy considers only packet distribution at the demultiplexer, and is based on the shortest queue first (SQF) algorithm. Specifically, when a new packet arrives, the demultiplexer checks the queue lengths of all the K ICBs, and selects the shortest queue to send the packet. If multiple ICBs have the same shortest length, the demultiplexer selects the one with the smallest index.

We will first analyze the characteristic of ICBs and prove later that, with Policy A, the size of any ICB is bounded by a small value, i.e. $2L$, while the demultiplexer is guaranteed to be work conserving. Denote the queue length of the ith ICB as $\widehat{B_i}$. (Since we always consider the values of different variables at the same time point, we omit the time parameter in the notation for simplicity.) Due to the space limitation, we only show the proofs of part of the lemmas and theorems.

Lemma 1 *The difference of queue lengths between any two ICBs is less than or equal to L, i.e.*

$$|\widehat{B_i} - \widehat{B_j}| \leq L \tag{3.1}$$

Proof. Assume the queue length difference between any two ICBs could be greater than L, or $\widehat{B_i} - \widehat{B_j} > L$. Since the length $\widehat{L_n}$ of the last packet n of the ith ICBs is less than or equal to largest packet size L, we have $(\widehat{B_i} - \widehat{L_n}) - \widehat{B_j} > L - \widehat{L_n} > L - L = 0$. In other words, when packet n was selecting its CSS, it did not choose the one with the shortest ICB length and this contradicts the scheduling policy. Hence the assumption is not possible. □

Based on whether all the CSSs are retrieving packets from their ICBs, we define two statuses. In the *partially-busy* status, at least one CSS is idle (with empty ICBs), while in the *fully-busy* status, all CSSs are busy (without empty ICBs).

Theorem 1 *In the partially-busy status, the queue length of any ICB is bounded by the largest packet length, L, i.e.*

$$\widehat{B_i} \leq L \tag{3.2}$$

Proof. In the partially-busy status, the minimum queue length of ICB equals to zero when the ICB is idle. Also from Lemma 1 we know that the maximum queue length difference between two queues are less than or equal to L. Thus the maximum ICB queue length is less than or equal to L when system is in the partially-busy status. □

Lemma 2 *In the fully-busy status, the total length of all ICBs is less than or equal to $(K-1)L$, i.e.*

$$\sum_{i=1}^{K} \widehat{B_i} \leq (K-1)L \tag{3.3}$$

Lemma 3 *In the fully-busy status, the maximum value of the minimum ICB queue length is less than or equal to $(1 - \frac{1}{K})L$, i.e.*

$$\min\{\widehat{B_i}\} \leq (1 - \frac{1}{K})L \tag{3.4}$$

Theorem 2 *In the fully-busy status, the queue length of any ICB is bounded by two times of the largest packet length,* $2L$*, i.e.*

$$\widehat{B_i} \leq 2L \tag{3.5}$$

Proof. Theorem 2 can be proved by Lemma 1 and 3. □

Theorem 3 *For a* 1×1 *vPPS adopting the SQF scheduling policy, the ICB queue length is bounded by* $2L$*, i.e.*

$$\widehat{B_i} \leq 2L \tag{3.6}$$

Proof. By Theorem 1 and 2, Theorem 3 is proved. □

3.2.3 Policy B: OCB based Packet Distribution and Retrieval

We now present the second policy for the 1×1 vPPS, which controls both the packet distribution and collection schedules, as well as the switching at CSS. First, we describe the detail processes and parameter definitions of each phase, namely *Packet Distribution, Switching at CSSs* and *Packet Collection.* Then we show by lemmas and theorems that by employing Policy B, the 1×1 vPPS emulates FIFO OQ switch with small bounded OCB size.

Packet Distribution

When packet n arrives, the demultiplexer distributes packets to different CSSs. The basic idea is that the demultiplexer selects the CSS i with the earliest ***O****CB* ***E****ntry* ***S****tart time* $\widehat{OES}_{n,i}$. If multiple CSSs have the same earliest OCB entry start time, the one with the smallest index will be selected.

The calculation of $\widehat{OES}_{n,i}$, which represents packet n's entry start time of the ith OCB, is different for different packet categories. To simplify the analysis, we

use a fixed time $\widehat{T}$ to represent the maximum delay from the input port to the CSS output queue. Denote the ***I**nput port **A**rrival time* of packet n as $\widehat{IA_n}$ and its ***CSS** output queue **A**rrival time* as $\widehat{CA_n}$. Then we have $\widehat{CA_n} = \widehat{IA_n} + \widehat{T}$.

For the first K packets, according to the policy, each of them selects the CSS with an empty OCB and with the same index. When these packets arrive at the CSS output queues, the OCB entry is available. However, they will not enter OCB but wait until time $\widehat{X}$, where $\widehat{X} = \widehat{CA_1} + \frac{L}{r}$. Thus, the first K packets have the same OCB entry start time, i.e.

$$\widehat{OES_{1,1}} = ... = \widehat{OES_{K,K}} = \widehat{X} = \widehat{CA_1} + \frac{L}{r} \tag{3.7}$$

On the other hand, when a packet after the first K one arrives at the CSS, the output queue may already have packets. Packet n will not start entering OCB until the last packet in the CSS output queue finishes its OCB entry. Denote the last packet in CSS i as packet m and its ***O**CB **E**ntry **F**inish time* as $\widehat{OEF_{m,i}}$. Then $\widehat{OES_{n,i}}$ is calculated as

$$\widehat{OES_{n,i}} = max(\widehat{CA_n}, \widehat{OEF_{m,i}}); \forall n > K \tag{3.8}$$

Then the demultiplexer selects the CSS which has the smallest OCB entry start time as the destination CSS for packet n.

Switching at CSSs

As mentioned before, the CSS of 1×1 PPS can be treated as a queue. Thus, when packet n arrives at the CSS, it will stay in the output queue until the OCB entry start time comes and then start to enter the OCB.

Packet Collection

Finally, the multiplexer collects packets from OCBs. Specifically, the multiplexer collects packets one by one according to their arrival order to the input port. Recall that the first packet enters the OCB at time $\widehat{X}$. The multiplexer will start to collect the first packet at time $\widehat{D}$, where $\widehat{D} = \widehat{X} + \frac{L}{r}$. Denote the ***O**CB **D**eparture **S**tart time* of packet n from the ith OCB as $\widehat{ODS}_{n,i}$. Therefore,

$$\widehat{ODS}_{1,i} = \widehat{D} = \widehat{X} + \frac{L}{r} \tag{3.9}$$

Lemma 4 *With Policy B, at any time after the OCB entry start time of the first packet, all OCB entries are busy.*

Lemma 5 *Packet n is already in the OCB when the multiplexer starts to collect it. In other words, packet n's OCB departure start time $\widehat{ODS}_{n,i}$ is greater than or equal to its OCB entry finish time $\widehat{OEF}_{n,i}$, i.e.*

$$\widehat{ODS}_{n,i} \geq \widehat{OEF}_{n,i} \tag{3.10}$$

Proof. Since the multiplexer collects packets by their arriving order at rate R, we can calculate the $\widehat{ODS}_{n,i}$ as,

$$\widehat{ODS}_{n,i} = \widehat{ODS}_{1,i} + \frac{\sum_{x=1}^{n-1} L_x}{R} \tag{3.11}$$

By (3.9), we have

$$\widehat{ODS}_{n,i} = \widehat{X} + \frac{L}{r} + \frac{\sum_{x=1}^{n-1} L_x}{R} \tag{3.12}$$

While

$$\widehat{OEF}_{n,i} = \widehat{OES}_{n,i} + \frac{L_n}{r} \leq \widehat{OES}_{n,i} + \frac{L}{r} \tag{3.13}$$

The OCB entry start time $\widehat{OES}_{n,i}$ of packet n is maximized when all K CSSs has the same OCB entry start time. Since all OCBs start the packet entry at the same time and are always busy, the maximal $\widehat{OES}_{n,i}$ is calculated by

$$\widehat{OES}_{n,i} \leq \widehat{OES}_{1,1} + \frac{\sum_{x=1}^{n-1} L_x}{Kr} \tag{3.14}$$

By (3.7) and (3.14), (3.13) becomes

$$\widehat{OEF}_{n,i} \leq \widehat{X} + \frac{\sum_{x=1}^{n-1} L_x}{R} \tag{3.15}$$

The lemma is proved by subtracting (3.15) from (3.12). □

Theorem 4 *The* 1×1 *vPPS with Policy B emulates a* 1×1 *FIFO OQ switch.*

Proof. The multiplexer collects packet one by one by their arriving order. In Lemma 5, we proved that every packet is ready in the OCB when the multiplexer starts to collect it. In other words, the multiplexer does not wait between two packet collections. Thus the switch is working conserving. On the other hand, the switch departure start time (OCB departure start time) of the first packet is bounded. Thus all packets leave the switch continuously with a bounded delay. Hence, the 1×1 vPPS with Policy B emulates an FIFO OQ switch. □

Theorem 5 *With Policy B, the queue length* $\widehat{C_i}$ *of any OCB is bounded by* $2L$*, i.e.*

$$\widehat{C_i} \leq 2L \tag{3.16}$$

Proof. The queue length of any OCB keeps increasing until the multiplexer collects packet from it. So when packet n is being collected by the multiplexer, the current queue length $\widehat{C_{n,i}}$ of the *ith* OCB equals to the differences between the total packets arrived $\widehat{E_{n,i}}$ and the total packets left $\widehat{D_{n,i}}$, i.e.

$$\widehat{C_{n,i}} = \widehat{E_{n,i}} - \widehat{D_{n,i}} \tag{3.17}$$

In Lemma 4, it is proved that all CSSs send packets to OCB continuously. Thus when packet n starts to depart from OCB i at time $\widehat{ODS_{n,i}}$, $\widehat{E_{n,i}}$ is calculated as

$$\widehat{E_{n,i}} = (\widehat{ODS_{n,i}} - \widehat{OES_{1,1}})r = \frac{\sum_{x=1}^{n-1} L_x}{K} + L$$

When the nth packet departs from the ith OCB, all of the previous packets entered the same OCB have already left from the OCB. Thus, $\widehat{D_{n,i}}$ should equal to the total length of all the previous packets in the same OCB, $\widehat{P_{n,i}}$. Since the OCB entry is always busy. $\widehat{P_{n,i}}$ is minimized when the OCB enter time of packet n is minimized. In this case, the output queue length of the ith CSS is L shorter than the queue lengths of other CSS. Then we have

$$\widehat{D_{n,i}} = \widehat{P_{n,i}} \geq \frac{\sum_{x=1}^{n-1} L_x - (K-1)L}{K} \tag{3.18}$$

Thus, by (3.18) and (3.18)

$$\widehat{C_{n,i}} = \widehat{E_{n,i}} - \widehat{D_{n,i}} = (2 - \frac{1}{K})L \leq 2L \tag{3.19}$$

Hence, the queue length of any OCB i is bounded by $2L$. □

3.3 General Variable-Length Parallel Packet Switch

In this section, we present the general vPPS. We first describe the switch architecture, scheduling algorithms and parameter definitions. Then, we show by analysis that vPPS can emulate an FIFO OQ switch with speedup of 2.

3.3.1 Switch Architecture

As shown in Figure 3.3, an $N \times N$ vPPS consists of N demultiplexers, $2K-1$ CSSs, and N multiplexers. The vPPS has bandwidth of R, and each CSS has bandwidth r, where $r = R/K$. Each demultiplexer acting as an input of the vPPS, distributes arriving packets to the CSSs. The packets are then transmitted through the CSSs, and finally collected by multiplexers, which act as outputs of vPPS. Similar

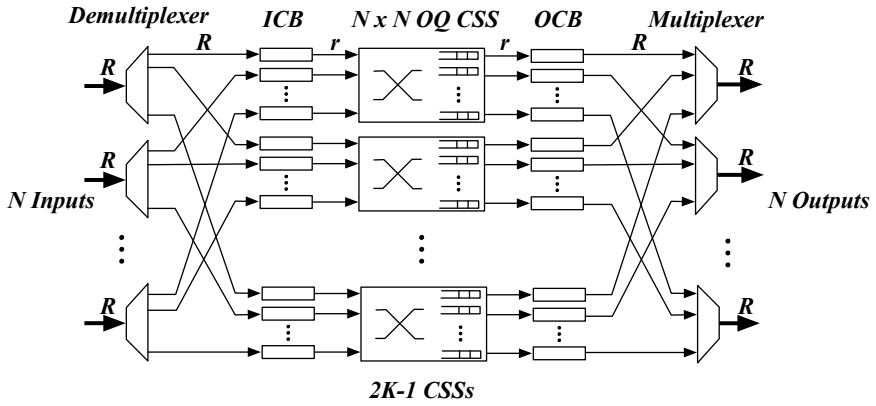


Figure 3.3: General Variable Length Parallel Packet Switch

with 1×1 vPPS, each input (output) of the CSS has an ICB (OCB) accommodate the bandwidth difference with the demultiplexer (multiplexer). Note that the demultiplexers and multiplexers do not have high speed buffers.

3.3.2 Scheduling Algorithms

The demultiplexers and multiplexers work as follows.

Packet Distribution

In general, the packet distribution in the vPPS can be divided into two stages, and each stage uses a policy similar to Policy A and Policy B of the 1×1 vPPS, respectively. In the first stage, the demultiplexer chooses K candidates out of all $2K - 1$ CSSs based on their ICB length. Specifically, when packet n arrives, the demultiplexer checks the ICB status of each CSS and then selects the K CSSs with the shortest ICB queue length as candidates. In the second stage, the demultiplexer chooses the final CSS from the K candidates based on their OCB entry start time. To be specific, the demultiplexer selects the CSS i with the earliest ***O****CB* ***E****ntry* ***S****tart time* $OES_{n,i}$. If multiple CSSs has the same earliest OCB entry start time, the one with the smallest index will be selected.

The vPPS also calculates the $OES_{n,i}$ differently for different packet categories. Similarly, we use a fixed time T to represent the maximum delay from the input port to the CSS output queue. Denote the ***I**nput port **A**rrival time* of packet n as IA_n and its ***C**SS output queue **A**rrival time* as CA_n. Then we have $CA_n = IA_n + T$.

For the first K packets, each of them selects the CSS with an empty output queue and with the same index. For example, packet 1 will select CSS1. When these packets arrive at the CSS output queue, the OCB entry is available. However, they will not enter OCB until time X, where $X = CA_1 + \frac{L}{r}$. Thus, the first K packets have the same OCB entry start time, i.e.

$$OES_{1,1} = ... = OES_{K,K} = X = CA_1 + \frac{L}{r} \tag{3.20}$$

On the other hand, for packets after the first K ones, they may have to wait to enter the OCB until the last packet in the same CSS output queue finishes its OCB entry. Denote the last packet in CSS i as packet m and its ***O**CB **E**ntry **F**inish time* as $OEF_{m,i}$. The OCB entry start time of packet n is calculated as

$$OES_{n,i} = max(CA_n, OEF_{m,i}), \forall n > K \tag{3.21}$$

Then the demultiplexer selects the CSS which has the smallest OCB entry start time as the destination CSS for packet n.

Switching at CSSs

For simplicity, we assume all the CSSs are output queued switches. Therefore, after a packet arrives at the CSS, it will stay in the output queue until the OCB entry start time comes and then start to enter the OCB.

Packet Collection

The multiplexer collects packets one by one according to their arrival order at the input port. Recall that the first packet enters the OCB at time X. The multiplexer

will start to collect the first packet at time D, where $D = X + \frac{L}{r}$. Denote the ***O**CB **D**eparture **S**tart time* of packet n from the ith OCB as $ODS_{n,i}$. Therefore,

$$ODS_{1,i} = D = X + \frac{L}{r} \tag{3.22}$$

3.3.3 Performance Analysis

Theorem 6 *In the vPPS, the queue length B_i of any ICB is bounded by $2L$, i.e.*

$$B_i \leq 2L \tag{3.23}$$

Proof. Recall that in the first stage of the packet distribution policy, all $2K - 1$ CSSs will be divided in two groups, one with $K-1$ CSSs and another with K CSSs. Denote the former as Group 1 and the latter as Group 2. In the second stage, one of the CSS i in Group 2 will be chosen as the final CSS of the arrival packet. From the policy, we know that the ICB queue length of CSS i is less than or equal to the queue length of any other CSSs in Group 1. Then if we consider CSS i and all $K - 1$ CSSs in Group 1 together as a new Group 3, we find that the ICB queue length of CSS i is the shortest among the all K CSSs in Group 3. In other words, the CSS selection in vPPS can be considered as selecting the CSS with the shortest ICB length among K selected CSSs. Thus by Theorem 3, we can prove that the queue length of any CSS in Group 3 is bounded by $2L$. Since the CSS in Group 3 has the largest ICB queue length, the ICB queue length of CSSs in both Group 1 and Group 2 is bounded by $2L$. □

Lemma 6 *In the vPPS, packet n is already in the OCB when the multiplexer starts to collect it. In other words, packet n's OCB departure start time $ODS_{n,i}$ is greater than or equal to its OCB entry finish time $OEF_{n,i}$, i.e.*

$$ODS_{n,i} \geq OEF_{n,i} \tag{3.24}$$

Proof. Lemma 6 can be proved by using the similar method in Lemma 5. □

Theorem 7 *The vPPS can emulate an FIFO OQ switch with speedup of 2.*

Proof. Similar to the proof of Theorem 4, the FIFO OQ switch emulation can be proved by Lemma 6. The total bandwidth of the $2K-1$ CSSs is $(2-1/K)R$, which indicates speedup of 2. □

Theorem 8 *In the vPPS, the queue length C_i of any OCB i is bounded by $2L$, i.e.*

$$C_i \leq 2L \tag{3.25}$$

Proof. The queue length of any OCB keeps increasing until the multiplexer collects packet from it. In other words, C_i may reach its maximum only at the time when packet n departs from the OCB, i.e. at $ODS_{n,i}$.

It is observed that, by the time $ODS_{n,i}$, all the packets that have entered OCB i prior to packet n have already left. Thus at this moment, the OCB length $C_{n,i}$ should equal to the total amount of packets that entered OCB i during $OES_{n,i}$ to $ODS_{n,i}$. And this total amount is maximized if the ith OCB entry keeps busy during $OES_{n,i}$ to $ODS_{n,i}$, i.e.

$$C_{n,i} \leq (ODS_{n,i} - OES_{n,i})r \tag{3.26}$$

By (3.20), (3.21) and (3.22), we have

$$C_{n,i} \leq (ODS_{n,i} - CA_n)r = 2L$$

Hence, the queue length of any OCB i is bounded by $2L$. □

3.4 Summary

In this chapter, we study the PPS architecture which handles variable-length packets directly without SAR. We first describe a 1×1 variable-length PPS. Two scheduling policies are presented. With the first policy, it is proved that input conversion

buffers of size $2L$ are sufficient. With the second policy, it is proved that output conversion buffers are bounded by $2L$ as well. Next, we present the general $N \times N$ variable-length PPS. It extends the 1×1 PPS by expanding the switch structure and combining the two scheduling policies. It is showed that such a PPS can emulate an FIFO OQ switch, i.e. emulating an FIFO OQ switch with bandwidth R by $2K - 1$ center stage switches each with bandwidth r, where $r = R/K$. Further we prove that input and output conversion buffers of size $2L$ are sufficient.

CHAPTER 4

EFFICIENT VM PLACEMENT WITH MULTIPLE DETERMINISTIC AND STOCHASTIC RESOURCES IN DATA CENTERS

This chapter investigates the virtual machine (VM) placement challenge to maximize the resource utilizations of the DCN. Existing VM placement algorithms usually assume that VMs' demands for resources are deterministic and stable. However, for certain resources, such as network bandwidth, VMs' demands are bursty and time varying, and demonstrate stochastic nature. In this chapter, we propose a Max-Min Multidimensional Stochastic Bin Packing (M^3SBP) algorithm to find the effective placement results for VMs with multiple deterministic and stochastic resources.

We first formulate VM placement in DCN as Multidimensional Stochastic VM Placement (MSVP) problem, with the objective to minimize the number of required servers and at the same time satisfy a predefined resource availability guarantee. As the MSVP problem is proven NP-hard, we propose the polynomial time M^3SBP algorithm to quickly find solutions. The basic idea of is to maximize the minimum utilization ratio of all the resources of a server, while satisfying the demands of VMs for both deterministic and stochastic resources. We also conduct simulations to evaluate the performance of M^3SBP. The results demonstrate that M^3SBP guarantees the availability requirement for stochastic resources, and M^3SBP needs the smallest number of servers to provide the guarantee among the benchmark algorithms.

4.1 Introduction

VMs are attractive to modern data centers because they may significantly promote the efficiency and flexibility [BKB07] [HMGW08] [XF11]. However, such an incentive highly relies on a well-designed VM placement scheme [SMLF09] [VT09]. This is

because an inefficient placement scheme may result in lower resource utilization and thus needs more physical servers, which will further lead to not only higher capital investments on equipment and facilities, but also increased operational expenditures on energy and labor.

VM placement needs to consider VMs' demands for various resources. CPU and memory are traditionally the two major considerations. More recently, due to the increasing concerns on data center energy [AMW+10] [HSM+10] and emerging bandwidth intensive applications [BAM10] [KSG+09], VMs' power consumption [KZLK10] [KAGS11] and bandwidth requirement are also taken into account when computing the placement. It has started attracting attention in the research community to find optimal VM placement for VMs with multiple resource demands. One common assumption of the existing works [XF10] [MPZ10] is that, all resources are *deterministic* resources for which the demands are stable over time. The placement computing for this type of resource can be simply done by comparing the VM's resource demand with the server's available capacity. We refer VM placement algorithms handling only deterministic resources as deterministic algorithms.

However, recent studies [BAM10] [CZS+11] [KSG+09] [KRT00] indicate that VMs' demands for certain resources are highly bursty, and can be modeled as stochastic processes. In other words, the real demands of these *stochastic* resources are fluctuating, and it is difficult to obtain an accurate fixed-value measure. One such example is network bandwidth, and it is shown [KSG+09] [WMZ11] that bandwidth demands of VMs in data centers can be approximated by the normal distribution. As a result, besides supplying fixed-value deterministic resources requested by VMs, servers provide an availability guarantee for stochastic resources in the form of a violation probability threshold, specified in the service level agreements (SLAs). The threshold gives the worst-case likelihood that a server cannot satisfy the dy-

namical demands of a VM for stochastic resources. It may seem straightforward for deterministic algorithms to handle stochastic resources by estimating an equivalent fixed-value demand. However, it has been shown in [WMZ11] that this estimation is not accurate, since the equivalent demand for a stochastic resource of individual VMs vary under different placement schemes. Such a naive approach may result in either waste of server resources or violation of servers' availability guarantee.

4.2 Problem Formulation of Multidimensional Stochastic VM Placement

In this section, we describe the MSVP problem and present the problem formulation.

