On the Design of High Performance Data Center Networks

A Dissertation Presented

by

Zhiyang Guo

to

The Graduate School

in Partial Fulfillment of the Requirements

for the Degree of

Doctor of Philosophy

in

Electrical Engineering

Stony Brook University

May 2014

Stony Brook University

The Graduate School

Zhiyang Guo

We, the dissertation committee for the above candidate for the

Doctor of Philosophy degree, hereby recommend

acceptance of this dissertation

Yuanyuan Yang - Dissertation Advisor
Professor, Department of Electrical and Computer Engineering

Sangjin Hong - Chairperson of Defense
Associate Professor, Department of Electrical and Computer Engineering

Dmitri Donetski
Associate Professor, Department of Electrical and Computer Engineering

Esther M. Arkin
Professor, Department of Applied Mathematics and Statistics

This dissertation is accepted by the Graduate School

Charles Taber
Dean of the Graduate School

Abstract of the Dissertation

On the Design of High Performance Data Center Networks

by

Zhiyang Guo

Doctor of Philosophy

in

Electrical Engineering

Stony Brook University

2014

Massive modern data centers consisting of tens of thousands of servers have emerged to form the backbone of a variety of powerful distributed computing frameworks, in which efficient communication is often required among huge data sets stored in tens of thousands of servers across a data center. This renders the performance of the data center network (DCN) essential to the successful operation of a data center.

This dissertation focuses on several important issues in the design space of high performance data center networks. First, we present a systematic study on optical packet switch, which is a key component in next generation DCNs. The study can be divided into two parts. In the first part, we present an efficient analytical model for a novel hybrid optical/electrical packet switch called OpCut. In the second part, we study the multicast scheduling problem in all-optical packet switches. We propose a novel optical buffer structure, a Low Latency Multicast

Scheduling (LLMS) Algorithm that guarantees delay upper bound, and a pipeline and parallel architecture that enables line-rate scheduling.

Second, we study how to deploy high performance multicast communication in data center networks. Multicast can tremendously benefit many cloud-based services that require one-to-many group communication, such as redirecting search queries to multiple indexing servers and replicating file chunks in distributed file systems, through releasing the sender from duplicated transmission tasks, thus, significantly improving network latency. Exploring the unique novel features and techniques in data centers, our research focuses on the following issues: (1) Achieving cost-efficient provisioning of nonblocking multicast fat-tree DCNs by exploring server redundancy in data centers; (2) Developing a practical multicast flow scheduling algorithm that ensures guaranteed flow bandwidth and high network throughput under volatile data center traffic.

Third, we study the recently developed hybrid packet/circuit (Hypac) switched DCN architecture, which arguments the traditional electrical packet switched (EPS) network with a high-speed optical circuit switched (OCS) network. Considering that the OCS/EPS networks have unique strengths and weaknesses, we propose a time-efficient Collaborative Bandwidth Allocation (CBA) algorithm that configures both networks in a complementary manner. Also, we show that given sufficient bandwidth from both networks, a Hypac DCN can *guarantee* 100% throughput with a bounded delay using the proposed CBA algorithm. Through comprehensive evaluations, we demonstrate CBA significantly improves the performance of Hypac DCNs in many aspects.

In summary, this dissertation combines algorithm design, mathematical modeling, network optimization, theoretical analysis and simulation techniques to provide a thorough investigation on the above issues. The outcome of this research would benefit many cloud applications that rely on group communication, and have a significant impact on the fundamental design principles for future DCNs.

Contents

List of Figures

List of Tables

Acknowledgements

I am deeply appreciative of the many individuals who have supported my work and continually encouraged me through my Ph.D. journey. Without their encouragement, thoughtful feedback, and patience, I would not have been able to see it through.

First and foremost, I would like to express my sincere gratitude to my advisor, Prof. Yuanyuan Yang, for her continuous support of my Ph.D study and research. She patiently provided the vision, encouragement and advise necessary for me to proceed through the doctorial program and complete my dissertation. I have learned a great deal from her unique perspective on research, her sharp insight, and her personal integrity and expectations of excellence. I could not have imagined having a better advisor and mentor for my Ph.D. study.

Besides my advisor, I would like to thank the rest of my dissertation committee: Prof. Dimitri Donetski, Prof. Esther M. Arkin, and Prof. Sangjin Hong, for their encouragement and insightful comments. I also would like to thank the individuals in the ECE department who have made my stay at Stony Brook pleasant: Rachel Ingrassia, Alex Harris and Anthony Olivo.