We consider a scenario in which there are n VMs with m kinds of resources to be placed into a number of servers. Among the m resources, there are both deterministic and stochastic resources. Taking into account the homogeneous architectures of modern data centers [AFLV08] [GHJ+09], the servers are assumed to have identical capacities. In order to host VM v_i, server s needs to meet all of its resource demands. For the deterministic resource, this can be simply done by assigning the same amount of resource as requested. However, the demand of stochastic resource is time-varying, and it is challenging to calculate an accurate resource allocation. Thus, for each stochastic resource, a violation probability threshold is defined to specify the maximum probability that the server's capacity of this resource is exceeded. In our scenario, we assume all stochastic resources share the same violation probability threshold α as in the SLA. Therefore, considering the multidimensional and stochastic characteristics of VM's demands, a placement scheme is considered valid only if it satisfies the following two conditions:

Condition 1) The capacities of the server's deterministic resources should not be exceeded by the total amount of demands of all hosted VMs;

Condition 2) The probability that the capacities of the server's stochastic resources are exceeded is no larger than the given violation probability threshold.

Thus, the objective of the MSVP problem becomes that to find a VM placement scheme, such that the above two conditions are satisfied and the number of required servers in the network is minimized.

Denote the demand of a deterministic resource p of VM v_i as $D_p(v_i)$ and server s's corresponding capacity as $C_p(s)$. *Condition 1* can be represented as follows,

$$\forall p \in P, \sum_{i \in U} D_p(v_i) \leq C_p(s) \tag{4.1}$$

where P is the set of all deterministic resources and U is the set of VMs hosted by s. Assume the demand of stochastic resource q of VM v_i independently follows a normal distribution $N(\mu_q(v_i), \sigma_q^2(v_i))$. An equivalent total demand of all VMs within the same server for each stochastic resource can be calculated based on each VM's distribution and the server's violation probability threshold α as follows,

$$\forall q \in Q, \sum_{i \in U} \mu_q(v_i) + \Phi^{-1}(1-\alpha)\sqrt{\sum_{i \in U} \sigma_q^2(v_i)} \tag{4.2}$$

where Q is the set of stochastic resources of v_i, U is the set of VMs already placed in the current server, and $\Phi^{-1}(1-\alpha)$ is the quantile of $N(0,1)$ at probability α. Thus, Condition 2 can be quantified by the equivalent total demand as follows,

Quantified Condition 2: The capacities of each server's stochastic resources should not be exceeded by the equivalent total demand of all hosting VMs.

Denote server s's capacity of stochastic resource q as $C_q(s)$. Then, *Condition 2* can be formulated as follows,

$$\forall q \in Q, \sum_{i \in U} \mu_q(v_i) + \Phi^{-1}(1-\alpha)\sqrt{\sum_{i \in U} \sigma_q^2(v_i)} \leq C_q(s) \tag{4.3}$$

We define the TCR ratio for each resource of a server to be the ratio between the total demand of all VMs within the same server and the server's capacity for this

resource. Denote $R_p(s)$ and $R_q(s)$ as the TCRs of deterministic resource p and stochastic resource q, respectively. Then we have,

$$\forall p \in P, R_p(s) = \frac{\sum_{i \in U} D_p(v_i)}{C_p(s)} \tag{4.4}$$

$$\forall q \in Q, R_q(s) = \frac{\sum_{i \in U} \mu_q(v_i) + \beta \sqrt{\sum_{i \in U} \sigma_q^2(v_i)}}{C_q(s)} \tag{4.5}$$

where $\beta = \Phi^{-1}(1 - \alpha)$.

Therefore, by combining (4.1), (4.3), (4.4) and (4.5) , the MSVP problem can be formulated as follows,

$$\begin{aligned} & \text{minimize} \quad |S| \\ \text{s.t.} \quad & \forall p \in P, \forall s \in S, R_p(s) \leq 1 \\ & \forall q \in Q, \forall s \in S, R_q(s) \leq 1 \end{aligned} \tag{4.6}$$

where S is the set of servers to host the VMs and $|S|$ is the size of S.

We can see that the classic NP-hard multidimensional bin packing problem is a special case of the MSVP problem, with the standard deviation of the demand of each stochastic resource set to 0. Thus, MSVP is also an NP-hard problem.

4.3 Max-Min Multidimensional Stochastic Bin Packing (M^3SBP) Algorithm

In this section, we present the Max-Min Multidimensional Stochastic Bin Packing (M^3SBP) algorithm that finds an approximation result for the MSVP problem. This algorithm is inspired by First Fit Decreasing (FFD) [CJGMV98], and Dominant Resource First (DRF) [GZH+11]. FFD solves the classical bin packing problem, by first sorting the items in the decreasing order of their sizes and then packing larger items with higher priorities. DRF tackles the fair resource allocation problem, where bins with multiple resources are shared by different users. The user's dominant share is defined as the maximum share that the user has been allocated of any resource.

DRF seeks to maximize the minimum dominant share across all users. Both FFD and DRF yield higher server utilizations and thus fewer servers than other naïve bin packing algorithms do.

The basic idea of M^{3}SBP is as follows. For each newly powered-on server (new server in short), M^{3}SBP finds a set of VMs which can maximize its minimum TCR, so that the minimum resource utilization of each server is maximized and hence the total number of servers needed to power on is minimized. M^{3}SBP runs in iterative rounds, and in each round one VM is selected and placed into the new server. If there still exist VMs without placement when the new server is full, another server will be powered on. The iteration repeats until all VMs find their placement.

4.3.1 Algorithm Description

Initially, all VMs are marked as unplaced and added into the unplaced-VM set V. M^{3}SBP employs the set V and the resource capacities of server s as inputs. The algorithm runs in iterations, and in each iteration, one VM will be placed into s and removed from V. M^{3}SBP ends when V is empty and then outputs the mapping between VMs and servers as the result. Specifically, each iteration of M^{3}SBP can be further divided into two steps: Candidate Finding and Placement. Pseudo-codes are shown in Algorithm 1.

Step 1 ***Candidate Finding****:* The goal of Step 1 is to find a set of candidate VMs that can be potentially placed in the new server s. For each VM v_i in V, M^{3}SBP calculates the TCRs of all resources of s as if v_i was placed in s. By (4.4) and (4.5), the TCRs of deterministic and stochastic resources can be computed by using following equations, repectively,

$$\forall p \in P, R_p(v_i, s) = \frac{D_p(v_i) + \sum_{j \in U} D_p(v_j)}{C_p(s)} \tag{4.7}$$

$$\forall q \in Q, R_q(v_i, s) = \frac{(\mu_q(v_i) + \sum_{j \in U} \mu_q(v_j)) + \beta\sqrt{\sigma_q^2(v_i) + \sum_{j \in U} \sigma_q^2(v_j)}}{C_q(s)} \tag{4.8}$$

The minimum and the maximum TCRs, $R_{min}(v_i, s)$ and $R_{max}(v_i, s)$, are then derived. If $R_{max}(v_i, s)$ is no greater than 1, it indicates that v_i can be placed in s. Then, the algorithm records $R_{min}(v_i, s)$ and adds v_i into set V' that stores the candidate VMs. If V' is empty after all VMs are tested, M^{3}SBP powers on another new server s and repeat Step 1. Otherwise, the algorithm continues with Step 2.

Step 2 ***Placement****:* In this step, M^{3}SBP compares the minimum TCR values of all candidate VMs in V' and chooses the VM v_F, which has the maximum value, as the selected VM. Then, M^{3}SBP places v_F into server s and adds it into set U. Finally, M^{3}SBP removes v_F from V.

4.3.2 Illustration Example

In this subsection, we give a simple example to illustrate the algorithm. Assume that each server has identical capacities of memory, CPU, power and network bandwidth, denoted by C_m, C_c, C_p and C_b, respectively. In this example, their values are set to be 8 GB, 4 GHz, 1000 W and 1 Gbps. We assume that there are 3 VMs, v_1, v_2 and v_3, to be placed into the servers. Each VM has deterministic demands on memory, CPU and power consumption resources, denoted by $D_m(v_i)$, $D_c(v_i)$ and $D_p(v_i)$, respectively, and has a stochastic demand on the network bandwidth resource, which follows a normal distribution $N(\mu_b(v_i), \sigma_b^2(v_i))$. Table 4.1 summaries all demands of the 3 VMs. The violation threshold α is set as 0.01%.

Initially, all VMs are added to unplaced-VM set V and the first server s_1 is powered on. In the first step, *Candidate Finding*, M^{3}SBP calculates the TCR for each of the 3 VMs as if the VM was already placed in the server. By (4.7) and (4.8), we have $R_m(v_1, s) = 0.625, R_c(v_1, s) = 0.25, R_p(v_1, s) = 0.225, R_b(v_1, s) = 0.208$.

Algorithm 1 Max-Min Multidimensional Stochastic Bin Packing (M^{3}SBP)

procedure M^{3}SBP(V, s)
  while $V \neq \emptyset$ **do**
   // Step 1: Candidate Finding
    for all $v_i \in V$ **do**
      CalculateTCR(v_i, s);
      if $R_{max}(v_i, s) \leq 1$ **then**
        Record $R_{min}(v_i, s)$;
        Add v_i into V';
      end if
    end for
    if $V' = \emptyset$ **then**
      Power on a new server s;
      Repeat *Step 1*;
    end if

   // Step 2: Placement
    $R(v_F, s) \leftarrow max\{\forall v_i \in V', R_{min}(v_i, s)\}$
    Add v_F into U;
    Remove v_F from V;
  end while
end procedure

procedure CalculateTCR(v_i, s)
  // Deterministic Resources
  $\forall p \in P, R_p(v_i, s) \leftarrow \frac{D_p(v_i)+\sum_{j\in U} D_p(v_j)}{C_p(s)}$;

  // Stochastic Resource
  $\forall q \in Q, R_q(v_i, s) \leftarrow$
    $\frac{(\mu_q(v_i)+\sum_{j\in U} \mu_q(v_j))+\beta\sqrt{\sigma_q^2(v_i)+\sum_{j\in U} \sigma_q^2(v_j)}}{C_q(s)}$;

  // Derive Max and Min TCRs
  $R_{min}(v_i, s) \leftarrow min\{\forall r \in \{P, Q\}, R_r(v_i, s)\}$;
  $R_{max}(v_i, s) \leftarrow max\{\forall r \in \{P, Q\}, R_r(v_i, s)\}$;
end procedure

	$D_m(v_i)$	$D_c(v_i)$	$N(\mu_b, \sigma_b^2)$	$D_p(v_i)$
v_1	5.0 GB	1.0 GHz	(200, 2^2) Mbps	225 W
v_2	2.0 GB	1.5 GHz	(500, 5^2) Mbps	300 W
v_3	1.0 GB	2.5 GHz	(300, 3^2) Mbps	150 W

Table 4.1: Summary of Resource Demands of 3 VMs

Then the maximum and minimum TCR of v_1 can be derived as $R_{max}(v_1, s) = 0.625$ and $R_{min}(v_1, s) = 0.208$. Since $R_{max}(v_1, s) \leq 1$, the server has enough resource for v_1. Thus v_1 is added to the candidate set V' and $R_{min}(v_1, s)$ is recorded. Repeat the same calculation for v_2 and v_3, we have $R_{max}(v_2, s) = 0.519, R_{min}(v_2, s) = 0.25$ and $R_{max}(v_3, s) = 0.625, R_{min}(v_3, s) = 0.125$. Thus both v_2 and v_3 are added to V' and their R_{min} are recorded. Then, in the second step *Placement*, M^{3}SBP compares the $R_{min}(v_i, s)$ of VMs in V' and finds v_2 with the maximum value. As a result, M^{3}SBP places v_2 into s_1 by adding v_2 into s_1's set of hosting VMs, U. Repeat the above two steps for v_1 and v_3. The final result is that, v_1 and v_2 are placed in s_1, while v_3 is placed in s_2.

4.3.3 Complexity Analysis

The complexity of the M^{3}SBP algorithm can be calculated in two phases. First, we count the number of execution times of **while** loop. Denote the size of V during **while** loop's kth execution as l_k. Initially, V contains all VMs. Thus $l_0 = n$, where n is the total amount of VMs. During each iteration of the **while** loop, exact one VM finds its placement and the size of V decreases by one. Thus, after n times of execution, V will be empty, i.e. $l_n = 0$. Therefore, the execution time of the **while** loop is the same as the number of VMs, n. In addition, l_k can be calculated by using the following equation, $l_k = n - k$.

Second, we calculate the complexity inside the **while** loop. In *Candidate Finding* phase, it takes $O(\log m)$ time to find the minimum and maximum TCRs for each

VM in V, where m is the total number of resources. Since there are l_k unplaced VMs in the kth iteration, it needs a total of $O(l_k \times \log m)$ time to finish the candidate searching. In the *Placement* phase, it takes $O(\log l'_k)$ time to find the maximum of the minimum TCRs, where l'_k denotes the size of candidate VM set V' of the kth while loop iteration. In the worst-case scenario, l'_k can be as large as l_k. Then the time complexity of the *Placement* phase becomes $O(\log l_k)$. Thus, the time complexity inside the **while** loop is

$$O(l_k \times \log m) + O(\log l_k) \tag{4.9}$$

It is showed that $l = n - k$, and thus (4.9) becomes

$$O((n-k) \times \log m) + O(\log(n-k)) \tag{4.10}$$

Since $O(n-k) > O(\log(n-k))$ and typically $n >> m$, (4.10) can be simplified to $O(n-k)$. By combining the complexity of inside and outside of the **while** loop together, the total time complexity of M^{3}SBP is $O(n(n-k)) = O(n^2)$.

4.4 Performance Evaluation

In this section, we present the performance evaluation configuration and results analysis of the M^{3}SBP algorithm. We have conducted multiple simulations to evaluate different aspects of the performance, including the number of used servers, the guarantee of violation probability threshold and the algorithm effectiveness. Detail simulation configurations are described in Section 4.4.1. Results and analysis are shown in Section 4.4.2, Section 4.4.3 and Section 4.4.4.

4.4.1 Simulation Configuration

In the simulations, each VM is assumed to have four types of resource demands: CPU, memory, power consumption and bandwidth. The first three are deterministic, while the bandwidth demand is stochastic and follows the normal distribution.

We employ the resource-**D**emand to server-**C**apacity **R**atio (DCR) to identify the demand intensity of each resource. For each VM, we define the resource, which has the largest DCR value among all resources, as the *intensive resource*, and define others as the *non-intensive resources*. A data center may have VMs with different resource intensities, such as memory-intensive and CPU-intensive VMs. This kind of data centers demands the most resources from the multidimensional placement algorithm. It is because that if all VMs have the same intensive resource, the placement problem is then reduced to the classical one-dimensional VM placement problem.

To simulate the mixed-intensity situation, we configure 4 groups of VMs each with a different intensive resource, and each group contains 1/4 of all VMs. For each VM, the intensive resource randomly selects its DCR value from a higher range between 30% to 40%, and other non-intensive resources randomly select their DCR values from a lower range between 5% to 10%. For the stochastic bandwidth demand, the selected DCR value represents the ratio between the mean of the bandwidth demand and the server capacity. The bandwidth's standard deviation is set to be 0.5% of the mean demand by default. This percentage may change later in different simulations. Servers are assumed to have the same capacity of, 24 GHz (8 cores $\times$ 3.0 GHz/core) of CPU, 48 GB of memory, 2000 W of power supply and 1 Gbps of bandwidth. Then we can calculate the demands by multiplying the DCRs with the server's capacities. The total number of VMs is set to be 2000 and the SLA violation probability is set to be 0.01%.

The following example illustrates how resource demands of a *Memory-intensive* VM are configured. The memory resource's DCR is randomly selected between 30% to 40%, assuming 34%. CPU's and power consumption's DCR are randomly selected between 5% to 10%, assuming 6% and 7.5%, respectively. The DCR of

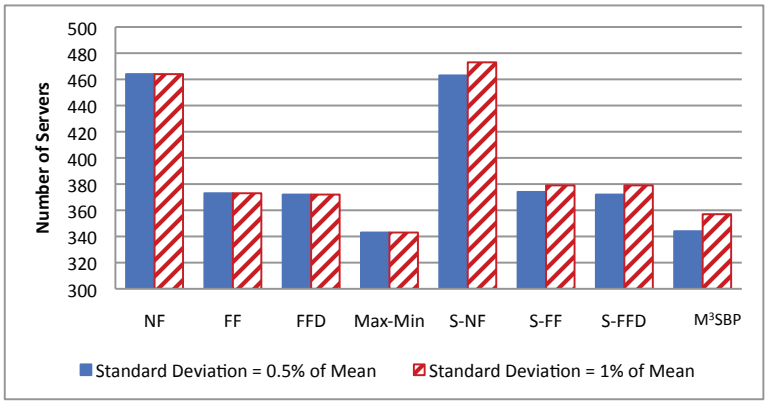


Figure 4.1: Number of servers used by different placement algorithms

bandwidth mean is also randomly selected from the lower range, assuming 5.5%. The standard deviation is set be 0.5% of the mean demand which in this case is 5.5% × 0.5% = 0.0275% of server's bandwidth capacity. Then, from server's capacities, we can derive VM's demands as 48 GB × 34% = 16.32 GB of memory, 24 GHz × 6% = 1.44 GHz of CPU, 2000 W × 7.5% = 150 W of power and 1 Gps × 5.5% = 55 Mbps mean with 0.275 Mbps standard deviation of bandwidth.

4.4.2 Number of Servers

In this subsection, we present the simulation results on the number of used servers. We compare M^{3}SBP with other bin packing algorithms, including both deterministic and stochastic algorithms. Deterministic algorithms, which compute placement only with fixed value demands, includes Next-Fit (NF) [CJGMV98], First-Fit (FF) [CJGMV98], FFD and Max-Min. NF places the VM into the current server if the demands of all resources are satisfied, or otherwise starts a new server. FF looks at all existing bins and places the VM into the lowest numbered server if it fits, or otherwise powers on a new server. FFD employs the same packing strategy as that of FF, except that, before the placement, FFD sums the DCRs of all resources of each VM and sorts the VMs in the decreasing order of their DCR summations. Max-Min

has identical procedures as those of M^{3}SBP except that it treats the bandwidth as a deterministic resource and employ the mean bandwidth demand in the placement computing. Same as Max-Min, all other deterministic algorithms view bandwidth as a deterministic resource, and also employ the mean bandwidth demand when computing the placement.

Comparison stochastic algorithms include stochastic NF (S-NF), stochastic FF (S-FF) and stochastic FFD (S-FFD). These algorithms are modified based on the classic NF, FF and FFD algorithms, respectively. The modified algorithms calculate the equivalent TCR for stochastic resources by using the same equation (4.8) as in M^{3}SBP. We have conducted two sets of simulations. Both of them follow the same default configurations described in Section 4.4.1, except that each simulation uses a different standard deviation to mean ratio. The standard deviation represents the burst level of the VM's traffic. In the first set, the standard deviation is 0.5% of the mean, while in the second set the ratio is 1%.

Figure 4.1 illustrates the results. The solid and strip columns show results when the standard deviation is equal to 0.5% and 1% of the mean, respectively. Comparing the number of servers used by M^{3}SBP with that of all other stochastic algorithms, we can see that M^{3}SBP uses the fewest servers in both simulations. Then if we compare M^{3}SBP with the deterministic algorithms, we can find that M^{3}SBP still uses almost the least number of servers when VM's traffic burst level is low. When VM's traffic is more bursty, the number of servers used by all stochastic algorithms increase. This is reasonable since stochastic algorithms take the increasing burst into consideration, and allocate more server resources for VMs to prevent their network traffic from exceeding server's capacity. On the other hand, however, deterministic algorithms cannot detect the change of VM's burst level. This brings negative influence on server availability guarantees which is discussed in detail in Section 4.4.3.

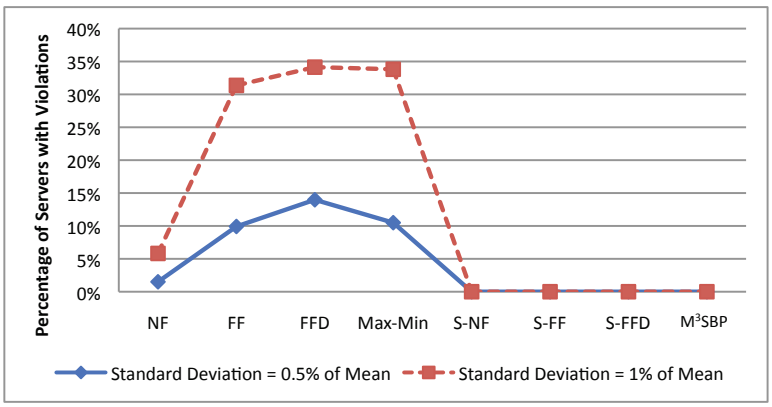


Figure 4.2: Percentage of servers violating the target violation probability threshold

4.4.3 Server Availability Guarantee

In this subsection, we evaluate how well the M^{3}SBP algorithm guarantees the availability requirement when VMs demand for stochastic resources. Stochastic algorithms compute placement based on the target violation probability threshold. After the VM placement, we verify whether the availability requirement is guaranteed by comparing the real traffic with the server's bandwidth capacity. In each simulated second, we carry out the following procedures. First, we generate traffic for all VMs according to their stochastic parameters. Then, we calculate the total traffic amount for each server by adding up the traffic of all its hosted VMs. Lastly, we compare the total traffic amount with server's bandwidth capacity. If the former is larger, we say that in this second the bandwidth capacity is exceeded and the server's availability requirement is violated, and count the second as a violated second. At the end, the real violation ratio is calculated by dividing the total number of simulated seconds by the total number of violated seconds. Then, if the real violation ratio is higher than the target violation probability threshold, we say that this server failed to guarantee the availability requirement. The target violation probability threshold is set to 0.01% and the simulation emulates 7 days of network traffic.

Following these procedures, we evaluate the placement results of the previous simulations. Figure 4.2 shows the percentage of servers which have violated the target violation probability threshold. We can see that all deterministic algorithms have violated servers. Moreover, when the traffic is more bursty, the number of violated servers of deterministic algorithms rises up dramatically. Both FF, FFD and Max-Min have more than 30% of servers violates the target violation probability threshold when standard deviation is equal to 1% of the mean. In contrast, none of the stochastic algorithms, including M^{3}SBP, has violated servers. This demonstrates that M^{3}SBP can guarantee the server's availability well.

4.4.4 Effectiveness

In this subsection, we evaluate the effectiveness of M^{3}SBP algorithm. Effectiveness of a placement algorithm is defined as follows: 1) if two algorithms consume the same number of servers, the more effective one possesses less percentage of server violations; or 2) if two algorithms all have zero server violation, the more effective one uses a smaller number of servers.

By applying these definitions to the results in Figure 4.1 and 4.2, we find that M^{3}SBP is more effective than all other stochastic algorithms. This is because it is compliant with the second part of the effectiveness definition that M^{3}SBP requires the fewest number of servers while obtaining zero server violation.

However, it is not straightforward to compare the effectiveness between M^{3}SBP and deterministic algorithms, which possess a higher violation ratio but need fewer servers. One solution is to gradually enlarge the demand of stochastic resources used by deterministic algorithms in placement computing, so that the deterministic algorithms will use more servers and generate fewer violations.

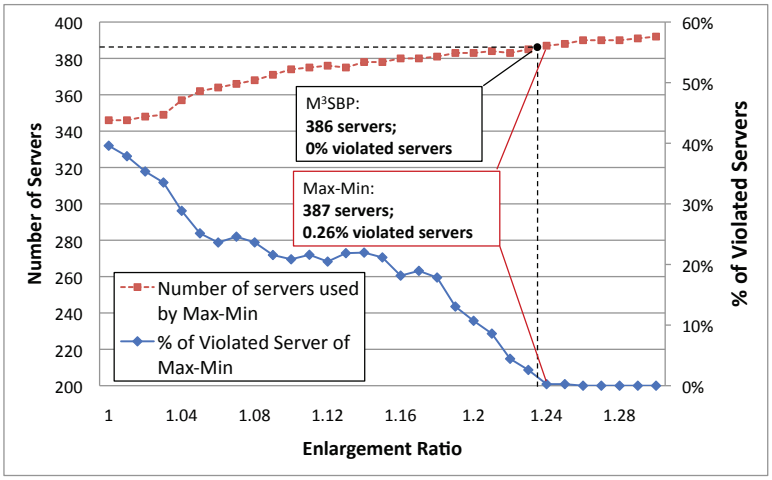


Figure 4.3: Number of servers and percentage of violated servers of Max-Min when gradually increasing the enlargement ratio from 1.00 to 1.30.

In our simulation, we gradually increase the demand of bandwidth resource in the deterministic Max-Min algorithm from the original value to 1.30 times of it. From Figure 4.3, we can see that, as expected, when the bandwidth demand increases, the number of used servers of Max-Min also increases and the percentage of violated server decreases. When the bandwidth demand is enlarged to 1.24 times of the original value, Max-Min uses more servers than M^{3}SBP does. However, there are still 0.26% of servers in Max-Min violates the target violation probability threshold. Thus, M^{3}SBP finds better placement in shorter amount of time than Max-Min. In other words, M^{3}SBP is more effective than Max-Min. Due to space limitations, the results of comparisons between M^{3}SBP and other deterministic algorithms are omitted. All comparisons lead to the same conclusion. Therefore, we can say that M^{3}SBP is more effective than all benchmark algorithms.

4.5 Summary

In this chapter, we have studied VM placement in data centers when the VMs have multiple demands for various resources and some of them are stochastic resources.

We formulate the Multidimensional Stochastic VM Placement (MSVP) problem. Because MSVP is NP-hard, we propose a fast algorithm named Max-Min Multidimensional Stochastic Bin Packing (M^3SBP), to calculate the VM placement for large scale data centers. Numerical simulations are conducted to evaluate M^3SBP's performance. In the simulations, each VM requests four types of resources, CPU, memory, power consumption and bandwidth. Among those resources, the first three are deterministic while bandwidth is stochastic, and we employ the normal distribution to model the bandwidth demand. The results show that M^3SBP uses fewer servers than other benchmark algorithms, while guaranteeing the server's availability requirement. In addition, the results also demonstrate that M^3SBP is more effective in finding the desired placement result than other benchmark algorithms.

CHAPTER 5

OPENFLOW BASED FLOW-LEVEL BANDWIDTH PROVISIONING FOR CICQ SWITCHES

This chapter investigates the flow level bandwidth provisioning challenge to effectively isolate traffic of different VMs hosted by the same server. Existing flow-level bandwidth provisioning solutions suffer from a number of drawbacks, including high implementation complexity, poor performance guarantees, and inefficiency to process variable length packets. In this chapter, we propose the Flow-level Bandwidth Provisioning (FBP) algorithm for Combined Input Crosspoint Queued (CICQ) switches, which reduces the switch scheduling problem to multiple instances of fair queuing problems and provides guaranteed bandwidth to each flow in the DCN.

FBP can closely emulate the ideal Generalized Processing Sharing model, and accurately guarantee the provisioned bandwidth. We also implement FBP in the OpenFlow software switch to obtain realistic performance data by a prototype. Leveraging the capability of OpenFlow to define and manipulate flows, we experimentally demonstrate a practical flow-level bandwidth provisioning solution. In addition, we conduct extensive simulations and experiments to evaluate the design. The simulation data verify the correctness of the analytical results, and show that FBP achieves tight performance guarantees. The experiment results demonstrate that our OpenFlow based prototype can conveniently and accurately provision bandwidth at the flow level.

5.1 Introduction

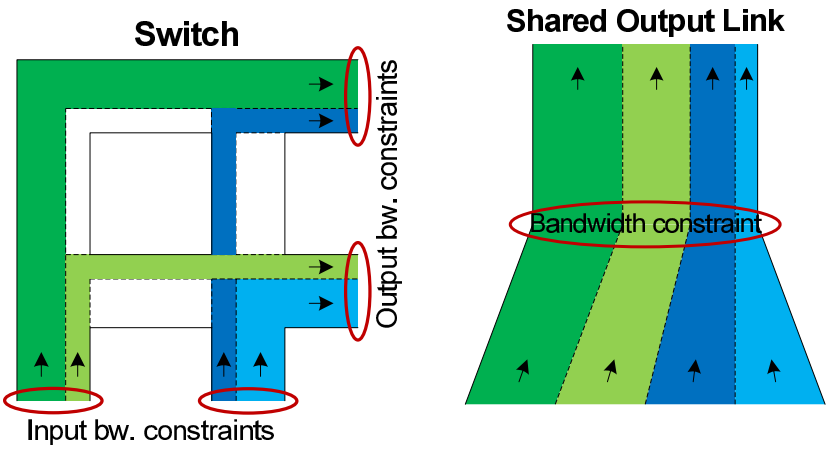
Bandwidth provisioning at the flow level is necessary, as it differentiates traffic at sufficiently fine granularity [GKP+08]. It is particularly important for virtualization based data centers. In such an environment, VMs reside in a single physical server,

and their traffic shares the same physical network adapter and is correspondingly fed into the same switch port. Flow-level bandwidth provisioning is able to isolate traffic of different VMs and make the shared underlying network infrastructure transparent to the VMs. The recently proposed Virtual Ethernet Port Aggregator (VEPA) protocol [VEP] off-loads all switching activities from hypervisor-based virtual switches to actual physical switches. As can be seen, VEPA requires flow-level bandwidth provisioning on switches to support traffic isolation between VMs.

Existing algorithms [CIM05] [CGMP99] [PY08] achieve flow-level bandwidth provisioning by emulating PIFO OQ switches, but they suffer from a number of drawbacks. First, they have high hardware complexity and time complexity. Specifically, they require a crossbar with speedup of at least two, i.e. the crossbar having twice bandwidth as that of the input port or output port, and they may need large expensive on-chip memories for the crossbar. In addition, they run in a centralized mode with up to N iterations for an $N \times N$ switch, or in other words the scheduling time increases proportionally with the switch size. Second, they cannot achieve constant service guarantees. Constant service guarantees mean that for any flow, the difference between its service amount in a specific algorithm and in the ideal Generalized Processing Sharing (GPS) [PG93] model is bounded by constants, i.e. the equations in Theorem 1 of [BZ96]. The reason is that Worst-case Fair Weighted Fair Queueing (WF^2Q) (including its variants) [BZ96], the only known fair queuing algorithm to achieve constant service guarantees, does not use a PIFO queuing policy [IM03], and hence the PIFO OQ switch emulation approach does not work. Third, although there have been switch designs [SPK08] in the literature to directly handle variable length packets, the existing flow-level bandwidth provisioning algorithms can only handle fixed length cells. The SAR process may waste bandwidth due to padding bits [Tur09].

In this chapter, we study the flow-level bandwidth provisioning problem in the OpenFlow and CICQ context. Our objective is twofold: to design an efficient flow-level bandwidth provisioning algorithm with constant service guarantees, and to experimentally demonstrate a practical flow-level bandwidth provisioning solution based on the OpenFlow protocol. CICQ switches are special crossbar switches with an on-chip buffer at each crosspoint, which is made available by recent development in VLSI technology [Kor06] [PKC07], and thus they are also called buffered crossbar switches. We consider CICQ switches because the crosspoint buffers decouple input ports and output ports, and greatly simplify the scheduling process.

OpenFlow [MSA+06] is an open protocol that gives access to the forwarding plane of switches and routers, so that users can control their traffic in the network. It has been deployed in large-scale testbeds like GENI [opea], and considered in many recent data center designs [MPF+09] [AfRR+10]. OpenFlow provides a rich set of options to define flows based on a combination of packet header fields, and use a flow table to allow users to flexibly control their traffic. Bandwidth provisioning has been recognized as an essential component of OpenFlow, to isolate traffic of different users or different types [sli]. Bandwidth provisioning is also recognized as an essential component for OpenFlow [sli]. The current OpenFlow implementation supports a Hierarchical Token Bucket (HTB) [htb] based framework called slicing, which is necessary but not sufficient to provide tight performance guarantees [BZ96]. As stated in [sli], slicing is a minimum but not complete QoS scheme. Slicing utilizes the HTB technique, which is a combination of token bucket traffic shaping and deficit round robin fair queuing [SV96]. HTB assures only minimal bandwidth, and cannot accurately guarantee the provisioned bandwidth. In addition, OpenFlow has a special controller called FlowVisor [flo], which creates slices of network resources, such as network bandwidth, and provides traffic isolation between different slides.



(a) For Switches (b) For Shared Output Links

Figure 5.1: GPS as Ideal Fairness Model

In our OpenFlow based bandwidth provisioning solution, there will be a central controller and a number of switches. On the one hand, the controller collects resource and request information from the switches, allocates bandwidth for flows, and updates the flow tables of switches to enforce the provisioned bandwidth. On the other hand, the switches receive flow definition and bandwidth allocation information from the controller, and run the proposed switch scheduling algorithm to guarantee the allocated bandwidth. The focus of this chapter is for the switches to accurately guarantee the allocated bandwidth of each flow by emulating the ideal GPS model. In GPS, each flow has a virtual dedicated channel with the allocated bandwidth, as shown in Figure 5.1(a). Thus, there is no interference between different flows, and each flow always receives the exact amount of its allocated bandwidth. Our goal is to bound the difference between the service amount of any flow in our algorithm and in GPS by constants, or in other words to achieve constant service guarantees. A more detailed problem formulation will be presented in Section 5.2.1.

5.2 Flow-Level Bandwidth Provisioning for CICQ Switches

In this section, we formulate the flow-level bandwidth provisioning problem, and present the FBP algorithm for CICQ switches.

5.2.1 Problem Formulation

The considered CICQ switch structure is shown in Figure 5.2. The switch has N input ports and N output ports. Denote the i^{th} input port as In_i and the j^{th} output port as Out_j. The input ports and output ports are connected by a buffered crossbar without speedup. In other words, each input port or output port has bandwidth of R, and so does the crossbar. For flow-level bandwidth provisioning, it is necessary for input ports to separate the traffic of different flows, i.e. storing incoming packets on a per flow basis. Denote the k^{th} flow from In_i to Out_j as F_{ijk}, and the queue at In_i to store its packets as Q_{ijk}. Besides a queue for each flow, In_i has a virtual output buffer for each Out_j, denoted as B_{ij}, to store the next packet departing from In_i to Out_j. Note that B_{ij} is not a physical buffer, but a pointer pointing to the head packet of one of the queues from In_i to Out_j. Each crosspoint of the crossbar has a small buffer. Denote the crosspoint buffer connecting In_i and Out_j as X_{ij}. There are no buffers at output ports.

Our objective is to accurately provision bandwidth for each flow by emulating the ideal GPS model. GPS views flows as fluids of continuous bits, and creates a virtual dedicated channel for each flow based on its allocated bandwidth, as shown in Figure 5.1(a). Because GPS is a fluid based system, a flow can smoothly stream from the input port to the output port without buffering in the middle. We thus assume that packets in GPS will skip the virtual output buffers and crosspoint buffers. GPS is also the ideal packet scheduling model of fair queuing algorithms for shared output links, as shown in Figure 5.1(b).

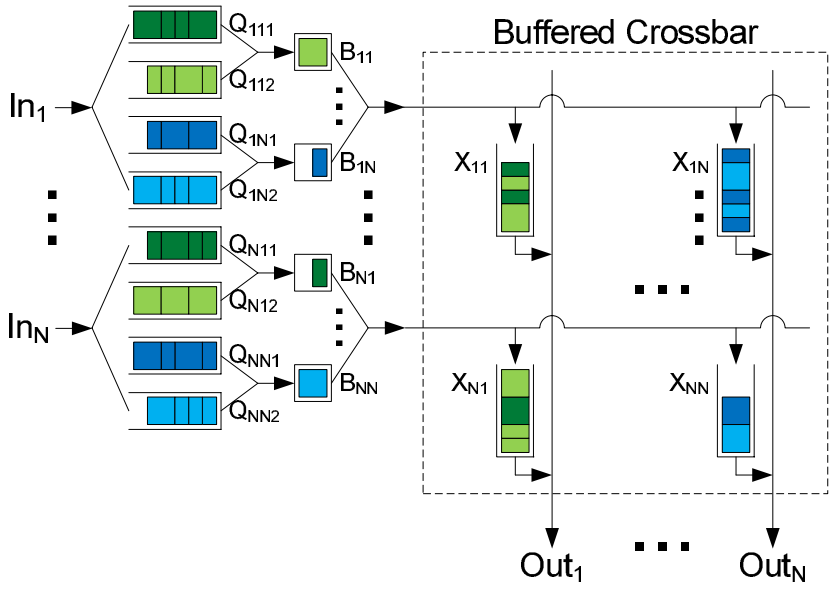


Figure 5.2: Structure of CICQ switches

Assume that a flow F_{ijk} has been allocated a certain amount of bandwidth R_{ijk}. Use $toO_{ijk}(0,t)$ and $\widehat{toO}_{ijk}(0,t)$ to represent the numbers of bits transmitted by F_{ijk} to the output port during interval $[0,t]$ in our algorithm and GPS, respectively. Formally, the objective is to bound the difference $|toO_{ijk}(0,t) - \widehat{toO}_{ijk}(0,t)|$ by constants, independent of R_{ijk} and t. Note that for feasible bandwidth allocation, no input or output should have over-subscription, i.e. $\forall i, \sum_{j,k} R_{ijk} \leq R,$ and $\forall j, \sum_{i,k} R_{ijk} \leq R$. The feasibility requirement is only for bandwidth allocation. Temporary overload is allowed for any input port and output port, with overloading packets being temporarily stored in input buffers.

5.2.2 Algorithm Description

The basic idea of the FBP algorithm is to reduce the switch scheduling problem to three stages of fair queuing, which we call flow scheduling, input scheduling, and output scheduling, respectively. Flow scheduling selects a packet from one of the flow queues Q_{ijk} from In_i to Out_j, and sends it to the virtual output buffer B_{ij}.

Input scheduling selects a packet from one of the N virtual output buffers B_{ij} of In_i, and sends it to the corresponding crosspoint buffer X_{ij}. Output scheduling selects a packet from one of the N crosspoint buffers X_{ij} of Out_j, and sends it to the output port. The detailed description of each scheduling stage is as follows.

Flow Scheduling

Flow scheduling utilizes the WF^{2}Q [BZ96] fair queuing algorithm to multiplex different flows of the same input-output pair as a single logical flow, to simplify input scheduling. For easy description, denote the n^{th} packet of F_{ijk} as P^n_{ijk}. Flow scheduling calculates two time stamps for each packet p: virtual flow start time $\widehat{FS}(p)$ and finish time $\widehat{FF}(p)$. They are the departure time of the first bit and last bit of p in GPS, and are calculated as $\widehat{FS}(P^n_{ijk}) = \max(A(P^n_{ijk}), \widehat{FF}(P^{n-1}_{ijk}))$ and $\widehat{FF}(P^n_{ijk}) = \widehat{FS}(P^n_{ijk}) + L(P^n_{ijk})/R_{ijk}$, where $A(p)$ is the arrival time of p, and $L(p)$ is the packet length.

The first step of flow scheduling identifies eligible packets. A packet is eligible for flow scheduling if it has started transmission in GPS. Specifically, a packet p is eligible at time t if its virtual flow start time is less than or equal to t, i.e. $\widehat{FS}(p) \leq t$. The second step selects among eligible packets the one p with the smallest virtual flow finish time, i.e. $\forall p', \widehat{FS}(p') \leq t \rightarrow \widehat{FF}(p') \geq \widehat{FF}(p)$. The selected packet will be sent to the corresponding virtual output buffer B_{ij}, to participate in input scheduling. If there are no eligible packets, flow scheduling will wait until the next earliest virtual flow start time. Additionally, we define two time stamps for p: actual flow start time $FS(p)$ and finish time $FF(p)$, to represent the actual departure time of its first bit and last bit from Q_{ijk} in flow scheduling. Flow scheduling multiplexes all flows from In_i to Out_j as a logical flow F_{ij}, which has bandwidth $R_{ij} = \sum_k R_{ijk}$. Thus, the last bit of p will leave Q_{ijk} at $FF(p) = FS(p) + L(p)/R_{ij}$.

Note that flow scheduling is only a logical operation of the input port buffer to determine the sequence of packets to participate in input scheduling. There is no actual packet transmission for flow scheduling, because the packet is in the input buffer both before and after flow scheduling.

Input Scheduling

Input scheduling uses WF^{2}Q to multiplex the logical flows F_{ij} of the same input In_i to share the bandwidth to the crosspoint buffers. Input scheduling also calculates two time stamps for each packet p: virtual input start time $\widehat{IS}(p)$ and finish time $\widehat{IF}(p)$, which are equal to the actual flow start and finish time, respectively, i.e. $\widehat{IS}(p) = FS(p)$ and $\widehat{IF}(p) = FF(p)$. Similar as flow scheduling, the first step of input scheduling identifies eligible packets whose virtual input start time is no later than the current scheduling time. The second step finds among eligible packets the one with the smallest virtual input finish time. The selected packet is then sent from the virtual output buffer to the crosspoint buffer. Additionally, we define the actual input start time $IS(p)$ and finish time $IF(p)$ to represent the time that the first bit and last bit of p leave B_{ij} in input scheduling, respectively. We have $IF(p) = IS(p) + L(p)/R$, since the bandwidth of the crossbar is R.

Output Scheduling

Output scheduling utilizes the WFQ [PG93] fair queuing algorithm to allow the crosspoint buffers of the same output to share the bandwidth to the output link. We can use WFQ instead of WF^{2}Q for output scheduling because input scheduling has restricted admission of packets into the crosspoint buffers. Output scheduling uses only one time stamp for a packet p: virtual output finish time $\widehat{OF}(p)$, which can be calculated as $\widehat{OF}(p) = \widehat{IF}(p) + L_m/R + L_m/R_{ij}$, where L_m is the maximum packet length. Output scheduling simply retrieves the packet with the smallest

virtual output finish time from the crosspoint buffers of an output and send it to the output link. Additionally, define the actual output start time $OS(p)$ and finish time $OF(p)$ to represent the actual departure time of the first bit and last bit of p from X_{ij}. Since the bandwidth of the crossbar is R, we have $OF(p) = OS(p) + L(p)/R$.

5.3 Performance Analysis

We now analyze the performance of FBP, and will show that it achieves constant service guarantees, tight delay guarantees, and bounded crosspoint buffer sizes. Since the three scheduling stages of FBP use the well studied WF^{2}Q [BZ96] and WFQ [PG93] fair queuing algorithms, our analysis will leverage the existing results for them. Both WF^{2}Q and WFQ schedule packets of multiple flows to emulate the ideal GPS model, and share some features in common. As indicated in [BZ96] and [PG93], there is an important property between the virtual departure time $\widehat{\mathcal{F}}(p)$ of a packet p in the virtual dedicated channel and the actual departure time $\mathcal{F}(p)$ in the physical multiplexed channel with bandwidth $\mathcal{R}$: $\mathcal{F}(p) \leq \widehat{\mathcal{F}}(p) + L_m/\mathcal{R}$.

Recall that flow scheduling uses WF^{2}Q to multiplex all the flows from In_i to Out_j, which share bandwidth of R_{ij}, as a logical flow. By the above property, we have $FF(p) \leq \widehat{FF}(p) + L_m/R_{ij}$. Input scheduling uses WF^{2}Q to multiplex the logical flows from In_i to different Out_j as an aggregate flow. For input scheduling, we can view the virtual input finish time $\widehat{IF}(p)(= FF(p))$ as the departure time of p in the virtual dedicated channel. Since the physical multiplexed channel for input scheduling, i.e., the channel from the input buffer to the crossbar, has bandwidth of R, we can obtain $IF(p) \leq \widehat{IF}(p) + L_m/R$.

Output scheduling uses WFQ to multiplex flows from different crosspoint buffers, and we show below that $\widehat{OF}(p)$ is the departure time of p in the virtual dedicated channel for packets from X_{ij} to Out_j and the its bandwidth is denoted as R_{ij} .

For easy representation, denote the n^{th} packet from In_i to Out_j as P_{ij}^n, and define $\widehat{OS}(P_{ij}^n) = \widehat{OF}(P_{ij}^n) - L(P_{ij}^n)/R_{ij}$.

Lemma 7 *By $\widehat{OS}(P_{ij}^n)$, P_{ij}^{n-1} has left X_{ij} and P_{ij}^n has arrived at X_{ij} in the virtual dedicated channel.*

Proof. First, it is easy to see that $\widehat{OS}(p) \geq \widehat{IF}(p) + L_m/R \geq IF(p)$, which means that p has arrived at the crosspoint buffer by $\widehat{OS}(p)$, and thus can start transmission in the virtual dedicated channel. Second, by the definition

$$\begin{aligned}\widehat{OS}(P_{ij}^{n+1}) &= \widehat{IF}(P_{ij}^{n+1}) + \frac{L_m}{R} + \frac{L_m}{R_{ij}} - \frac{L(P_{ij}^{n+1})}{R_{ij}} \\ &\geq \widehat{IF}(P_{ij}^n) + \frac{L_m}{R} + \frac{L_m}{R_{ij}} \\ &= \widehat{OF}(P_{ij}^n) \end{aligned} \tag{5.1}$$

we know that P_{ij}^{n-1} has left X_{ij} in the virtual dedicated channel by $\widehat{OS}(P_{ij}^n)$, and thus P_{ij}^n can start transmission without conflict. □

According to Lemma 7, we can safely view $\widehat{OS}(p)$ as the departure time of the first bit of p in the virtual dedicated channel, and thus $\widehat{OF}(p)$ is the departure time of the last bit of p in the virtual dedicated channel. Therefore, we have

$$OF(p) \leq \widehat{OF}(p) + \frac{L_m}{R} \tag{5.2}$$

5.3.1 Service Guarantees

We now show that FBP achieves accurately provisioned bandwidth, in the sense that the difference between the service amount of any flow in FBP and GPS at any time is bounded by constants.