I am grateful towards my many student colleagues for all the fun and memory we had together during the past six years. Many thanks go to my lab mates: Dawei Gong, Ji Li, Cong Wang, Zhenhua Li, Jun Duan, Hao Li and Zheming Zhang, for their helpful discussion and the countless hours we spent together in the lab. I am also thankful for the companionship of my dear friends: Xiaoying Wang, Cunyou Lv, Jinghui Jian, Zhichao Chen and Shuo Li. Without their company, my Ph.D. journey would be much less colorful and fun, and for that I owe them my heartfelt appreciation. Special thanks also go to my host family, Mr. Ernest Edricks and Mrs. Carolyn Edricks. Their kind hospitality makes me feel that I am never too far away from home.

Finally, I wish to dedicate this dissertation to my parents, Guangxiu Pei and Haiyan Guo, and my wife, Xiao Fu, for their unconditional love and support. Their encouragement was in the end what made this dissertation possible, and I would like to express my deepest gratitude and love for their dedication and the many years of support.

List of Publications

Journal Publications and Book Chapters

- Z. Guo, J. Duan and Y. Yang, “On-line Multicast Scheduling with Bounded Congestion in Fat-tree Data Center Networks,” IEEE Journal on Selected Areas in Communications (J-SAC), vol. 32, no. 1, Jan. 2014.
- Z. Guo and Y. Yang, “On Nonblocking Multirate Multicast Fat-tree Data Center Networks with Server Redundancy,” to appear in IEEE Transactions on Computers (TC)
- Z. Zhang, Z. Guo and Y. Yang, “Bufferless Routing in Optical Gaussian Macrochip Interconnect,” to appear in IEEE Transactions on Computers (TC)
- Z. Guo and Y. Yang, “High Speed Multicast Scheduling in Hybrid Optical Packet Switches with Guaranteed Latency,” IEEE Transactions on Computers. (TC), vol.62, no.10, pp.1972-1987, Oct. 2013
- Z. Zhang, Z. Guo and Y. Yang, “Efficient All-to-All Broadcast in Gaussian On-Chip-Networks,” IEEE Transactions on Computers (TC), vol.62, no.10, pp.1959-1971, Oct. 2013
- Z. Guo and Yuanyuan Yang, “Low-Latency Multicast Scheduling Algorithm for All-Optical Interconnects,” to appear in IEEE Transactions on Communications (TCOM)
- Z. Guo and Y. Yang, “Exploring Server Redundancy in Nonblocking Multicast Data Center Networks,” to appear in IEEE Transactions on Computers (TC).

- Z. Guo and Y. Yang, "Multicast Communication in Data Center Networks," in Data Center Handbook, Springer, to appear in 2014.

Conference Publications

- Z. Guo and Y. Yang, " Collaborative Network Configuration in Hybrid Electrical/optical Data Center Networks," IEEE IPDPS, 2014
- Z. Guo, J. Duan and Y. Yang, "Oversubscription Bounded Multicast Scheduling in Fat-tree Data Center Networks," IEEE IPDPS, 2013
- Z. Guo and Y. Yang, "High-Speed Multicast Scheduling for All-Optical Packet Switches" IEEE NAS, 2013
- Z. Guo and Y. Yang, "Multicast Fat-tree Data Center Networks with Bounded Link Oversubscription," IEEE INFOCOM, 2013
- Z. Zhang, Z. Guo and Y. Yang, "Bounded-Reorder Packet Scheduling in Optical Cut-through Switch," IEEE INFOCOM, 2013
- Z. Zhang, Z. Guo and Y. Yang, "Bufferless Routing in Optical Gaussian Macrochip Interconnect," IEEE HOTI, 2012
- Z. Guo, Z. Zhang and Y. Yang, "Exploring Server Redundancy in Nonblocking Multicast Data Center Networks," IEEE INFOCOM, 2012
- Z. Guo and Y. Yang, "On Nonblocking Multirate Multicast Fat-tree Data Center Networks with Server Redundancy," IEEE IPDPS, 2012
- Z. Guo and Y. Yang, "Pipelining Multicast Scheduling in All-Optical Packet Switches with Delay Guarantee," IEEE ITC, 2011
- Z. Guo, Z. Zhang and Y. Yang, "Performance Modeling of Hybrid Optical Packet Switches with Shared Buffer," IEEE INFOCOM, 2011
- Z. Guo, X. Luo and Y. Jin, "Improving Network Resource Utilization in Hybrid Packet/Circuit Multicasting for IPTV Delivery," OFC/NFOEC, 2008
- Z. Guo and Y. Yang, "Achieving Performance Guarantee via Joint Traffic Scheduling in Hybrid Electrical/Optical Data Center Networks," IEEE HOTI, 2014 (under review)

- Z. Guo and Y. Yang, “Augmenting Data Center Network with A Fast Reconfigurable Optical Multistage Interconnect,” IEEE Globecom, 2014 (under review)
- Z. Li, Z. Guo and Y. Yang, “BCube Connected Crossbars(BCCC): An Expandable Network for Data Centers Using Dual Port Commodity Servers,” ACM/IEEE ANCS, 2014 (under review)
- J. Duan, Z. Guo and Y. Yang,“Cost Efficient and Performance Guaranteed Virtual Network Embedding in Fat-Tree Data Center Networks, ACM/IEEE ANCS, 2014 (under review)

Chapter 1

Introduction and Related Work

This chapter explains the background, related work, design goals and contributions of the dissertation.

1.1 Data Center Networks in Cloud Computing

Massive modern data centers consisting of tens of thousands of servers, such as Microsoft's Azure platform [1], Google's App engine and Amazon's EC2 platform [2], have emerged to form the backbone of cloud computing, which has transformed the IT industry in a profound way. Cloud computing [3, 4] fulfills the long held dream of computing as a utility, which relieves developers from the risk of over-provisioning or under-provisioning of their services, as they can dynamically expand or contract their presence according to user demand in a "pay-as-you-go" manner. Moreover, the scalable and elastic computing provided by cloud also allows companies with large batch-oriented tasks to get results as quickly as programs can scale, since using 1000 servers for one hour costs no more than using one server for 1000 hours [3]. As a result, cloud computing drastically increases developer productivity, application performance, and cost-efficiency. Attracted by these appealing features, many companies are moving their services such as e-commerce, scientific computing and social networking to the cloud.

Ideally speaking, a cloud, or data center, should create an illusion that there is an infinite pool of available computing and storage resource, from which tenants can draw from. To achieve this goal, service *agility*- the ability to assign any server to any service [5], of a data center is of paramount importance, as it is a determining factor in many vital metrics of a cloud such as job completion time, server utilization, operation cost, etc. Meanwhile, service agility critically hinges on the performance of the *data center network* (DCN), because network congestion creates partition among servers, which severely restricts the ability of a data center to send workloads to any servers.

To further understand the importance of DCNs, we also need to understand the characteristics of common cloud services and applications. Cloud applications broadly fall into two categories: interactive applications and batch applications. Interactive applications need fast response times to keep users engaged. One example is search, where each query is sent to many servers in a data center, then the responses are sent to a few machines to produce aggregated results, both via a DCN [6]. Batch applications, such as analytic and data mining, though impose less stringent requirement on response time in comparison, often deploy distributed computation frameworks like MapReduce [7], which involve the cooperation of many servers and put high traffic loads on network. Additionally, both types of cloud-hosted applications usually rely on distributed file service, such as Google File System (GFS) [11], which also relies on DCN to provide transparent, scalable access for any server to the data stored at any other server. As efficient communication among servers is integral to most cloud applications and services, the DCN performance is essential to the success of cloud computing.

1.2 Design Goals of DCNs

The cloud computing paradigm imposes a series of stringent requirements on the design of DCNs. Generally speaking, a DCN should be scalable, expandable, manageable and cost-efficient, as well as provides predictably high bandwidth and performance isolation between applications. Additionally, it should be tailored to effectively accommodate common traffic patterns, such as one-to-many traffic and many-to-many traffic, generated by popular cloud applications and services. Next, we illustrate these requirements in detail.

- **Scalability:** Today's data centers contain hundreds of thousands of servers, whereas the most high-end switches have only hundreds of ports [18]. Finding a good network architecture that prevents the DCN from becoming a limiting factor in determining the number of servers deployed in a data center is vitally important.

- **Expendability:** As the scale of data centers is increasing constantly, it is imperative that the network does not impose an obstacle for data center size expansion [20]. Data center operators should be able to add new servers and switches into an existing DCN with little alteration of the overall network structure.

- **Cost-efficiency:** A data center represents a significant investment in capital outlay and ongoing costs, in which the DCN occupies a significant portion [5, 12–14]. The cost of a DCN comes from two parts: the equipment cost, which goes to switches, load balancers, links, power supplies, etc, and operation costs, which mostly incurred by power consumption and management of network equipments. How to achieve cost-efficiency is a primary concern in DCN design, as it is important to the profitability of a data center.