Define $toO_{ijk}(t_1, t_2)$, $toX_{ijk}(t_1, t_2)$, and $toB_{ijk}(t_1, t_2)$ to denote the numbers of bits transmitted by F_{ijk} during interval $[t_1, t_2]$ to Out_j, X_{ij}, and B_{ij} in FBP, respectively. Correspondingly, use $\widehat{toB}_{ijk}(t_1, t_2)$ to represent the number of bits transmitted

by F_{ijk} to B_{ij} during $[t_1, t_2]$ in GPS. With the virtual dedicated channel of F_{ijk}, $\widehat{toB}_{ijk}(t_1, t_2)$ is also the number of bits sent by F_{ijk} to Out_j during $[t_1, t_2]$ in GPS.

Lemma 8 *When a packet P_{ijk}^n starts transmission to its destination output port in FBP, the number of bits transmitted to the output port by its flow F_{ijk} in FBP is greater than or equal to that in GPS minus $4L_m$, i.e. $toO_{ijk}(0, OS(P_{ijk}^n)) \geq \widehat{toB}_{ijk}(0, OS(P_{ijk}^n)) - 4L_m$.*

Proof. By the definition of $OS(P_{ijk}^n)$, P_{ijk}^{n-1} has finished output scheduling at $OS(P_{ijk}^n)$ in FBP, i.e.

$$toO_{ijk}(0, OS(P_{ijk}^n)) = \sum_{a=1}^{n-1} L(P_{ijk}^a) \tag{5.3}$$

On the other hand

$$\begin{aligned}
&\widehat{toB}_{ijk}(0, OS(P_{ijk}^n)) \\
&= \widehat{toB}_{ijk}(0, OF(P_{ijk}^n) - \frac{L(P_{ijk}^n)}{R}) \\
&\leq \widehat{toB}_{ijk}(0, \widehat{OF}(P_{ijk}^n) + \frac{L_m}{R} - \frac{L(P_{ijk}^n)}{R}) \\
&= \widehat{toB}_{ijk}(0, \widehat{IF}(P_{ijk}^n) + \frac{2L_m}{R} + \frac{L_m}{R_{ij}} - \frac{L(P_{ijk}^n)}{R}) \\
&\leq \widehat{toB}_{ijk}(0, \widehat{FF}(P_{ijk}^n)) + R_{ijk}(\frac{2L_m}{R} + \frac{2L_m}{R_{ij}} - \frac{L(P_{ijk}^n)}{R}) \\
&= \sum_{a=1}^{n} L(P_{ijk}^a) + R_{ijk}(\frac{2L_m}{R} + \frac{2L_m}{R_{ij}} - \frac{L(P_{ijk}^n)}{R})
\end{aligned} \tag{5.4}$$

By (5.3) and (5.4), we have

$$\begin{aligned}
&\widehat{toB}_{ijk}(0, OS(P_{ijk}^n)) - toO_{ijk}(0, OS(P_{ijk}^n)) \\
&\leq L(P_{ijk}^n) + 2L_m\frac{R_{ijk}}{R} + 2L_m\frac{R_{ijk}}{R_{ij}} - L(P_{ijk}^n)\frac{R_{ijk}}{R} \\
&= L(P_{ijk}^n)(1 - \frac{R_{ijk}}{R}) + 2L_m\frac{R_{ijk}}{R} + 2L_m\frac{R_{ijk}}{R_{ij}} \\
&\leq L_m(1 - \frac{R_{ijk}}{R}) + 2L_m\frac{R_{ijk}}{R} + 2L_m\frac{R_{ijk}}{R_{ij}} \\
&\leq L_m(1 + \frac{R_{ijk}}{R}) + 2L_m\frac{R_{ijk}}{R_{ij}} \\
&\leq 4L_m
\end{aligned} \tag{5.5}$$

□

The following theorem shows that FBP achieves constant service guarantees.

Theorem 9 *At any time, the difference between the numbers of bits transmitted by a flow to the output port in FBP and GPS is greater than or equal to $-4L_m$ and less than or equal to L_m, i.e. $-4L_m \leq toO_{ijk}(0,t) - \widehat{toB}_{ijk}(0,t) \leq L_m$.*

Proof. Without loss of generality, assume that $t \in [OF(P_{ijk}^n), OF(P_{ijk}^{n+1}))$. First, we prove $toO_{ijk}(0,t) - \widehat{toO}_{ijk}(0,t) \geq -4L_m$ as follows.

If $t \in [OF(P_{ijk}^n), OS(P_{ijk}^{n+1}))$, by noting $toO_{ijk}(t, OS(P_{ijk}^{n+1})) = 0$, we have

$$\begin{aligned}
&toO_{ijk}(0,t) - \widehat{toB}_{ijk}(0,t) \\
&= toO_{ijk}(0, OS(P_{ijk}^{n+1})) - toO_{ijk}(t, OS(P_{ijk}^{n+1})) - \\
&\quad \widehat{toB}_{ijk}(0, OS(P_{ijk}^{n+1})) + \widehat{toB}_{ijk}(t, OS(P_{ijk}^{n+1})) \\
&= (toO_{ijk}(0, OS(P_{ij}^{n+1})) - \widehat{toB}_{ijk}(0, OS(P_{ijk}^{n+1}))) + \\
&\quad + \widehat{toB}_{ijk}(t, OS(P_{ij}^{n+1})) \\
&\geq -4L_m + \widehat{toB}_{ijk}(t, OS(P_{ijk}^{n+1})) \\
&\geq -4L_m
\end{aligned} \tag{5.6}$$

Otherwise, if $t \in [OS(P_{ij}^{n+1}), OF(P_{ij}^{n+1}))$, by noting $toO_{ijk}(OS(P_{ijk}^{n+1}), t) = (t - OS(P_{ijk}^{n+1}))R$, we have

$$\begin{aligned}
& toO_{ijk}(0,t) - \widehat{toB}_{ijk}(0,t) \\
= & \; toO_{ijk}(0, OS(P_{ijk}^{n+1})) + toO_{ijk}(OS(P_{ijk}^{n+1}), t) - \\
& \widehat{toB}_{ijk}(0, OS(P_{ijk}^{n+1})) - \widehat{toB}_{ijk}(OS(P_{ijk}^{n+1}), t) \\
\geq & -4L_m + (t - OS(P_{ijk}^{n+1}))R - \widehat{toO}_{ijk}(OS(P_{ijk}^{n+1}), t) \\
\geq & -4L_m + (t - OS(P_{ijk}^{n+1}))(R - R_{ijk}) \\
\geq & -4L_m
\end{aligned} \tag{5.7}$$

Next, we prove $toO_{ijk}(0,t) - \widehat{toB}_{ijk}(0,t) \leq L_m$. Since flow scheduling uses WF^{2}Q, by Theorem 1 in [BZ96], we have $toB_{ijk}(0,t) \leq \widehat{toB}_{ijk}(0,t) + L_m$ and thus $toO_{ijk}(0,t) \leq toB_{ijk}(0,t) \leq \widehat{toB}_{ijk}(0,t) + L_m$. □

5.3.2 Delay Guarantees

FBP also achieves delay guarantees as stated by the following theorem. Note that $OF(p)$ and $\widehat{FF}(p)$ are the departure time of a packet p in FBP and GPS, respectively.

Theorem 10 *For any packet P_{ijk}^n , the difference between its departure time in FBP and GPS is greater than or equal to $L(P_{ijk}^n)(2/R - 1/R_{ijk})$ and less than or equal to $2L_m(1/R + 1/R_{ij})$, i.e.*

$$L(P_{ijk}^n)(2/R - 1/R_{ijk}) \leq OF(P_{ijk}^n) - \widehat{FF}(P_{ijk}^n) \leq 2L_m(1/R + 1/R_{ij})$$

Proof. First, we prove $OF(P_{ijk}^n) - \widehat{FF}(P_{ijk}^n) \geq L(P_{ijk}^n)(2/R - 1/R_{ijk})$. By the flow scheduling and input scheduling policies, we have $IS(p) \geq FS(p) \geq \widehat{FS}(p)$, or

$$IF(P_{ijk}^n) - \frac{L(P_{ijk}^n)}{R} \geq \widehat{FF}(P_{ijk}^n) - \frac{L(P_{ijk}^n)}{R_{ijk}} \tag{5.8}$$

By the output scheduling policy, we know $OS(p) \geq IF(p)$, or

$$OF(P_{ijk}^n) - \frac{L(P_{ijk}^n)}{R} \geq IF(P_{ijk}^n) \tag{5.9}$$

Combining (5.8) and (5.9), we then have proved that

$$OF(P_{ijk}^n) - \widehat{FF}(P_{ijk}^n) \geq L(P_{ijk}^n)(2/R - 1/R_{ijk}). \tag{5.10}$$

Next, we prove $OF(P_{ijk}^n) - \widehat{FF}(P_{ijk}^n) \leq 2L_m(1/R + 1/R_{ij})$. By (5.2), we know

$$\begin{aligned} OF(P_{ijk}^n) &\leq \widehat{OF}(P_{ijk}^n) + \frac{L_m}{R} \\ &= \widehat{IF}(P_{ijk}^n) + \frac{L_m}{R} + \frac{L_m}{R_{ij}} + \frac{L_m}{R} \\ &\leq \widehat{FF}(P_{ijk}^n) + \frac{2L_m}{R} + \frac{2L_m}{R_{ij}} \end{aligned} \tag{5.11}$$

□

5.3.3 Crosspoint Buffers

A nice feature of FBP is that it has a size boundary for the crosspoint buffers, which are significantly expensive on-chip memories. Define $toX_{ij}(t_1, t_2)$ and $toO_{ij}(t_1, t_2)$ to be the numbers of bits transmitted by F_{ij} during interval $[t_1, t_2]$ to X_{ij} and Out_j (i.e. out of X_{ij}) in FBP, respectively.

Lemma 9 *When a packet P_{ij}^n starts transmission to the output in FBP, the number of buffered bits at its crosspoint buffer X_{ij} is bounded by $3L_m$, i.e. $toX_{ij}(0, OS(P_{ij}^n)) - toO_{ij}(0, OS(P_{ij}^n)) \leq 3L_m$.*

Proof. By the definition of $OS(P_{ij}^n)$, P_{ij}^{x-1} has finished its output scheduling at time $OS(P_{ij}^n)$, then we have,

$$toO_{ij}(0, OS(P_{ij}^n)) = \sum_{a=1}^{n-1} L(P_{ij}^a) \tag{5.12}$$

On the other hand,

$$\begin{aligned}
& toX_{ij}(0, OS(P_{ij}^n)) \\
&= toX_{ij}(0, OF(P_{ij}^n) - \frac{L(P_{ij}^n)}{R}) \\
&\leq toX_{ij}(0, \widehat{OF}(P_{ij}^n) + \frac{L_m}{R} - \frac{L(P_{ij}^n)}{R}) \\
&= toX_{ij}(0, \widehat{IF}(P_{ij}^n) + \frac{2L_m}{R} + \frac{L_m}{R_{ij}} - \frac{L(P_{ij}^n)}{R}) \qquad (5.13)
\end{aligned}$$

Define $\widehat{toX}_{ij}(0,t)$ to represent the number of bits sent by the logical flow F_{ij} in the virtual dedicated channel with bandwidth R_{ij} during interval $[0,t]$. Recall that input scheduling uses the WF^{2}Q scheduling algorithm. Thus, by Theorem 1 in [BZ96], we know $toX_{ij}(0,t) \leq \widehat{toX}_{ij}(0,t) + L_m(1 - R_{ij}/R)$, and

$$\begin{aligned}
& toX_{ij}(0, OS(P_{ij}^n)) \\
&\leq \widehat{toX}_{ij}\left(0, \widehat{IF}(P_{ij}^n) + \frac{2L_m}{R} + \frac{L_m}{R_{ij}} - \frac{L(P_{ij}^n)}{R}\right) + L_m(1 - \frac{R_{ij}}{R}) \\
&\leq \widehat{toX}_{ij}\left(0, \widehat{IF}(P_{ij}^n)\right) + R_{ij}\left(\frac{2L_m}{R} + \frac{L_m}{R_{ij}} - \frac{L(P_{ij}^n)}{R}\right) + L_m(1 - \frac{R_{ij}}{R}) \\
&= \sum_{a=1}^{n} L(P_{ij}^a) + R_{ij}\left(\frac{2L_m}{R} + \frac{L_m}{R_{ij}} - \frac{L(P_{ij}^n)}{R}\right) + L_m(1 - \frac{R_{ij}}{R})
\end{aligned}$$

By (5.12) and (5.14), we can obtain

$$\begin{aligned}
& toX_{ij}(0, OS(P_{ij}^n)) - toO_{ij}(0, OS(P_{ij}^n)) \\
&\leq L(P_{ij}^n) + 2L_m\frac{R_{ij}}{R} + L_m - L(P_{ij}^n)\frac{R_{ij}}{R} + L_m(1 - \frac{R_{ij}}{R}) \\
&\leq L(P_{ij}^n)(1 - \frac{R_{ij}}{R}) + L_m\frac{R_{ij}}{R} + 2L_m \\
&\leq L_m(1 - \frac{R_{ij}}{R}) + L_m\frac{R_{ij}}{R} + 2L_m \\
&\leq 3L_m \qquad (5.14)
\end{aligned}$$

□

The following theorem gives the bound of the crosspoint buffer size.

Theorem 11 *In FBP, the maximum number of bits buffered at any crosspoint buffer at any time is bounded by $3L_m$, i.e. $toX_{ij}(0,t) - toO_{ij}(0,t) \leq 3L_m$.*

Proof. Without loss of generality, we assume that $t \in [OF(P_{ij}^n), OF(P_{ij}^{n+1}))$. If $t \in [OF(P_{ij}^n), OS(P_{ij}^{n+1}))$, we have

$$\begin{aligned}
& toX_{ij}(0,t) - toO_{ij}(0,t) \\
=\ & toX_{ij}(0, OS(P_{ij}^{n+1})) - toO_{ij}(0, OS(P_{ij}^{n+1})) - \\
& toX_{ij}(t, OS(P_{ij}^{n+1})) + toO_{ij}(t, OS(P_{ij}^{n+1})) \\
\leq\ & 3L_m - toX_{ij}(t, OS(P_{ij}^{n+1})) \\
\leq\ & 3L_m
\end{aligned} \tag{5.15}$$

Otherwise, if $t \in [OS(P_{ij}^{n+1}), OF(P_{ij}^{n+1}))$

$$\begin{aligned}
& toX_{ij}(0,t) - toO_{ij}(0,t) \\
=\ & toX_{ij}(0, OS(P_{ij}^{n+1})) - toO_{ij}(0, OS(P_{ij}^{n+1})) + \\
& toX_{ij}(OS(P_{ij}^{n+1}), t) - toO_{ij}(OS(P_{ij}^{n+1}), t) \\
\leq\ & 3L_m + toX_{ij}(OS(P_{ij}^{n+1}), t) - toO_{ij}(OS(P_{ij}^{n+1}), t) \\
\leq\ & 3L_m + toX_{ij}(OS(P_{ij}^{n+1}), t) - (t - OS(P_{ij}^{n+1}))R \\
\leq\ & 3L_m
\end{aligned} \tag{5.16}$$

□

5.3.4 Complexity Analysis

As can be seen, in order to transfer an incoming packet to the output link, flow scheduling, input scheduling, and output scheduling each is conducted once. The time complexity of both WF^{2}Q and WFQ has been shown to be $O(\log M)$ [Val07] [DKS89] to schedule M flows. Assuming that an input-output pair has at most M

flows, then the time complexity of flow scheduling is $O(\log M)$. The time complexity of input scheduling and output scheduling is the same $O(\log N)$, because each of the two scheduling stages handles N flows. Regarding space complexity, a packet needs two time stamps, for the virtual start time and finish time of a scheduling stage.

5.3.5 Implementation Advantages

FBP is practical to implement with a number of advantages. First, FBP can be implemented in a distributed manner, since there is no centralized scheduler, and different input ports or output ports need no information exchange. The virtual output finish time of a packet can be calculated based on its virtual input finish time by the input port and carried by the packet to the crosspoint buffer for output scheduling. Second, FBP can directly process variable length packets without SAR. Because of distributed scheduling, there is no synchronized operation between different input ports and output ports, and thus each can independently process packets of variable length one by one. Note that packets in most real networks are of variable length. Compared with fixed length cell scheduling, variable length packet scheduling can achieve higher throughput and shorter latency [Tur09, PY09]. Finally, FBP requires no speedup and has a small bounded crosspoint buffer size of $3L_m$, reducing the hardware cost.

5.3.6 Comparison with Existing Solutions

We summarize the comparison of FBP and existing flow-level bandwidth provisioning algorithms in [CGMP99] and [CIM05] in Table 5.1. First of all, we can notice that only FBP achieves $O(1)$ service guarantees, and avoids SAR for variable length packets. Since the other algorithms emulate PIFO OQ switches that run fair queuing algorithms at output ports, their performance guarantees are proportional to the number of flows at the output port, i.e. $O(MN)$. Also, they can only schedule

Table 5.1: Comparison with Existing Algorithms

	More speedup, more buffers	CCF, DTC, GBVOQ	FBP
Service guarantees	$O(MN)$	$O(MN)$	$O(1)$
SAR for var. len. packets	Yes	Yes	No
Crossbar speedup	3, 2	2	1
# of crosspoint buffers	$O(N^2)$, $O(N^3)$	0	$O(N^2)$
Time complexity	$O(\log M + \log N)$	$O(\log M + N \log N)$, $O(\log M + \log N)$, unbounded	$O(\log M + \log N)$
Distributed scheduling	Yes	No, No, Yes	Yes

fixed length cells. Next, comparing FBP with the algorithms in [CGMP99], we can see that FBP needs less speedup and fewer crosspoint buffers. Finally, comparing FBP with the algorithms in [CIM05], we can see that FBP achieves better time complexity and enables distributed scheduling. The trade-off is that FBP uses the CICQ switch structure with N^2 crosspoint buffers.

5.4 OpenFlow based Implementation

As stated in the introduction, Section 5.1, our second objective is to build an experimental prototype based on FBP to demonstrate a practical flow-level bandwidth provisioning solution. The prototype includes two components: OpenFlow switches running the FBP algorithm, and a NOX [nox] OpenFlow controller with a self-developed bandwidth provisioning component. On the one hand, we implement FBP in the OpenFlow version 1.0 software switch [opeb], which converts a Linux PC with multiple NICs to an OpenFlow switch. Implementing the FBP algorithm will enable the software switch to accurately guarantee the provisioned bandwidth at the flow level. On the other hand, we develop a NOX component as the control console for bandwidth provisioning, where the network administrator can define a flow and specify its allocated bandwidth. Leveraging the flow manipulation capability of the OpenFlow protocol, our prototype can flexibly define flows, allocate

bandwidth, and ensure the allocated bandwidth. In the following, we describe the implementation detail. The realistic performance data obtained from the prototype will be presented in Section 5.5.

5.4.1 FBP Enabled OpenFlow Software Switches

We implement FBP in the OpenFlow version 1.0 software switch, which is a user space program. In the earlier versions of the OpenFlow software switch, the *datapath* that manages the flow table was implemented as a kernel module. Starting from version 1.0, the entire program is implemented in the user space. The advantages of such a user space implementation include flexible development environment and good portability, but the trade-off is performance degradation caused by frequent context switches. Note that the main objective of the software switch is to provide a reference OpenFlow design for test and demonstration purposes, but not for use in production networks. Therefore, the software switch considers more about convenience of deployment and less about performance. In our case, the user space software switch allows us to develop the prototype faster and more economically than hardware switches.

The original software switch acts as a shared-memory OQ switch. When a packet arrives at the input NIC, the program copies the packet from the input NIC buffer to the main memory. It then searches the flow table for a matching flow for the packet. If there is a matching flow, the program will obtain the output NIC from the table entry, and immediately transfer the packet from the main memory to the output NIC buffer, from where the packet will be sent to the output link. Otherwise, if there is no matching flow, which means that the packet is the first one of a new flow, the program will forward the packet to the controller, and the controller will create a new entry in the flow table.

As can be seen, there is no concept of a crossbar in the original OpenFlow software switch, and thus our first task is to create a virtual buffered crossbar to emulate the CICQ switch. We allocate space in the memory for the VOQ buffers B_{ij}, and create the flow queues Q_{ijk} on demand, i.e. setting up a new flow queue when the controller creates a new entry in the flow table. We configure the bandwidth of the crossbar to be the same as that of the NIC and emulate the transmission delay from the VOQ buffer to the crosspoint buffer and from the crosspoint buffer to the output port. In the FBP enabled OpenFlow software switch, after a packet arrives at the input NIC, it is immediately retrieved to the flow queue in the memory using the *netdev_recv* function. The packet is then transmitted through the virtual crossbar, and finally delivered to the output NIC using the *netdev_send* function. *netdev_recv* and *netdev_send* are existing functions of the *netdev* module that manages the NICs.

The next challenge is to maintain accurate time stamps. For most Linux systems, the minimum time resolution is 1 μs [GET]. Further, to avoid excessive overhead by signal handling, the minimum time resolution provided by the *timeval* module of the original software switch is 1 *ms*. However, the effectiveness of FBP relies on accurate time stamps, and the existing time resolution is not sufficiently fine, especially for high speed switches. For example, assume that the minimum time resolution is 1 μs, and a software switch equipped with Gigabit NICs has 1 Gbps bandwidth. For simplicity, also assume that F_{ij1} is the only flow of In_i and Out_j, and thus R_{ij} =1 Gbps. If a packet P^n_{ij1} has length $L(P^n_{ij1})$ of 400 bits, and its actual flow start time $FS(P^n_{ij1})$ is 5 μs, then its actual flow finish time will be $FF(P^n_{ij1}) = FS(P^n_{ij1}) + L(P^n_{ij1})/R_{ij} = 5 + 0.4 = 5.4\ \mu s$. However, since the minimum time resolution is 1 μs, there is no way to differentiate 5 μs and 5.4 μs, and we have to round the latter to the former, which means the departure of P^n_{ij1} from its flow queue takes no time. More importantly, the error caused by the coarse time

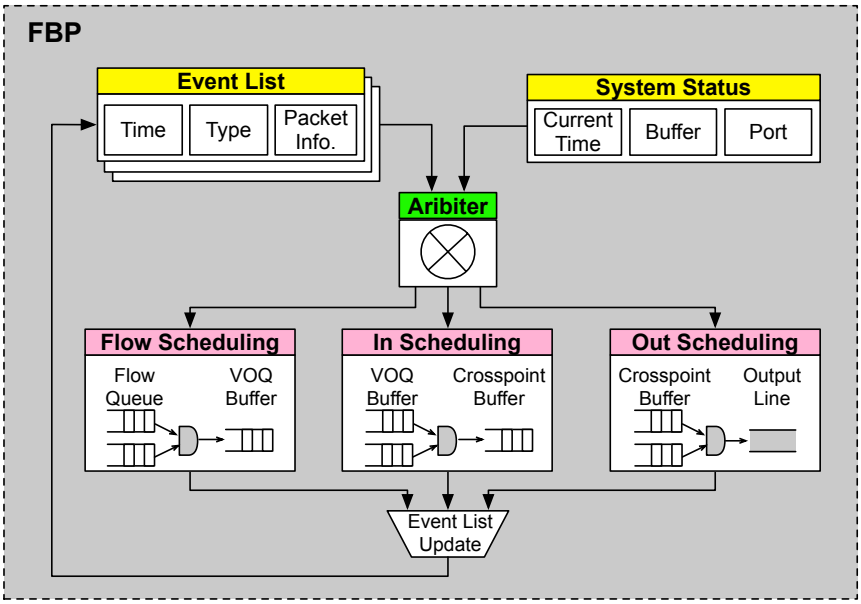


Figure 5.3: Event-driven Scheduling of FBP Enabled OpenFlow Software Switch

resolution will accumulate over time. To address the challenge, we maintain accurate logical time within the virtual crossbar, so as to calculate correct time stamps for scheduling. Only the packet arrival time is based on the original system time, and all other operations of FBP are based on the accurate logical time. Specifically, when a packet p is retrieved from the input NIC buffer, we call the existing *time_msec* function in the *timeval* module to obtain the packet arrival system time $A(p)$, which is an integer with *ms* time resolution. We then convert the integer system time value to the logical time as a double-precision floating-point number, and represent all the subsequent time stamps used by FBP as double-precision floating-point numbers. When the packet is sent to the output NIC, we obtain the logical time for the actual output finish time $OF(p)$, and use it to deduct the logical time $A(p)$ to derive the delay as a double-precision floating-point number. In this way, all the scheduling decisions of FBP are based on the more accurate double-precision logical time.

Finally, we extend the event driven mechanism of the original software switch to control the operation of the virtual crossbar, as illustrated in Figure 5.3. The original

program uses an event driven mechanism, and monitors two types of events: packet arrival and time out. The program is normally blocked, and wakes up to process the assigned job when an event happens. We add all the possible types of events of the virtual crossbar to the event list, each with the necessary information, including the event time, event type, and associated packet. All the events are linked in an increasing order of the event time. When a timer triggers, the program retrieves the first event in the event list and processes it. Note that processing an event may insert new events to the list. Because of the coarse resolution of the system time, multiple events may happen when the program wakes up, in which case the program will continue processing the event at the head of the event list until the time of the next event is in the future.

5.4.2 Bandwidth Provisioning NOX Component

NOX is an open-source OpenFlow controller written in C++ and Python. The C++ code provides fundamental low-level APIs that are compliant with the the OpenFlow protocol. The Python code implements the high-level control functionality, and interacts with the underlying C++ APIs. NOX enables customization with new functionality by adding new components written in Python.

We have developed a NOX component as the control console for the new bandwidth provisioning functionality. It accepts flow definition and bandwidth allocation as inputs, and sends OpenFlow commands as outputs to switches to set up flow table entries. For example, the network administrator can use the new NOX component to define the traffic from IP address 130.94.11.22 to 131.94.33.44 as a flow, and assign it 10 Mbps bandwidth. In our implementation, we use a configuration file to store all the flow definition and bandwidth allocation. The NOX component periodically checks the configuration file, and communicates the specified information

to the OpenFlow switches. Each line of the configuration file contains 13 entries. The first 12 entries are the packet header fields to define a flow [OPEc], and the last entry gives the allocated bandwidth of this flow. We add a timer for the NOX component to read the configuration file every five seconds, and use an array to store all the flow definition and bandwidth allocation information, with each array item corresponding to a defined flow.

Every time when the NOX component reads the configuration file, it compares the information read from the file with that already in the array. In the first case, if it detects a new flow defined in the configuration file, it adds the information to the array. When the first packet of the new flow arrives at a switch, the packet will be forwarded to the NOX controller. Our NOX component has a *packet_in_callback* function, which will be triggered by such a packet arrival event. By checking the packet header fields, the component recognizes that the packet belongs to the new defined flow, and uses the standard Layer 2 self-learning process to find a path for the flow. Next, the component adds a new entry in the flow table of each switch on the path, along with the provisioned bandwidth, by sending a flow table modification message of type *OFPFC_ADD*.

To send the provisioned bandwidth information from the controller to the switch, we need to modify the OpenFlow message format, the message sending function of the controller, and the message receiving function of the switch. First, we modify the flow modification message structure *ofp_flow_mod* in the *openflow.h* header file by adding a field named *bw* of type *uint32_t*. *openflow.h* defines the OpenFlow protocol format, and is shared by the controller and switch. To allow backward compatibility, for a regular flow without provisioned bandwidth, its *bw* field can be set to 0. Second, for the controller, we enhance the Python function *send_flow_command* in the *core.py* module and the C++ function *Pycontext::send_flow_command* in the

pycontext.cc module, to add the bandwidth information to the message sent to the switch. Third, for the switch, when it receives the *OFPFC_ADD* message, it adds a new flow table entry with the provisioned bandwidth. In addition, to store the bandwidth information in the flow table, we modify the structure of *sw_flow*, which stores all the information of a flow and is located in *switch-flow.h* header file. We add a new filed named *bw* of type *uint32_t* to store the bandwidth information. Future packets of the flow will match the newly added flow table entry, and will be transmitted by using the provisioned bandwidth.

In the second case, if the component detects that an already defined flow was removed from the configuration file, it sends a flow table modification message of type *OFPFC_DELETE* to the switches to delete the corresponding flow table entry. Future packets of this flow will be processed by default without reserved bandwidth.

In the third case, if the component detects that the allocated bandwidth of a defined flow changed, it first updates the bandwidth in the array. Although the OpenFlow protocol defines a flow table modification message of type *OFPFC_MODIFY*, it can only modify the associated actions. Alternatively, our component sends a flow table modification message of type *OFPFC_DELETE* to delete the existing flow table entry of each switch. When the next packet of this flow arrives at a switch, the switch will treat it as if it was the first packet of a new flow and send it to the controller. The controller will then set up a new flow table entry for each switch, but with the updated bandwidth. Future packets of the flow will then be transmitted with changed bandwidth.

5.4.3 Scalability of OpenFlow based Implementation

A good bandwidth provisioning solution needs to be scalable to support large numbers of flows and high traffic rates. We analyze the scalability of the OpenFlow

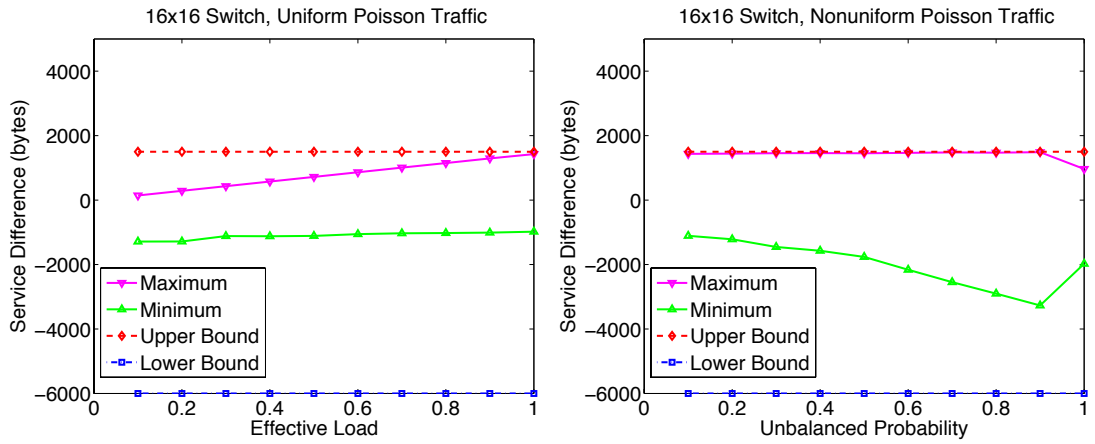
based implementation from the aspects of the switches and controller, respectively. For the switches, we have shown that our scheduling algorithms have low logarithmic time complexity, and thus can scale to high traffic rates. Further, it enhances scalability for switches of different roles to define flows at different granularity levels. Edges switches have only a small number of connected hosts, and thus can define a flow as the traffic generated by a single VM or application for flexible control. On the contrary, core switches handle enormous traffic, and the flows already shaped by edge switches can be combined as an aggregate flow to reduce management overhead. For the controller, it has been shown that an OpenFlow controller can handle all the flows of an enterprise network with tens of thousands of hosts [CFP+07]. In addition, OpenFlow has been considered in many recent data center designs [MPF+09] [AfRR+10], and the experiments demonstrate the feasibility to use a central controller to manage large scale data centers. Finally, there are several recent proposals [YRFW10] [CMT+11] to scale the control of OpenFlow-like flow networks, and they can be utilized to enhance the scalability of the controller.

5.5 Simulation and Experiment Results

We have implemented the FBP algorithm in a Java based network simulator and the OpenFlow software switch. In this section, we present the numerical results from the simulations and experiments, to evaluate our design and validate the analytical results in Section 5.3.

5.5.1 Simulation Results

In the simulations, we consider a 16×16 CICQ switch without speedup. Each input port or output port has 1 Gbps bandwidth. There are two flows from In_i to Out_j with $R_{ij2} = 2R_{ij1}$, and thus the total number of flows is $16 \times 16 \times 2 = 512$. The packet length is uniformly distributed between 40 and 1500 bytes, and packets



(a) Uniform Traffic (b) Nonuniform Traffic

Figure 5.4: Service Difference

arrive based on a Markov modulated Poisson process [PY09]. We use two traffic patterns. For traffic pattern one, or uniform traffic, we set $R_{ij} = R/N$, and change the effective load of the incoming traffic from 0.1 to 1 by step 0.1. For traffic pattern two, or nonuniform traffic, we fix the effective load to 1, and define R_{ij} by i, j and an unbalanced probability w as follows

$$R_{ij} = \begin{cases} R(w + \frac{1-w}{N}), & \text{if } i = j \\ R\frac{1-w}{N}, & \text{if } i \neq j \end{cases} \tag{5.17}$$

where w is increased from 0 to 1 by step 0.1.

Service Guarantees

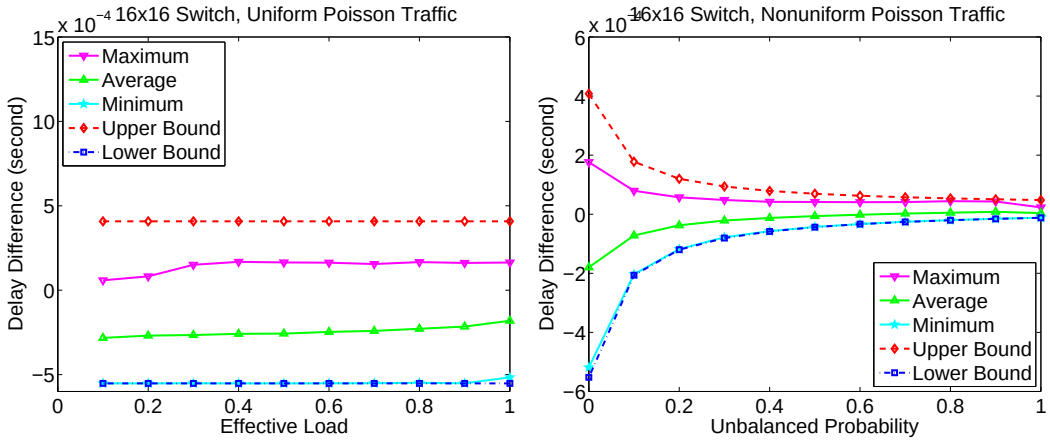
By Theorem 9, we know that the service difference of a flow in FBP and GPS at any time has a lower bound of $-4L_m$ and upper bound of L_m. We first look at the simulation data on service guarantees. Figure 5.4(a) shows the maximum and minimum service differences among all the flows during the entire simulation run under uniform traffic. As can be seen, the maximum service difference increases with the traffic load, but does not exceed the theoretical upper bound. The mini-

mum service difference is comparatively constant and always greater than the lower bound. The gap between the minimum service difference and the lower bound is caused by rounding the ratios of R_{ijk}/R_{ij} and R_{ijk}/R to integers in the proof of Lemma 8. In other words, the minimum service difference is determined by the bandwidth ratios but not the traffic load. Figure 5.4(b) shows the simulation data under nonuniform traffic. We can see that the maximum service difference is almost coincident with the upper bound. Note that the maximum service difference drops when the unbalanced probability becomes one. The reason is that in this case, all packets of In_i go to Out_i. Thus, there is no switching necessary, and packet scheduling is only conducted between the two flows of the same input-output pair. Therefore, the maximum service difference is $L_m(R_{ij2}/R_{ij}) = 1000$ bytes. On the other hand, the minimum service difference is always greater than the lower bound. It drops gradually when the unbalanced probability increases, and rises when the unbalanced probability becomes one, for the same reason as above. The low bound looks tighter under nonuniform traffic, because $\max_{i,j,k}\{R_{ijk}/R\}$ now has a greater value. The minimum service difference can keep getting closer to the lower bound by increasing the bandwidth ratios R_{ijk}/R_{ij} and R_{ijk}/R.

Delay Difference

Recall that Theorem 10 gives the upper bound and lower bound for the delay difference of a flow in FBP and GPS. Because the lower bound value in the theorem depends on the lengths of individual packets, it is not convenient to plot the figure. To eliminate the dependency, we calculate the lower bound for all packets as follows

$$L(P_{ijk}^n)(\frac{2}{R} - \frac{1}{R_{ijk}}) \geq \begin{cases} L_m(\frac{2}{R} - \frac{1}{R_{ijk}}), & \text{if } R_{ijk} \leq \frac{R}{2} \\ 0, & \text{if } R_{ijk} > \frac{R}{2} \end{cases} \tag{5.18}$$



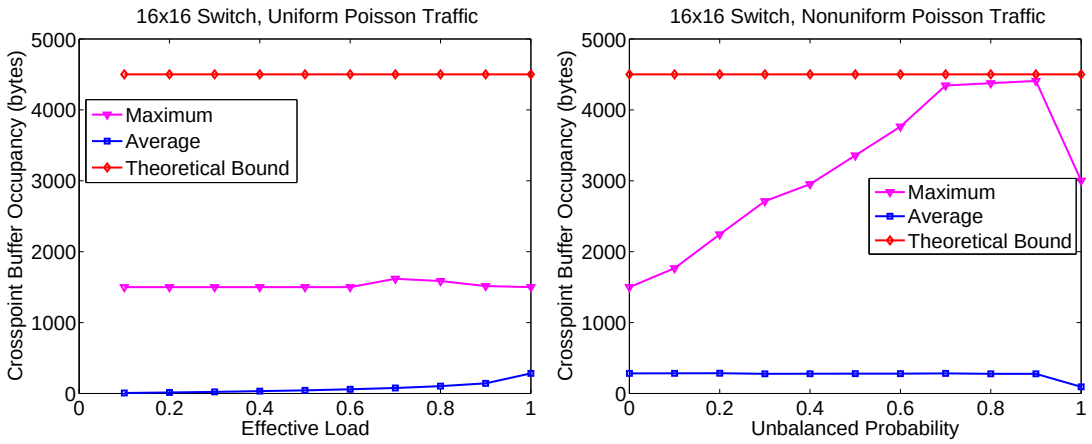
(a) Uniform Traffic (b) Nonuniform Traffic

Figure 5.5: Delay Difference

Figure 5.5(a) shows the maximum, average, and minimum delay differences of one representative flow F_{111} under uniform traffic. As can be seen, the minimum delay difference is almost coincident with the lower bound. The maximum delay difference is always less than the upper bound, and has a small value. This shows that under uniform traffic, FBP can well emulate GPS and a packet will not depart too late after its departure time in GPS. Note that the average delay difference is less than zero for all effective loads, which means that most packets leave earlier in FBP than in GPS when the incoming traffic is uniformly distributed. Figure 5.5(b) plots the data under nonuniform traffic. We can see that the simulation data fall perfectly within the theoretical bounds. With the increase of the unbalanced probability, the maximum delay difference increases, and the minimum and average delay differences increase.

Crosspoint Buffer Occupancy

We now look at the crosspoint buffer occupancy data and compare them with Theorem 11. Figure 5.6(a) shows the maximum and average crosspoint occupancies



(a) Uniform Traffic (b) Nonuniform Traffic

Figure 5.6: Crosspoint Buffer Occupancy

under uniform traffic. As can be seen, the maximum crosspoint occupancy is less than the theoretical bound $3L_m$ for all the effective loads. In addition, the average crosspoint occupancy is always less than 400 bytes, much lower than the maximum value. Figure 5.6(b) presents the data under nonuniform traffic. We can see that the theoretical crosspoint buffer size bound is tight. Specifically, the maximum crosspoint occupancy increase constantly with the unbalanced probability, and drops to 3000 bytes when the unbalanced probability becomes one. The average crosspoint occupancy is close to 300 bytes and drop to around 100 bytes when unbalanced probability becomes one.

5.5.2 Experiment Results

We install the FBP enabled OpenFlow software switch on Linux PCs for the following experiments. Each PC has an Intel Core 2 Duo 2.2 GHz processor, 2 GB RAM, and multiple 100 Mbps Ethernet NICs. The PC operating system is Ubuntu 10.04LTS with Linux kernel version 2.6.33. NOX version 0.8 [nox] is deployed as the OpenFlow controller.

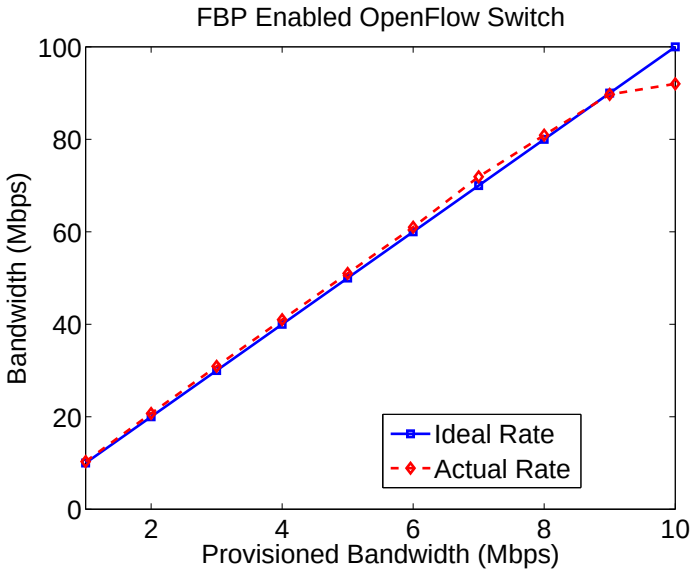


Figure 5.7: Experiment with Single Flow and Single Switch

Single Flow and Single Switch

In the first experiment, we compare the provisioned bandwidth of a flow with the measured bandwidth. We use a switch to connect two hosts, and set up an IPerf [ipe] TCP flow between the two hosts. By TCP congestion control, the TCP flow can automatically probe the available bandwidth in the link. We adjust the provisioned bandwidth of the flow from 10 Mbps to 100 Mbps by step 10 Mbps. Note that because the NIC has maximum bandwidth of 100 Mbps, its ideal throughput is also 100 Mbps. As shown in Figure 5.7, when the provisioned bandwidth is less than 90 Mbps, the throughput measured from the Iperf flow perfectly matches the expected value. However, when the provisioned bandwidth becomes 100 Mbps, the measured bandwidth is about 92.1 Mbps. The reasons might include the implementation overhead and the possibility that the NIC cannot reach its ideal throughput. As a comparison, the original OpenFlow software switch without FBP can achieve maximum bandwidth of about 94.5 Mbps.

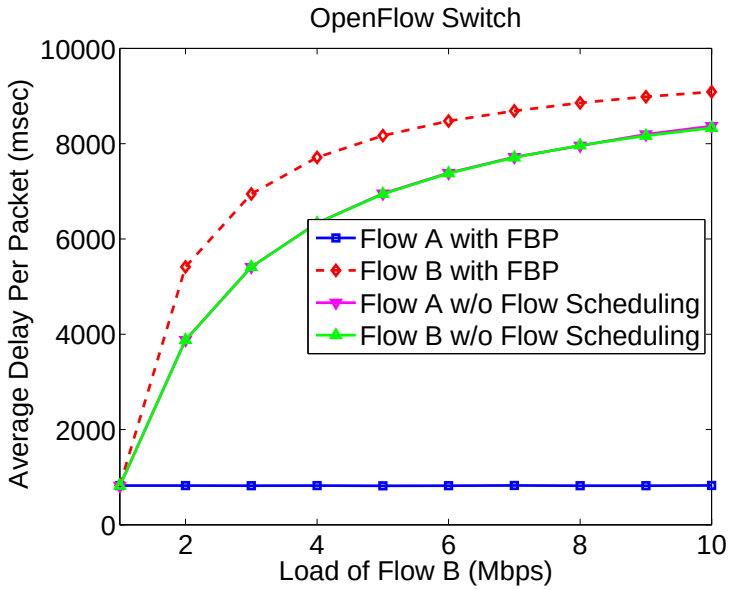


Figure 5.8: Experiment with Multiple Flows and Single Switch

Multiple Flows and Single Switch

In the second experiment, we compare FBP with a port-level bandwidth provisioning algorithm, i.e. without the flow scheduling phase. Similar as in the first experiment, a switch connects two hosts. There are now two IPerf UDP flows between the two hosts, which we call Flow A and Flow B, and they share the same switch input port and output port. We provision the bandwidth of each flow to be 1 Mbps. We fix the bandwidth of Flow A at 1 Mbps, and adjust the bandwidth of Flow B from 1 Mbps to 10 Mbps by 1 Mbps step. As shown in Figure 5.8, with the flow level bandwidth provisioning FBP, the average delay of Flow A remains constant no matter what the load of Flow B is. The average delay of Flow B rises quickly, because it injects traffic at a high rate than its provisioned bandwidth. On the contrary, with port level bandwidth provisioning, the average delay of both flows grow steadily with the load of Flow B. The results fully demonstrate that FBP is effective in achieving traffic isolation among flows and providing flow-level bandwidth provisioning.

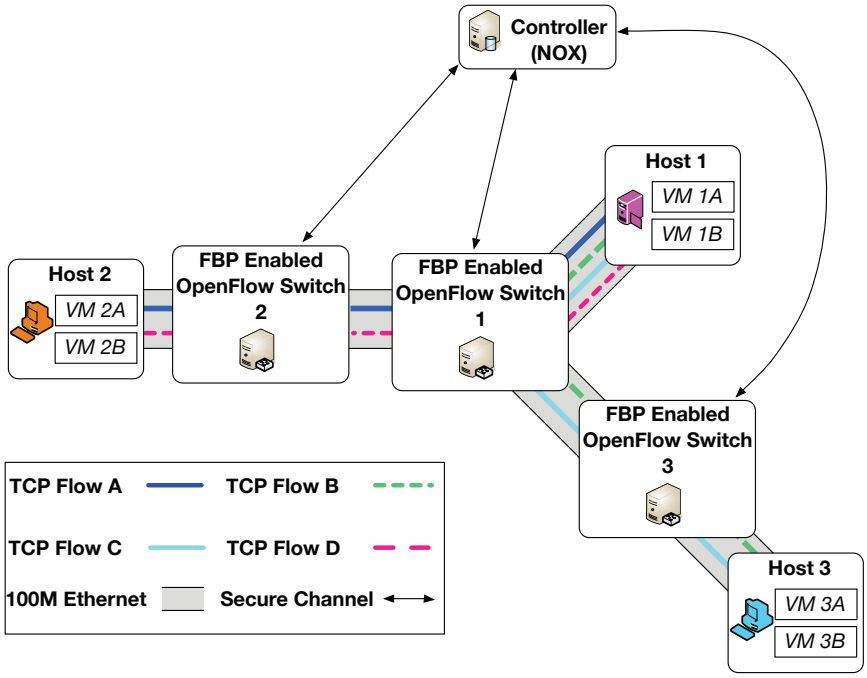


Figure 5.9: Topology of Experiment OpenFlow Network

Multiple Flows and Multiple Switches

In the third experiment, we set up an OpenFlow network with one controller, three switches, and three hosts, with the topology shown in Figure 5.9. Switch 1 connects Host 1, Switch 2, and Switch 3. Switch 2 connects Switch 1 and Host 2. Switch 3 connects Switch 1 and Host 3. Each host runs VirtualBox version 4.1.4 with two VMs. The VMs are configured with bridged networking [VIR] so that they will have public IP addresses. Denote the VMs on Host 1 as 1A and 1B, which emulate two TCP servers. Denote the VMs on Host 2 as 2A and 2B, and those on Host 3 as 3A and 3B, all emulating TCP clients. We set up four IPerf TCP flows: Flow A between VMs 1A and 2A, Flow B between VMs 1A and 3A, Flow C between VMs 1B and 3B, and Flow D between VMs 1B and 2B.

In the initial configuration, we set the provisioned bandwidth of Flows A, B, C, and D to be 15, 12, 8, and 6 Mbps, respectively. To measure the actual bandwidth of

each flow, we install WireShark on Switch 1 to capture packets of all the four flows. Figure 5.10 shows the continuous bandwidth measure of each flow by WireShark. Each pixel on the curve shows the average bandwidth of the flow during a one-second interval. We can see that the measured bandwidth of each flows perfectly matches the provisioning amount, demonstrating that our solution is effective in a multi-switch and multi-flow environment.

Before the 30^{th} second, we modify the configuration to increase the provisioned bandwidth of Flow A to 20 Mbps. When the NOX component reads the configuration, it detects the changed bandwidth allocation, and sends a command to the switches to realize this change. As can be seen from the figure, the measured bandwidth of Flow A quickly changes from 15 Mbps to 20 Mbps, and the measured bandwidth of the other flows remains the same. In a similar manner, before the 60^{th} second, we modify the configuration to exchange the provisioned bandwidth amounts of Flows B and C, and before the 90^{th} second, we reduce the provisioned bandwidth of Flow D to 2 Mbps. We can see that the prototype successfully handles all bandwidth change requests, with the bandwidth of designated flows smoothly changing to the new values, and the bandwidth of the remaining flows keeping stable.

5.6 Summary

Flow-level bandwidth provisioning ensures allocated bandwidth for individual flows, and is especially important for virtualization based computing environments such as data centers. However, existing solutions suffer from a number of drawbacks, including high hardware and time complexity, inability to achieve constant service guarantees, and inefficiency to process variable length packets. In this chapter, we have studied flow-level bandwidth provisioning for CICQ switches in the OpenFlow context. First, we propose the FBP algorithm, which reduces the scheduling prob-

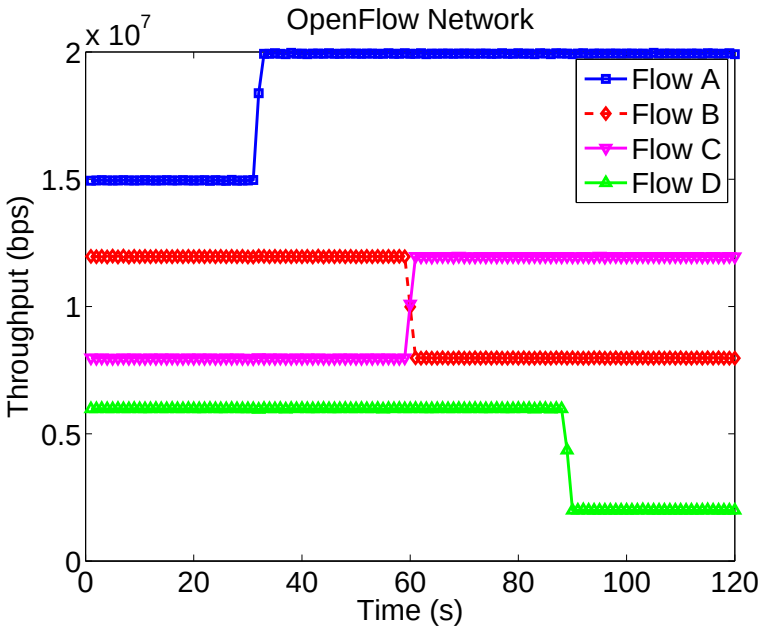


Figure 5.10: Experiment with Multiple Flows and Multiple Switches

lem on CICQ switches to multiple stages of fair queuing, with each stage utilizing a well studied fair queuing algorithm. We show by theoretical analysis that FBP can closely emulate the ideal GPS model, and achieve constant service guarantees and tight delay guarantees. FBP is economical to implement with bounded crosspoint buffer sizes and no speedup requirement, and is fast with low time complexity and distributed scheduling. In addition, we implement FBP in the OpenFlow software switch to build an experimental prototype. In conjunction with the existing capability of OpenFlow to flexibly define and manipulate flows, we have thus demonstrated a practical flow-level bandwidth provisioning solution. Finally, we conduct extensive simulations and experiments to evaluate our design. The simulation data successfully validate the analytical results, and the experiment results demonstrate that our prototype can accurately provision bandwidth at the flow level.

CHAPTER 6

HOST-NETWORK ENERGY EFFICIENCY CO-OPTIMIZATION FOR DATA CENTER NETWORKS

This chapter investigates the energy efficiency challenge for the DCN to achieve high energy conservation during off-peak traffic hours. Although there exist a number of energy optimization solutions for DCNs, they consider only either the hosts or network, but not both. Such separated optimization processes may lead to low energy saving performance and decline in the DCN's service quality. In this chapter, we propose a joint optimization scheme that simultaneously optimizes VM placement and network flow routing to maximize energy savings, and avoids server and network congestions to guarantee the service quality.

To effectively combine host and network based optimization, the joint optimization scheme utilize a unified representation method that converts the VM placement problem to a routing problem. In addition, to accelerate processing the large number of servers and an even larger number of VMs, the scheme takes a parallelizing approach that divides the DCN into clusters based on subnet IP addresses, and processes the clusters in parallel for fast completion. Further, to quickly find efficient paths for flows, the scheme employs a fast topology oriented multipath routing algorithm that uses depth-first search to quickly traverse the network and uses the best-fit criterion to maximize flow consolidation. We also build an OpenFlow based prototype to experimentally demonstrate the effectiveness of the scheme.

6.1 Introduction

Recent years, the size of data centers are growing rapidly. Some modern data centers are reported to contain more than 300k servers [NUM]. The energy consumptions of these huge data centers also increase significantly. It is estimated that 100 billion

kWh of energy are consumed by data centers annually and number is still growing [dat]. Therefore, improving the energy efficiency has become one of the top considerations when designing new data centers. Among the total energy consumption of a data center, the networking part accounts for 20% to 50% [SLX10] [AMW+10]. The data center network (DCN) provides connections with full bisection bandwidth to the servers in the data center. Because of the burty nature of the data center traffic, DCN might not be able to fully utilize the provisioned bandwidth. From the traffic pattern of any data centers, it is clear that there are huge differences between daytime and nighttime traffic volume, and between weekdays and weekends traffic volume. In traditional data center, even during the off-peak period, all switches and servers are powered on [GLF+00] [PBS+03]. And due to the huge energy overhead of the servers and switches, data center's energy consumption is not linear with its total load. In other words, energy is wasted in traditional data centers which has no energy efficiency optimization mechanism.

The general method of saving energy in data centers is to power off unnecessary devices. In literatures, two directions of finding unnecessary devices are studied [GWT+08] [GLL+09] [GLM+08] [GHJ+09] [MPF+09] [AFLV08]. One direction is to consolidate Virtual Machines (VMs) into fewer number of servers so that the idle servers can be powered off [HSM+10]. The other direction targeting the network side is to consolidate network flows onto fewer number of switches, and then power off those idle ones [MPZ10].

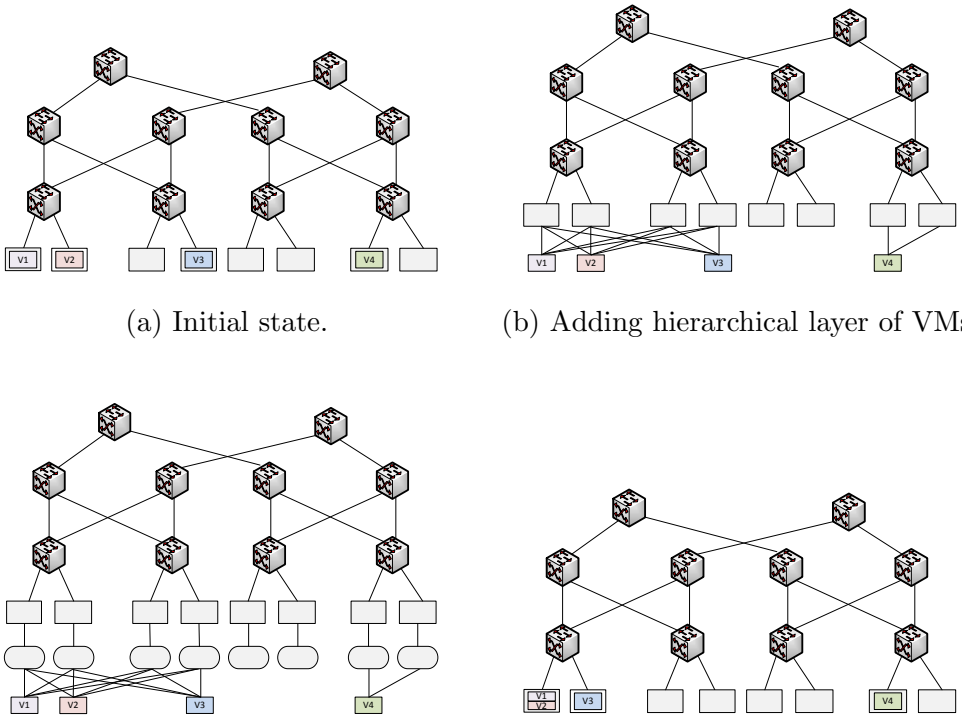
In this chapter, we study the energy efficiency optimization problem in data center networks, and propose a novel host-network energy efficiency co-optimization scheme which considers the VM and network flow consolidation simultaneously. There are several challenges. Our first challenge is how to coordinate the VM placement optimization and the flow routing optimization. We propose an unified rep-

resentation method which transforms the VM placement problem to adapt the flow routing problem. Therefore, one single optimization scheme can solve both problems. The next challenge is the speed of the flow routing path search. We propose a topology-aware recursive multipath routing algorithm which utilizes the depth first search algorithm to quickly traverse the hierarchy of the DCN, and utilizes the best fit to find the most proper flow routing path. Finally, the last challenge is how to increase the scalability of the optimization scheme so that it can be implemented in large data centers. We propose a parallel processing approach which divides the target data center into clusters and optimizes the clusters simultaneously. In addition, we have implemented the proposed scheme in a prototype, and conducted extensive experiments to evaluate the performance of the proposed scheme. The experiment results have demonstrated that our host-network energy efficiency co-optimization is effective and practical in improving the data center's energy efficiency.

6.2 Optimization Challenges and Solutions

In this section, we describe the design considerations in order for the host-network energy efficiency co-optimization scheme to be both efficient and scalable.

The first challenge is to solve the VM placement problem and the flow routing problem simultaneously for effective joint optimization. An idea solution is an unified representation of these two problems. Hierarchical structure of DCNs is analyzed in order to identify the unified representation of the VM placement and the flow routing. A multiple-layer fat tree based DCN is shown in Figure 6.1(a). It is shown that the VM and host relationship is similar to that of host and switch, since a VM picks the server just as a host picks the hosting switches. Based on this observation, we add an additional hierarchical VMs layer to the DCN. Specifically, a new node is added for each VM, and it connects with the host by a link if it can



(a) Initial state.

(b) Adding hierarchical layer of VMs.

(c) Adding dummy nodes for memory capacity constraints.

(d) Optimization results.

Figure 6.1: Unified representation of VM placement and flow routing.

migrate to the host. One simple example is shown in Figure 6.1(b). Figure 6.1(b) shows that V1, V2, and V3 are able to migrate to any host that is connected by the same aggregation switch. Also, V4 is able to migrate to any server that is connected by the same ToR switch. In the host-network energy efficiency co-optimization, the routing paths are searched for flows between VM pairs. If there is a path between a VM and a host, the VM will be hosted by that host. This provides a unified view for VM placement and flow routing.

In order to determine the capacity of the newly added links between VMs and host, we let the VMs connect to dummy nodes instead of to the hosts. Specifically, as shown in Figure 6.1(c), a dummy node is created for each host, and a link is added to connect the dummy node and the VMs. The capacities of the links are the

memory capacities of the hosts, since the host memory capacity may constrain the connection between the VMs and the host. It should be noticed that the VM node is different from physical server node when calculating flow routings, since a VM sends all its flows through the same host, and it selects only one link to connect to different dummy noes. Therefore, all the multiple traffic flows of a VM should share the same path between the VM and the hosting host. This should be considered when developing the unified representation of both types of optimization problems.

The other challenge is how to accelerate the processing of huge number of VMs in a data center The main idea is to use a parallelizing approach which divides the DCN into clusters based on subnet IP addresses, and processes them in parallel for fast completion. We assume that a VM will only migrate within its own subnet [MPF+09], and the servers and VMs in the same subnet are organized as a cluster. Intra-cluster and inter-cluster processing are separated to reduce the scale of the problem. For intra-cluster processing, we find routing paths for all flows between VMs in the same cluster and also determine the placement of the VMs. For VMs that only have inter-cluster flows, their placements are calculated according to their memory and bandwidth demands in inter-cluster processing. This is because taht DCN topology is usually symmetric, and the VM placement may not affect inter-cluster flow routing. The cluster-based parallel processing method reduces the solution search space and allows faster completion. In addition, intra-cluster processing in different clusters can be done in parallel to reduce the processing time since they are independent.

6.3 Host-Network Energy Efficiency Optimization Scheme

In this section, we present a topology aware multipath routing algorithm that quickly finds routing paths for inter-cluster and intra-cluster flows.

The key of the algorithm is to utilize the depth-first search to find a series of best-fit connections among the DCNs switch hierarchy. A DCN usually has a hierarchical network architecture which consists of multiple layers of switches. The depth-first search will traverse the hierarchy quickly to find the necessary layers to interconnect the two VMs. For redundancy considerations, there are usually more than one links between two switches. In order to better consolidate the flows, we employ the best-fit criterion to find the link that provide the smallest and sufficient bandwidth capacity. If the search cannot find any link with sufficient bandwidth, the search needs to backtrack to the previous layer and use best-fit to find another link among the rest of the links. As disused in Section 6.2, we divide the routing search into two processing stages, intra-cluster processing and inter-cluster processing.

First, the scheme searches routing paths for intra-cluster flows. It sorts the VMs by their memory demands in a descending order. Then, the scheme starts the unified VM placement and flow routing path search. It selects the VM with the largest memory demand and search for the routing path of its inter-cluster flows one by one. The routing path search has the following three steps. The first step is to determine the lowest connecting layer of the DCN hierarchy that connects the source and destination VMs. The lowest connecting layer becomes the highest layer that the routing path search is allowed to go up to. In other words, we localize the routing paths and thus save the higher layer switches for future flows. Since the network topology and the IP address assignment rules are known in advance, we can easily determine the lowest connecting layer by comparing the IP addresses of the source and destination VMs. After the lowest connecting layer is found, the scheme employs the depth-first and best-fit method to search routing paths for intra-cluster flows. Starting from the source VM, the scheme searches upstream in the network hierarchy and choose the link to the higher layer with the best-

fit available bandwidth. When reaching the connecting layer, the scheme changes the search direction downstream. It will apply the same depth-first and best-fit method. The search stops when it reaches the destination VM. As a result, the switch sequence along the search path becomes the routing path of the intra-cluster flow. As described in Section 6.2, we convert the VM placement problem to be part of the unified flow routing path problem. In other words, the scheme searches for the optimum VM placement and the optimum flow routing path simultaneously. If one or more VMs of the flow do not have hosting server before the routing path search, the server(s) on the routing path is(are) the new hosting server(s). When all the flows of the VM find their routing paths, the scheme will continue the search for the VM with the next largest memory demand. Table 6.1 shows the pseudo code of the depth-first and best-fit search.

Then the scheme searches routing paths for Inter-cluster flows. It applys the depth-first and best-fit approach to find the routing paths for the flows whose source and destination VM are in different clusters by . In the intra-cluster processing stage, all VMs that have intra-cluster flows, have found their placement hosts and corresponding ToR switches. If an inter-cluster flow is associated with one of such VMs, the scheme will only determine the routing path between the two hosts that are hosting the source and destination VMs. The scheme uses the same depth-first and best-fit approach as in the intra-cluster processing stage. If either the source or destination VM has not found its placement host, the routing algorithm will include the VM placement section in the routing path finding by adding a dummy node layer in the network hierarchy. Figure 6.1(d) illustrates the host-network energy efficiency co-optimization result of the example DCN in Figure 6.1(a).

```
DFS(G, a, b, d) // G: network, a: source, b: destination, d: demand
1   H = necessary-layer-to-connect(G, a, b);
2   path = {};
3   u = a; // temp variable indicating current location
4   next = 1; // flag indicating search direction, 1: upstream, -1: downstream
5   return SEARCH(u, path, next);

SEARCH(u, path, next) {
1   path = path + u;
2   if (u = b) return true;
3   if ( layer-of(u) = H) next = −1; // reverse search direction
                                     // after reaching connecting layer
4   if ( next = −1 && layer-of(u) = 1) return false; // failure at bottom layer
5   links = links of u to layer (layer-of(u) + next) and with available bandwidth ≥ d;
6   found = false;
7   while (links ≠ ∅ && found = false) {
8      v = best-fit(links); links = links \ {v};
9      found = SEARCH(v, path, next);
10  };
11  return found;
}
```

Table 6.1: Pseudo code description of depth-first best-fit search.

6.4 Prototype Implementation

In this section, we describe our implementation of the proposed scheme in a prototype using the Beacon OpenFlow controller, HP ProCurve OpenFlow switches, VMware vCenter server, and VMware ESXi hypervisor.

6.4.1 Hardware and Software Configuration

We have built a 4-pod and 16-host fat-tree prototype, as shown in Figure 6.2, to demonstrate the effectiveness and practicalness of our optimization algorithm in real networks. We utilize 2 OpenFlow enabled 48-port HP ProCurve 6600 switches running firmware version K.15.06.5008, and create 20 virtual switches. Each virtual switch is assigned with 4 ports, except that the first core layer switch has 3 extra

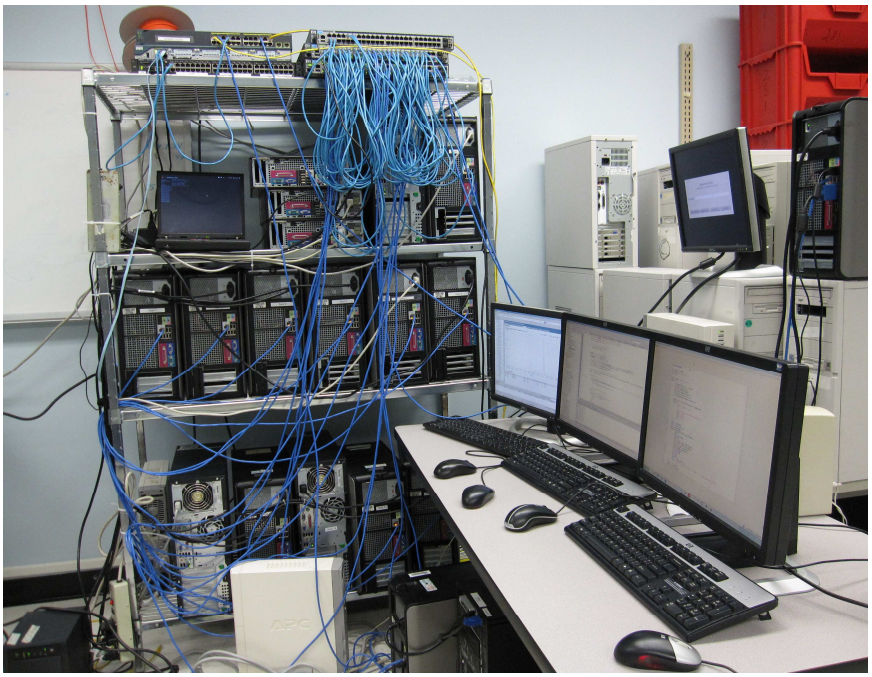


Figure 6.2: Photo of our prototype.

ports to allow connections for management nodes, including VMware vCenter server, Network File System (NFS) server, and DHCP server. All switches are managed by Beacon OpenFlow controller version 1.0.0 with a self-developed Equinox framework bundle that implements our optimization algorithm. Each host is running VMware ESXi hypervisor version 5.0.0 to host VMs running operating system of Ubuntu Server 12.04.1 LTS 64 bit. The hosts and VMs are configured to request IP address upon startup through DHCP protocol. When the controller detects the DHCP discovery message sent by a host or a VM, it records the host's or the VM's MAC address and location based on which input port of which ToR switch received the message. The IP address of the host or VM is updated when the controller detects the DHCP offer message. All hosts and VMs are remotely managed by VMware vCenter server version 5.0.0. Each VM's file system is provided by a NFS server implemented on a Linux PC running Ubuntu version 12.04.

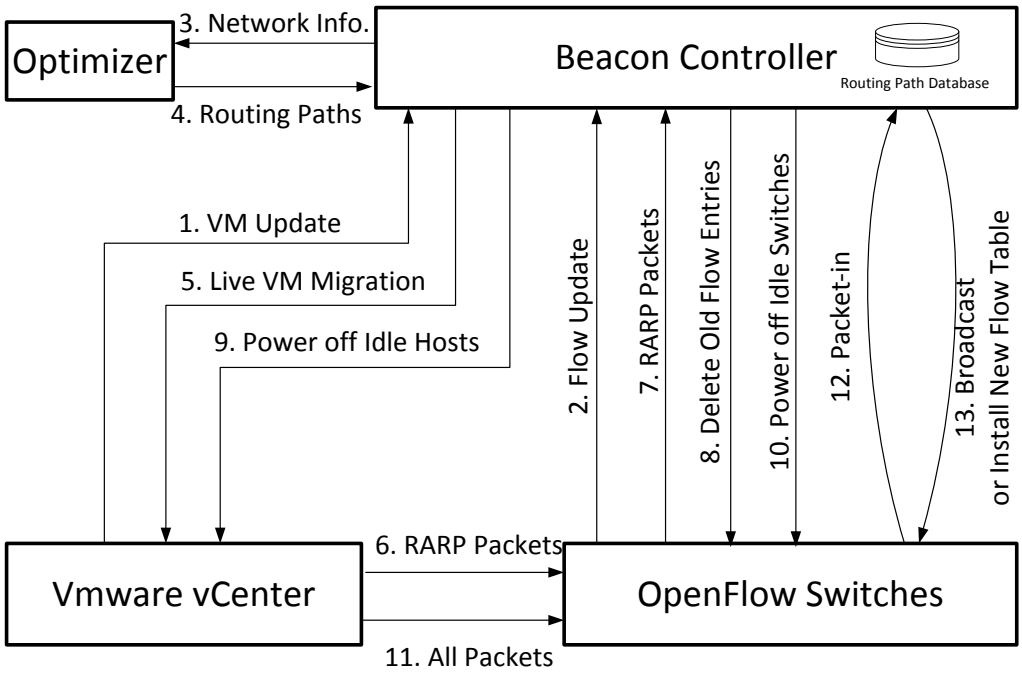


Figure 6.3: Major processes of the optimization.

Iperf UDP flows are employed to emulate the production traffic in data centers. The controller assigns initial routing paths to the flows. The initial routing paths are calculated by using Shortest Path Routing algorithm. If there exist multiple routing paths from the source to the destination, the controller selects one of them randomly. For each switch on the routing path, the controller also calculates each flow's input and output ports. The controller installs the flow table entries to all the switches on the routing path by sending them `ofp_flow_mod` messages with the flow's match information and the calculated input and output ports.

6.4.2 Optimization

Since the Beacon controller has the view of the entire network, we integrate the optimizer into the controller as one of its Equinox framework bundles. The optimizer can access all the information gathered in the initialization stage. The optimization

executes cycle by cycle and each cycle follows the four steps described below. The major exchanged data in one optimization cycle are shown in Figure 6.3.

Network Status Update

Before executing the optimization algorithm, the controller takes a snapshot of the current network. The main purpose of this step is to update the network topology, the VMs information and the flows information that might have changed between two optimization cycles.

For the switches and links, the controller utilizes the Link Layer Discovery Protocol (LLDP) implemented in the original Beacon controller. The network topology information is stored inside the controller and will be used later in the optimization stage describe in the next subsection. The memory and CPU capacities of each host are fetched from the host management node - VMware vCenter. Specifically, the controller sends `Get-VMHost` command with the host's IP address to the VMware vCenter through the VMware vSphere PowerCLI interface. The VMs location and capacity information can be gathered by using the similar methods. For the VM information, the controller sends a `Get-VM` command to the VMware vCenter to update each VM's location, CPU demand and memory demand. For the flow information, the controller sends `ofp_stats_request` messages of type `OFPST_FLOW` to each virtual switch to request the flow statistics. From the statistic reply messages, each flow's average bandwidth demand can be estimated by the flow's duration and total traffic amount. The controller stores the updated information in its database. Note that only the bandwidth of large flows will be stored, since these flows have the major impacts on the optimization result.

Optimal Network Scheme Calculation

Based on the network status snapshot, the optimizer executes the host-network joint optimization described in Section 6.3. An optimal network scheme, including the VM's new location and the flow's new routing path, is calculated. The routing path of each large flow are stored in a HashMap dictionary in the controller. The key of each map entry is a string that uniquely identifies each individual flow. The value of each map entry contains the flow table information of each switch on the routing path, including the switch's datapath ID, and the input and output port of the flow. In our prototype, Iperf UDP flows among VMs are the large flows. We concatenate the source VM's MAC address, the destination VM's MAC address and the transport layer port number of the destination VM to form the HashMap key.

For example, one Iperf flow is from VM 2 with MAC address `00:00:00:00:00:02` to VM 6 with MAC address `00:00:00:00:00:06` on transport layer port 5001. First we transform the MAC address to **Long** type and the source and destination MAC addresses become 2 and 6, respectively. Then, the HashMap key should be the string of `"265001"`. As shown in Figure 6.4, we assume that the optimized locations of VM 2 and VM 6 are on host 1 and host 4, respectively.We also assume that the optimized routing path of this Iperf flow is from ToR switch 301's port 1 to port 2, then from aggregation switch 202's port 12 to port 14 and then from ToR switch 302's port 7 to port 8. Then the routing path of this flow should be `<<301, 1, 2>, <202, 12, 14>, <302, 7, 8>>`.

VM Location and Flow Routing Path Adjustment

In this stage, the controller passes the optimization result to the VMWare vCenter which will execute live VM migrations to adjust the VM locations. Then the controller will adjust the flow routing paths accordingly. In out prototype, the live VM

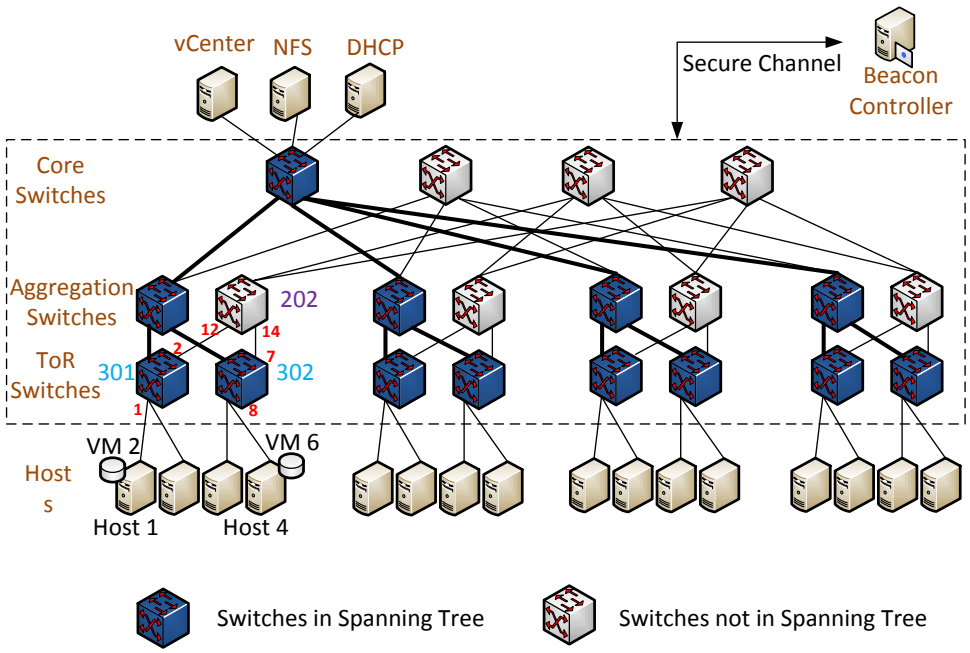


Figure 6.4: Fat-tree topology of the prototype.

migration is implemented by the VMware vMotion migration command `Move-VM`. Switches and hosts will be powered on if they are included in the optimization result.

Upon the completion of each VM's live migration, the VM will broadcast several `RARP` messages to announce its new location. When the controller detects such messages, it will delete flow table entries that are calculated based on the VM's old location. To be specific, the controller sends two messages to each switch to delete all flow entries that are related to the just migrated VM. One message is to delete flow entries whose source is the VM, while the other message is to delete flow entries whose destination is the VM. The two messages have the similar structure. They are both `ofp_flow_mod` messages of command type `OFPFC_DELETE`. The only differences are in the the flow match attribute and in the wildcards attributes. One sets the data link layer source address in the wildcard attributes as the address of

the migrated VM and set the wildcards to `OFPFW_ALL^OFPFW_DL_SRC`. The other one sets the data link layer destination address as the address of the migrated VM and set the wildcards to `OFPFW_ALL^OFPFW_DL_DST`.

Powering Off Idle Hosts, Switch Ports and Switches

In this stage, the controller sends commands to power off idle hosts, idle switch ports and idle switches. A host is considered idle if it is not hosting any active VM. A switch port is considered idle if no traffic is on the port currently, and according to the optimization result, no traffic will pass through this port before the next optimization cycle. A switch is considered idle if all of its ports are idle. For the idle hosts, the controller sends out `Stop-VMHost` commands through the VMware PowerCLI interface to power them off. For the idle switch ports and idle switches, the controller sends port down commands and switch power off command through the switch's command line interface to power them off, repectively.

The optimization cycle completes, when all idle hosts and idle switches are powered off. After this moment, when a packet is sent to the controller to find the routing path, 1) if the packet is a broadcast packet, the controller will ask the switches in the spanning tree to flood this packet; 2) if the packet is a unicast packet and belongs to a large flow with a stored routing path, the controller will fetch the path in the HashMap database and then install flow table entries to switches on the routing path; 3) if the packet is an unicast packet but belongs to a flow without a stored routing path, the controller will calculate a random routing path based on the current topology by using the Shortest Path Routing algorithm.

6.5 Prototype Experiments

We have conducted experiments in our prototype to evaluate the performance of the optimization scheme. In this subsection, we first describe the experiments configura-

tion, and then present the experiment results to demonstrate that the optimization scheme is effective and practical.

Experiment Configuration

Two experiments are executed. In the fist experiment, the average memory and traffic load of the prototype is set to be 15%, while in second experiment, the load is set to be 30%. VMs and network flows are generated according to the load and the following rules. Each VM's memory is randomly selected between 250 MB and 500 MB, and its CPU demand is randomly selected between 250 MHz and 500 MHz. The initial location of each VM is also randomly selected among the hosts. We adjust the number of VMs to meet the host utilization goal of each experiment. Each VM is configured to send out one Iperf UDP flow in average. The normal size of the flows are randomly selected between 20 Mbps and 250 Mbps and we adjust the flow size to meet the network utilization goal of each experiment if necessary. For the experiment with 15% memory and traffic load, 21 VMs and 24 Iperf UDP flows are generated; while for the second experiment with 30% memory and traffic load, 42 VMs and 49 Iperf UDP flows are generated.

Both experiments run the following processes. Initially all switches, hosts and VMs are powered on and all flows are started. We measure and record the current power consumption. Then, we run one full optimization cycle. After the optimization completes, we measure the power consumption again and compare it with the value before. Note that only the power consumption of the hosts are measured both before and after the optimization. This is because that the physical switches contains huge power consumption overhead and the power consumption of each virtual switch cannot be accurately measured. We employ Kill-A-Watt power meter model P4320 to measure the power consumption.

Flow Direction	Flow Bandwidth (Mbps)	Outgoing Traffic of Host 4 (Mbps)	Incoming Traffic of Host 4 (Mbps)
VM 4 → **VM 7**	83.7	-	83.7
VM 7 → VM 19	90.1	90.1	-
Total	**173.8**	**90.1**	**83.7**

(a) Before Optimization

Flow Direction	Flow Bandwidth (Mbps)	Outgoing Traffic of Host 4 (Mbps)	Incoming Traffic of Host 4 (Mbps)
VM 4 → **VM 7**	83.7	-	-
VM 4 → VM 17	27.8	27.8	-
VM 7 → **VM 19**	90.1	-	-
VM 9 → VM 3	26.8	26.8	-
VM 11 → **VM 9**	22.7	-	-
VM 11 → VM 20	56.0	56.0	-
VM 19 → VM 3	119.1	119.1	-
Total	**396.2**	**229.7**	**0**

(b) After Optimization

Table 6.2: Flow configuration and traffic amount of Host 4

Experiment Results - Routing Path Control

In order to verify whether the optimization adjusts the flow's routing paths correctly, we study the traffic amount of the hosts to see if they are as same as expected. Due to space limitation, we only present the result of Host 4 in this chapter. The results of other hosts follow the similar pattern and lead to the same conclusion.

Table 6.2(a) and (b) give the detailed configuration of the flows and traffic amount of Host 4, before and after the optimization, respectively. The VMs with bold name are the ones hosted by Host 4. Thus, before the optimization, only VM 7 is hosted by Host 4; and after the optimization, 3 additional VMs are hosted by Host 4, including VM 4, VM 11 and VM 19. If either a flow's source or destination VM is on Host 4, this flow accounts Host 4's network traffic. The traffic's direction and amount is the same as the flow's direction and bandwidth, respectively. For example, after the optimization, the flow from VM 4 to VM 17 accounts 27.8 Mbps of Host 4's total outgoing traffic. If both the source and destination VMs of a flow are

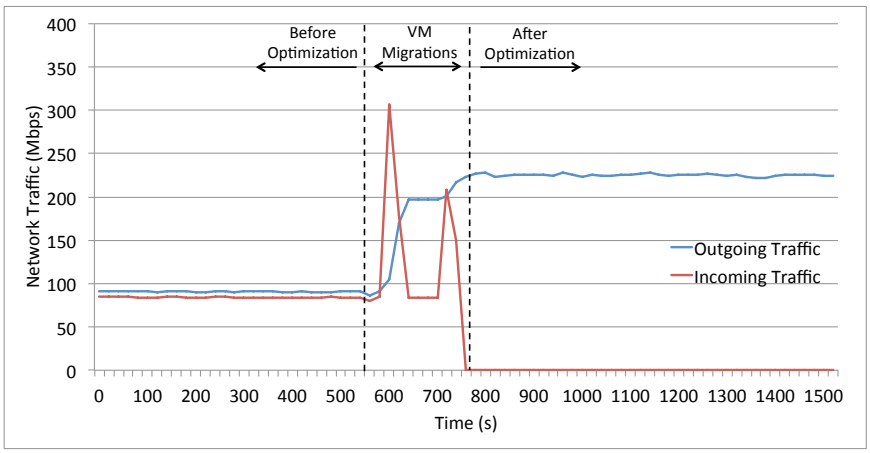
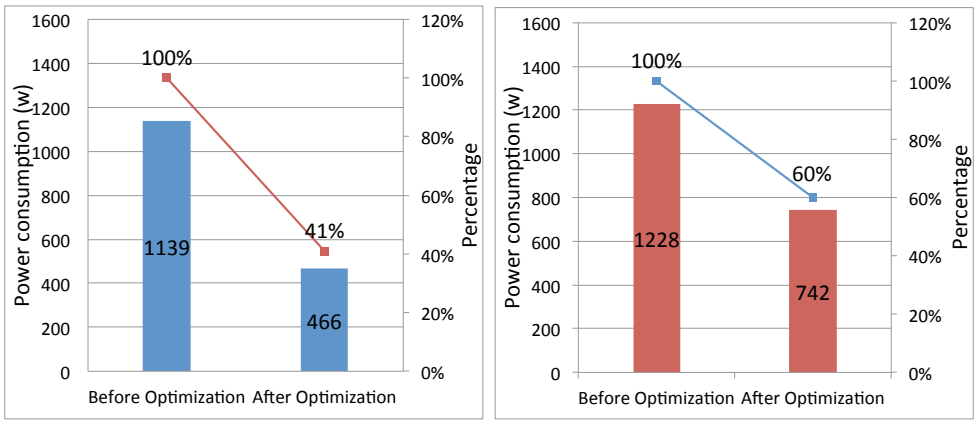


Figure 6.5: Measured incoming and outgoing traffic of Host 4.

on Host 4, the flow does not account any of Host 4's traffic. One example is the flow from VM 7 to VM 19 after the optimization. The total outgoing traffic and total incoming traffic of Host 4 should be the summation of each flow's outgoing traffic amount and the summation of each flow's incoming traffic amount, respectively.

Figure 6.5 shows the measured total outgoing and the measured total incoming traffic of Host 4 before and after the optimization. We can find that, before the optimization and the VM migrations, the amount of outgoing traffic and the amount of incoming traffic of Host 4 are the same as the calculated amount shown in Table 6.2(a). During the optimization, Host 4's incoming traffic amount changes dramatically. This is due to the VM migration traffics that has Host 4 as the destination. After the optimization, the total outgoing traffic of Host 4 stabilizes around 225 Mbps which has less than 2% difference as the calculated amount shown in Table 6.2(b). The incoming traffic amount of Host 4 after the optimization drops to zero which is the same as calculated in Table 6.2(b). In summary, by comparing the measured traffic amount with the calculated traffic amount, we show that the optimization can adjust the flow's routing paths correctly.



(a) Memory and traffic load = 15%. (b) Memory and traffic load = 30%.

Figure 6.6: Power consumption comparison before and after optimization.

Experiment Results - Power Consumption Reduction

Figure 6.6(a) and 6.6(b) illustrates the comparison of the power consumptions before and after the optimization when system load is equal to 15% and 30%, respectively. Both figures show that the power consumption after the optimization is lower than before the optimization. Specifically, Figure 6.6(a) shows that when system load is equal to 15%, the total power consumption drops from 1139W to 466W or from 100% to 41%. In other words, the optimization saves 59% of the original power consumption. Figure 6.6(b) shows that when system load is equal to 30%, the total power consumption drops from 1228W to 742W or from 100% to 60%. In other words, the optimization saves 40% of the original power consumption. It is worth noting that the optimization yields larger reduction of the power consumption when system load is lighter. This is because that, when the system load is lighter, VMs will be consolidated into fewer number of hosts and thus more idle hosts can be powered off to further reduce the power consumption. The experiment results have demonstrated that our joint host-network optimization is effective and practical in improving the data center's energy efficiency.

6.6 Summary

A host-network co-optimization scheme is presented to improve the energy efficiency of data center networks. A unified representation method is first used to convert a virtual machine placement problem to a routing problem, and then a parallelizing approach is used to divide the DCN into clusters, which can be solved in parallel for fast completion. A fast topology-aware multipath routing algorithm is also presented for quick path finding. Finally a 4-pod and 16-host prototype is described to evaluate performance of our design through extensive experiments. Results illustrate that our design superior over existing host- or network-only optimization solutions, and it is ideally suitable for improving the energy efficiency of DCNs.

CHAPTER 7

CONCLUSIONS AND FUTURE WORKS

7.1 Conclusions

In this dissertation, we presented a comprehensive optimization scheme, including optimizations on both host and network sides, to enhance the performance and energy efficiency of the data center network (DCN). The scheme consists of a parallel packet switch architecture to cost-efficiently increase DCN's bandwidth, a multidimensional virtual machine (VM) placement solution to improve the DCN's resource utilization, a flow level bandwidth provisioning algorithm to enable traffic isolation and performance guarantees for VMs, and a energy efficiency optimization to save DCN energy consumption during off-peak traffic hours. We also conducted theoretical analysis as well as empirical simulations and prototype experiments to demonstrate the feasibility, scalability and effectiveness of the scheme. The major contributions made by this dissertation are summarized as follows.

7.1.1 Utilizing Parallel Packet Switch Architecture to Increase DCN's Bandwidth Capacity

To effectively and cost-efficiently increase the DCN's network capacity , we proposed a variable length PPS (vPPS) architecture which combines several lower-speed switches to provide huge aggregate bandwidth and is able to process variable length packet directly. We studied a simplified 1×1 vPPS which is similar to the traditional inverse multiplexing system. We showed that two additional buffers, namely the input conversion buffer (ICB) and the output conversion buffer (OCB), are required to accommodate the rate difference between the input/output line and the center stage switch (CSS). We designed two different scheduling policies to limit the size of ICB and OCB for the simplified 1×1 vPPS, respectively. We showed that both ICB

size and OCB size can be bounded by $2L$, where L is the maximum packet length. Moreover, we proved that the second policy enables the switch to emulate an FIFO OQ switch. We investigated the general $N \times N$ vPPS by expanding the 1×1 switch structure and combing its two scheduling policies. We designed a scheduling policy based on the policies from the simplified 1×1 vPPS case to limit the size of ICB and OCB, respectively. We proved that the presented vPPS switch architecture with the proposed scheduling policy can emulate an FIFO OQ switch with speedup of 2, i.e. emulating an FIFO OQ switch with bandwidth R by using $2K - 1$ CSSs each with bandwidth r, where $r = R/K$.

7.1.2 Multidimensional Stochastic VM Placement to Improve DCN's Resource Utilization

To improve the DCN's resource utilization when VMs demanding for various deterministic and stochastic resources, we proposed a multidimensional stochastic VM placement scheme. We modeled the VM placement in data centers as a Multidimensional Stochastic VM Placement (MSVP) problem, with the objective to minimize the number of required servers while satisfy the service level agreement (SLA) availability guarantee. We proved that this problem is NP-hard. We proposed a polynomial time algorithm named Max-Min Multidimensional Stochastic Bin Packing (M^3SBP) to solve this problem. The basic idea is to maximize the minimum utilization ratio of all the resources of a server, while satisfying the VMs' demands for both deterministic and stochastic resources. We demonstrated by simulations that that M^3SBP guarantees the availability requirement for the stochastic resource while employing fewer servers than other benchmark algorithms do. We also showed that, compared to the modified deterministic algorithms that simply do over-provisioning for stochastic resources, M^3SBP finds the results more efficiently.

7.1.3 Enabling Performance Guarantees on Flow Level

To provide the fine-grained performance assurance in the DCN, we proposed a flow level traffic scheduling technique to provision bandwidth for each individual flow. We proposed the Flow-level Bandwidth Provisioning (FBP) algorithm, which assures the provisioned bandwidth and thus delay guarantees for each individual flows in the DCN. We analyzed the performance of FBP, and prove that it achieves constant service guarantees and tight delay guarantees. We proved that FBP is economical to implement with bounded crosspoint buffer sizes and no speedup requirement, and is fast with low time complexity and distributed scheduling. We implemented FBP in the OpenFlow software switch and integrate it with the NOX controller. We validated the constant service guarantees and tight delay guarantees by the empirical simulation results. We demonstrated by prototype experiment results that our prototype can accurately provision bandwidth at the flow level and is practical to implement in the DCN.

7.1.4 Host-Network Co-Optimization to Enhance DCN's Energy Efficiency

To improve the energy efficiency of the DCN during off-peak traffic time, we investigated the the optimization scheme which reduces the number of active devices in the network while maintaining the required network services. We proposed a host-network energy efficiency co-optimization scheme for DCN that combines VM placement and flow routing optimization, so that the energy efficiency can be improved on both sides. We developed an unified representation method which transforms the VM placement problem to adapt the flow routing problem. We developed a topology-aware recursive multi-path routing algorithm which utilizes the depth first search algorithm to traverse the hierarchies of the DCN, and utilizes the best fit to find the most proper flow routing path. We introduced a parallel process-

ing approach which divides the target data center into clusters and optimizes the clusters simultaneously. We built an OpenFlow hardware switch based prototype based on the proposed optimization scheme. We demonstrated the effectiveness and practicalness of the proposed optimization scheme by empirical experiment results.

7.2 Future Directions

One future research direction is to improve the vPPS architecture so that it can emulate a Push-In-First-Out (PIFO) OQ switch without speedup. The proposed vPPS in this dissertation can only emulates FIFO OQ switch, in which the packet output order is the same as that when the packet arrives at the switch. In contrast, in PIFO OQ switches, the arrival order and output order of each packet can be different. In other words, PIFO OQ switches allows the implementations of various QoS packet scheduling disciplines such as General Processor Sharing (GPS) and Weighted Fair Queueing (WFQ). The potential modification of the proposed vPPS in this dissertation is to add packet buffers at the demultiplexers and multiplexers. As a result, packets in the vPPS can be stored in the added buffers for extra time until the their output time required by the scheduling discipline comes.

Another research direction is to investigate the VM placement in DCNs with correlations between different resource demands. In this dissertation, we assume that all resource demands are independent from each other. However, in some situations, the change of one resource demand may affect other demands. For example, [CG05] shows that there is a strong, near-linear relationship between the network I/O speed and the CPU utilization. Modeling and considering such correlation in the VM placement calculation can further improve the DCN's resource utilization.

The performance evaluation method of the flow level performance guarantee is also worth future studies. This dissertation only evaluates FBP on an Openflow

software switch based prototype which runs on Linux user space. To evaluate F-BP's performance in a more realistic environment, we can implement FBP on a Network FPGA based hardware Openflow switch prototype. Network FPGA platform not only supports the Openflow protocol, but more importantly, it allows the implementation of the CICQ switch architecture. In addition, the entire Openflow switch can work in Linux kernel space which is greatly faster than in the user space. As a result, the performance evaluation will be more realistic and results will be more persuasive.

BIBLIOGRAPHY

[AC04] A. Aslam and K. J. Christensen. A parallel packet switch with multiplexors containing virtual input queues. *Computer Communications*, 13(13):1248–1263, Aug. 2004.

[AFLV08] M. Al-Fares, A. Loukissas, and A. Vahdat. A scalable, commodity data center network architecture. In *ACM SIGCOMM*, Seattle, WA, Aug. 2008.

[AfRR+10] M. Al-fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat. Hedera: dynamic flow scheduling for data center networks. In *USENIX NSDI*, San Josa, CA, Apr. 2010.

[AHK10] H. Attiya, D. Hay, and I. Keslassy. Packet-mode emulation of output-queued switches. *IEEE Transactions on Computers*, 59(10):1378–1391, Oct. 2010.

[AMW+10] D. Abts, M. Marty, P. Wells, P. Klausler, and H. Liu. Energy proportional datacenter networks. In *ACM/IEEE ISCA*, Saint-Malo, France, Jun. 2010.

[BAM10] T. Benson, A. Akella, and D. A. Maltz. Network traffic characteristics of data centers in the wild. In *ACM IMC*, Melbourne, Australia, Nov. 2010.

[BKB07] N. Bobroff, A. Kochut, and K. Beaty. Dynamic placement of virtual machines for managing sla violations. In *IFIP/IEEE IM*, Munich, Germany, May 2007.

[BZ96] J. Bennett and H. Zhang. Wf2q: worst-case fair weighted fair queueing. In *IEEE INFOCOM*, San Francisco, CA, Mar 1996.

[CFH+05] C. Clark, K. Fraser, S. Hand, F. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield. Live migration of virtual machines. In *USENIX NSDI*, Berkeley, CA, Apr. 2005.

[CFP+07] M. Casado, M. J. Freedman, J. Pettit, J. Luo, N. Mckeown, and S. Shenker. Ethane: taking control of the enterprise. In *ACM SIGCOMM*, Kyoto, Japan, Aug. 2007.

[CG05] L. Cherkasova and R. Gardner. Measuring cpu overhead for i/o processing in the xen virtual machine monitor. In *USENIX ATEC*, Anaheim, CA, Apr. 2005.

[CGJ97] E. G. Coffman, Jr., M. R. Garey, and D. S. Johnson. *Approximation algorithms for NP-hard problems*. PWS Publishing Co., 1997.

[CGMP99] S. Chuang, A. Goel, N. McKeown, and B. Prabhkar. Matching output queueing with a combined input output queued switch. In *IEEE INFOCOM*, New York, NY, Mar. 1999.

[CIM05] S. Chuang, S. Iyer, and N. McKeown. Practical algorithms for performance guarantees in buffered crossbars. In *IEEE INFOCOM*, Miami, FL, Mar. 2005.

[CJGMV98] E. G. Coffman Jr., G. Galambos, S. Martello, and D. Vigo. Bin packing approximation algorithm: combinatiorial analysis. *Handbook of Combinatorial Optimization*, 1998.

[CMT+11] A. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, and S. Banerjee. Devoflow: Scaling flow management for high-performance networks. In *ACM SIGCOMM*, Toronto, ON, Canada, Aug. 2011.

[CZS+11] M. Chen, H. Zhang, Y. Y. Su, X. Wang, G. Jiang, and K. Yoshihira. Effective vm sizing in virtualized data centers. In *IFIP/IEEE IM*, Dublin, Ireland, May 2011.

[dat] U.s. environmental protection agencys data center report to congress. `http://www.energystar.gov/ia/partners/prod_development/downloads/EPA_Datacenter_Report_Congress_Final1.pdf`.

[DKS89] A. Demers, S. Keshav, and S. Shenker. Analysis and simulation of a fair queueing algorithm. In *ACM SIGCOMM*, Austin, TX, Sep. 1989.

[flo] Flowvisor. `http://www.openflowswitch.org/wk/index.php/FlowVisor`.

[FSR10] W. Fisher, M. Suchara, and J. Rexford. Greening backbone networks: reducing energy consumption by shutting off cables in bundled links. In

ACM SIGCOMM Workshop on Green Networking, New Delhi, India, Aug. 2010.

[GET] Man page for gettimeofday. `http://www.kernel.org/doc/man-pages/online/pages/man2/gettimeofday.2.html`.

[GHJ+09] A. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. A. Maltz, P. Patel, and S. Sengupta. Vl2: a scalable and flexible data center network. In *ACM SIGCOMM*, Barcelona, Spain, 2009.

[GI99] A. Goel and P. Indyk. Stochastic load balancing and related problems. In *IEEE FOCS*, New York City, NY, Oct. 1999.

[GKP+08] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, and S. Shenker. Nox: towards an operating system for networks. *ACM SIGCOMM Computer Communication Review*, 38(3):105–110, Jul. 2008.

[GLF+00] D. Grunwald, P. Levis, K. I. Farkas, C. B. Morrey, III, and M. Neufeld. Policies for dynamic clock scheduling. In *USENIX OSDI*, San Diego, CA, Oct. 2000.

[GLL+09] C. Guo, G. Lu, D. Li, H. Wu, X. Zhang, Y. Shi, C. Tian, Y. Zhang, and S. Lu. Bcube: a high performance, server-centric network architecture for modular data centers. In *ACM SIGCOMM*, Barcelona, Spain, Aug. 2009.

[GLM+08] A. Greenberg, P. Lahiri, D. A. Maltz, P. Patel, and S. Sengupta. Towards a next generation data center architecture: scalability and commoditization. In *ACM SIGCOMM PRESTO*, Seattle, WA, Aug. 2008.

[GWT+08] C. Guo, H. Wu, K. Tan, L. Shi, Y. Zhang, and S. Lu. Dcell: a scalable and fault-tolerant network structure for data centers. In *ACM SIGCOMM*, Seattle, WA, Aug. 2008.

[GZH+11] A. Ghodsi, M. Zaharia, B. Hindman, A. Konwinski, S. Shenker, and I. Stoica. Dominant resource fairness: fair allocation of multiple resource types. In *USENIX NSDI*, Boston, MA, Mar. 2011.

[HMGW08] C. Hyser, B. Mckee, R. Gardner, and B. J. Watson. Autonomic virtual machine placement in the data center. Technical report, HP Technical Report HPL-2007-189, Feb. 2008.

[HS08] M. Hosaagrahara and H. Sethu. Max-min fairness in input-queued switches. *IEEE Transactions on Parallel and Distributed Systems*, 19(4):462–475, Apr. 2008.

[HSG+08] S. He, S. Sun, H. Guan, Q. Zheng, Y. Zhao, and W. Gao. On guaranteed smooth switching for buffered crossbar switches. *IEEE/ACM Transactions on Networking*, 16(3):718–731, Jun. 2008.

[HSM+10] B. Heller, S. Seetharaman, P. Mahadevan, Y. Yiakoumis, P. Sharma, S. Banerjee, and N. McKeown. Elastictree: saving energy in data center networks. In *USENIX NSDI*, Berkeley, CA, USA, 2010.

[htb] Hierarchical token bucket theory. `http://luxik.cdi.cz/~devik/qos/htb/manual/theory.htm`.

[IAM00] S. Iyer, A. Awadallah, and N. McKeown. Analysis of a packet switch with memories running slower than the line-rate. In *IEEE INFOCOM*, Tel Aviv, Israel, Mar. 2000.

[IM03] S. Iyer and N. McKeown. Analysis of the parallel packet switch architecture. *IEEE/ACM Transactions on Networking*, 11(2):314–324, Apr. 2003.

[ipe] Iperf: the tcp/udp bandwidth measurement tool. `http://sourceforge.net/projects/iperf/`.

[JCL+ed] H. Jin, T. Cheocherngngarn, D. Levy, A. Smith, D. Pan, J. Liu, and N. Pissinou. Joint host-network optimization for energy-efficient data center networking. In *IEEE IPDPS*, Boston, MA, May 2013 (Submitted).

[JPLP12] H. Jin, D. Pan, J. Liu, and N. Pissinou. Openflow based flow-level bandwidth provisioning for cicq switches. *IEEE Transactions on Computers*, 2012.

[JPP11] H. Jin, D. Pan, and N. Pissinou. Parallel packet switch without segmentation-and-reassembly. In *IEEE Global Communications Conference*, Houston, TX, Dec. 2011.

[JPXP12] H. Jin, D. Pan, J. Xu, and N. Pissinou. Efficient vm placement with multiple deterministic and stochastic resources in data centers. In *IEEE GLOBECOM*, Anaheim, CA, Dec. 2012.

[KAGS11] B. Krishnan, H. Amur, A. Gavrilovska, and K. Schwan. Vm power metering: feasibility and challenges. *ACM SIGMETRICS Performance Evaluation Review*, 38:56–60, Apr. 2011.

[Kat09] R. H. Katz. Tech titans building boom. *IEEE Spectrum*, 46(2):40–54, 2009.

[KHK09] Y. Kanizo, D. Hay, and I. Keslassy. The crosspoint-queued switch. In *IEEE INFOCOM*, Rio de Janeiro, Brazil, Apr. 2009.

[KK01] D. A. Khotimsky and S. Krishnan. Stability analysis of a parallel packet switch with bufferless input demultiplexors. In *IEEE ICC*, helsinki, Finland, Jun. 2001.

[Kor06] G. Kornaros. Bcb: a buffered crossbar switch fabric utilizing shared memory. In *EUROMICRO*, Cavtat/Dubrovnik, Croatia, Aug. 2006.

[KRT00] J. Keinber, Y. Rabani, and E. Tardos. Allocating bandwidth for bursty connections. *SIAM Journal on Computing*, 30:191–217, Apr. 2000.

[KSG+09] S. Kandula, S. Sengupta, A. Greenberg, P. Patel, and R. Chaiken. The nature of datacenter traffic: measurements and analysis. In *ACM IMC*, Chicago, IL, Nov. 2009.

[KZLK10] A. Kansal, F. Zhao, J. Liu, and N. Kothari. Virtual machine power metering and provisioning. In *ACM SOCC*, Indianapolis, IN, Jun. 2010.

[LK10] B. Lin and I. Keslassy. The concurrent matching switch architecture. *IEEE/ACM Transactions on Networking*, 18(4):1330–1343, Aug. 2010.

[LS06] H.-I. Lee and S.-W. Seo. Matching output queueing with a multiple input/output-queued switch. *IEEE/ACM Transactions on Networking*, 14(1):121–132, Feb. 2006.

[McK99] N. McKeown. The islip scheduling algorithm for input-queued switches. *IEEE/ACM Transactions on Networking*, 7(2):188–201, Apr. 1999.

[MGW09] D. Meisner, B. T. Gold, and T. F. Wenisch. Powernap: eliminating server idle power. In *ASPLOS*, Washington, DC, Mar. 2009.

[MH03] L. Mhamdi and M. Hamdi. Output queued switch emulation by a one-cell-internally buffered crossbar switch. In *IEEE GLOBECOM*, San Francisco, CA, Dec. 2003.

[MPF+09] R. N. Mysore, A. Pamboris, N. Farrington, N. Huang, P. Miri, S. Radhakrishnan, V. Subramanya, and A. Vahdat. Portland: a scalable fault-tolerant layer 2 data center network fabric. In *ACM SIGCOMM*, pages 39–50, New York, NY, USA, 2009.

[MPZ10] X. Meng, V. Pappas, and L. Zhang. Improving the scalability of data center networks with traffic-aware virtual machine placement. In *IEEE INFOCOM*, San Diego, CA, Mar. 2010.

[MRS03] B. Magill, C. Rohrs, and R. Stevenson. Output-queued switch emulation by fabrics with limited memory. *IEEE Journal on Selected Areas in Communications*, 21(4):606–615, May 2003.

[MSA+06] N. Mckeown, S. Shenker, T. Anderson, L. Peterson, J. Turner, H. Balakrishnan, and J. Rexford. Openflow: enabling innovation in campus networks. *ACM SIGCOMM Computer Communication Review*, 38(2):69–74, Apr. 2006.

[MSS02] S. Mneimneh, V. Sharma, and K.-Y Siu. Switching using parallel input-output queued switches with no speedup. *IEEE/ACM Transactions on Networking*, 10(5):653–665, Oct. 2002.

[MY10] J. Mudigonda and P. Yalagandula. Spain: Cots data-center ethernet for multipathing over arbitrary topologies. In *USENIX NSDI*, San Josa, CA, Apr. 2010.

[nox] Nox: an openflow controller. `http://www.noxrepo.org`.

[NUM] Who has the most web servers? `http://www.datacenterknowledge.com/archives/2009/10/13/facebook-now-has-30000-servers/`.

[opea] Geni openflow backbone deployment at internet2. `http://groups.geni.net/geni/wiki/OFI2`.

[opeb] Openflow 1.0 release. `http://www.openflowswitch.org/wk/index.php/OpenFlow_v1.0`.

[OPEc] Openflow switch specification version 1.0.0. http://www.openflow.org/documents/openflow-spec-v1.0.0.pdf.

[PBS+03] C. Patel, C. Bash, R. Sharma, M. Beitelmam, and R. Friedrich. Smart cooling of data centers. In *InterPack*, Maui, HI, Jul. 2003.

[PG93] A. Parekh and R. Gallager. A generalized processor sharing approach to flow control in integrated services networks: the single node case. *IEEE/ACM Transactions on Networking*, 1(3):344–357, Jun. 1993.

[PKC07] I. Papaefstathiou, G. Kornaros, and N. ChrysosUsing. Buffered crossbars for chip interconnection. In *Great Lakes Symposium on VLSI*, Stresa-Lago Maggiore, Italy, Mar. 2007.

[PY07] D. Pan and Y. Yang. Max-min fair bandwidth allocation algorithms for packet switches. In *IEEE IPDPS*, Long Beach, CA, Mar. 2007.

[PY08] D. Pan and Y. Yang. Providing flow based performance guarantees for buffered crossbar switches. In *IEEE IPDPS*, Miami, FL, Apr. 2008.

[PY09] D. Pan and Y. Yang. Localized independent packet scheduling for buffered crossbar switches. *IEEE Transactions on Computers*, 58(2):260–274, Feb. 2009.

[rsv] Resource reservation protocol (rsvp) – version 1 functional specification. http://www.ietf.org/rfc/rfc2205.txt.

[sli] Openflow slicing. http://www.openflowswitch.org/wk/index.php/Slicing.

[SLX10] Y. Shang, D. Li, and M. Xu. Energy-aware routing in data center network. In *ACM SIGCOMM Workshop on Green Networking*, New Delhi, India, Aug. 2010.

[SMLF09] B. Sotomayor, R. S. Montero, I. M. Llorente, and I. Foster. Virtual infrastructure management in private and hybrid clouds. *IEEE Internet Computing*, 13(5):14–22, 2009.

[SPK08] D. Simos, I. Papaefstathiou, and M. Katevenis. Building an foc using large, buffered crossbar cores. *IEEE Design & Test of Computers*, 25(6):538–548, Nov. 2008.

[SV96] M. Shreedhar and G. Varghese. Efficient fair queuing using deficit round robin. *IEEE/ACM Transactions on Networking*, 4(3):375–385, Jun. 1996.

[SXL07] L. Shi, G. Xia, and B. Liu. Performance guarantees for flow-mapping parallel packet pwitch. In *IEEE IPCCC*, New Orleans, LA, Apr. 2007.

[SZ98] I. Stoica and H. Zhang. Exact emulation of an output queueing switch by a combined input output queueing switch. In *IEEE/IFIP IWQoS*, San Francisco, CA, May 1998.

[Tur09] J. Turner. Strong performance guarantees for asynchronous crossbar schedulers. *IEEE/ACM Transactions on Networking*, 17(4):1017–1028, Aug. 2009.

[Val07] P. Valente. Exact gps simulation with logarithmic complexity, and its application to an optimally fair scheduler. *IEEE/ACM Transactions on Networking*, 15(6):1454–1466, Dec. 2007.

[VEP] Edge virtual bridge proposal. `http://ieee802.org/1/files/public/docs2008/new-congdon-vepa-1108-v01.pdf`.

[VIR] Virtualbox virtual networking. `http://www.virtualbox.org/manual/ch06.html`.

[VT09] H. N. Van and F. D. Tran. Autonomic virtual resource management for service hosting platforms. In *IEEE CLOUD*, Vancouver, Canada, May 2009.

[WMZ11] M. Wang, X. Meng, and L. Zhang. Consolidating virtual machines with dynamic bandwidth demand in data centers. In *IEEE INFOCOM*, Shanghai, China, Apr. 2011.

[WYHL09] B. Wu, K. Yeung, M. Hamdi, and X. Li. Minimizing internal speedup for performance guaranteed switches with optical fabrics. *IEEE/ACM Transactions on Networking*, 17(2):632–645, Apr. 2009.

[XF10] J. Xu and J. A. B. Fortes. Multi-objective virtual machine placement in virtualized data center environments. In *IEEE GREENCOM*, Hangzhou, China, Dec. 2010.

[XF11] J. Xu and J. A. B. Fortes. A multi-objective approach to virtual machine management in datacenters. In *ACM ICAC*, Karlsruhe, Germany, Jun. 2011.

[YRFW10] M. Yu, J. Rexford, M. J. Freedman, and J. Wang. Scalable flow-based networking with difane. In *ACM SIGCOMM*, New Delhi, India, Aug. 2010.

[ZMB07] X. Zhang, S. Mohanty, and L. Bhuyan. Adaptive max-min fair scheduling in buffered crossbar switches without speedup. In *IEEE INFOCOM*, Anchorage, AK, May 2007.

[ZXZ05] H. Zhong, D. Xu, and Z. Zhu. A parallel packet switch supporting variable-length packets. In *IEEE ICCCAS*, Hong Kong, China, May 2005.

[ZYLZ10] M. Zhang, C. Yi, B. Liu, and B. Zhang. Greente: Power-aware traffic engineering. In *IEEE ICNP*, Koyoto, Japan, Oct. 2010.

VITA

HAO JIN

2006	B.S., Electrical Engineering Nanjing University Nanjing, China
2012	Doctoral Candidate, Electrical Engineering Florida International University Miami, Florida

PUBLICATIONS

H. Jin, T. Cheocherngngarn, D. Pan, J. Liu, J. Andrian and N. Pissinou, "Joint Host-Network Optimization for Energy-Efficient Data Center Networking," *IEEE International Parallel & Distributed Processing Symposium*, submitted.

H. Jin, D. Pan, J. Xu and N. Pissinou, "Efficient VM Placement with Multiple Deterministic and Stochastic Resources in Data Centers," *IEEE Global Communications Conference* , Anaheim, CA, Dec. 2012.

T. Cheocherngngarn, H. Jin, J. Andrian, D. Pan and J. Liu, "Depth-first Worst-fit Search based Multipath Routing for Data Center Networks," *IEEE Global Communications Conference* , Anaheim, CA, Dec. 2012.

H. Jin, D. Pan, J. Liu, and N. Pissinou, "OpenFlow based flow level bandwidth provisioning for CICQ switches," *IEEE Transactions on Computers*, accepted for publication.

H. Jin, D. Pan and N. Pissinou, "Parallel Packet Switch without Segmentation-and-Reassembly," *IEEE Global Communications Conference*, Houston, TX, Dec. 2011.

S. Jiang, S. Georgekopoulos and H. Jin, "Effects of Periodic Reinforced-Concrete

Structures on Power Transmission," *IEEE International Conference on RFID*, Orlando, FL, Apr. 2012.

H. Jin, D. Pan, J. Liu, and N. Pissinou, "OpenFlow based flow level bandwidth provisioning for CICQ switches," *IEEE International Conference on Computer Communications*, Shanghai, China, Apr. 2011.

H. Jin, D. Pan, N. Pissinou, and K. Makki, "Achieving flow level constant performance guarantees for CICQ switches without speedup," *IEEE Global Communications Conference*, Miami, FL, Dec. 2010.