- **Manageability:** The large scale of a DCN makes it difficult for data center operators to manually manage each individual network device. Manual configuration is not only error-prone, but significantly increases the operation costs of a DCN[5]. Hence, an important requirement of the DCN design is good manageability. Ideally, network devices should be able to "plug-and-play" without much human intervention.

- **Predictability and performance isolation:** A data center is usually shared by numerous applications and services that may be owned by different tenants. Such shared nature of data centers leads to very volatile traffic patterns in the DCNs [8–10]. Meanwhile, it means that the network performance experienced by an application can be significantly influenced by other applications [59]. Network performance variability hurts application performance and causes provider revenue loss. It is also a major hindrance to cloud adoption for a wide range of applications that require reliable inter-server communication [52, 59]. Additionally, it forces applications to tailor their workloads to where bandwidth is available, which in turn degrades service agility. Ideally, a DCN should never be the limiting factor in the transmission rate between end-hosts, and tenants should receive guaranteed bandwidth regardless of traffic conditions.

- **Effective support of different traffic patterns:** The most common traffic pattern generated by cloud applications is one-to-one traffic, that is, a server communicates with another server. One-to-one traffic pattern can be effectively supported by unicast, an intrinsic network primitive in DCNs. On the other hand, many critical data center applications, frameworks and services, like distributed file services [11] and MapReduce [7], also generate other traffic patterns, such as one-to-any, one-to-many [31–34] and many-to-many, which cannot be well-supported by unicast. An important design goal of D-CNs is to effectively support these common traffic patterns.

1.3 A Taxonomy for DCN Research

A DCN is a very complex ecosystem with numerous interacting building blocks, therefore, its design space is enormous. This section gives a high-level illustration of the different aspects of current DCN research.

To better illustrate the various design concerns of DCNs, we first compare the DCN research with the Internet research, which has received ongoing attention for decades. The most significant difference between the Internet and DCNs is that a DCN is usually managed by a single entity, whereas there is no single sovereignty for the Internet. The unified environment of DCNs makes it possible to deploy a centralized platform that collectively configures all the network equipments, see, for example, the recent movement towards software defined networking [43–45], which enables more effective management of network resources with global knowledge of the network condition. This clearly contrasts the intrinsically-distributive Internet at large. For the same reason, it is much easier to implement changes spread across several design domains in a DCN architecture, which explains the fast evolution of DCN research in the past few years, whereas modification to even a single layer in the Internet incurs such difficulty that the research of the Internet is said to be "ossified" [50]. Finally, the structure of DCNs usually follow a defined topology such as fat-tree [18], whereas the topology of the Internet is non-deterministic and constantly changing.

Next, we divide the current DCN research into several domains as shown in Fig.1.1, and describe prominent work in each domain. It should be noted that since many issues in the DCN research are closely coupled, most research proposals cover several domains and there are often overlapping areas between the boundaries of different research domains. For example, architectures like VL2 [23] and BCube [19] propose novel solutions for several domains like topology design, management and load balancing, while OpenFlow[51], a novel switch control technology, constitutes an integral part in the domain of DCN management.

1.3.1 Switching

Switches are key components of a DCN, whose performance and cost largely determine the overall performance and cost of the DCN, hence, the design of switches suitable for cloud computing have attracted much attention[53–58].

In order to face the ever-increasing bandwidth demand and keep the power consumption low in DCNs, switches for cloud computing purpose must be able to provide high throughput, low-latency data transmission with small energy footprint. Traditional electrical switches cannot meet these requirements, because copper wires are inherently inefficient in high-rate data transmission due to their high error rate and severe heat dissipation. Much effort has been dedicated to developing optical switches for data centers, see, for example, [54–56]. Optical switches are vastly superior to the electrical counterparts in terms of energy-efficiency and bandwidth capacity, however, their practical implementation also faces a series of challenges such as the lack of effective optical memory [61].

Another line of research aims towards more efficient, uniform switch control in DCNs. Traditionally, the control software (or, the control plane) of a switch is tightly coupled with the switching hardware (the data plane), and is often proprietary and closed for modification. This creates many problems for data center operators, because it imposes an obstacle for flexible deployment of different traffic engineering policies. Recently, a novel switch control framework called OpenFlow [43, 44, 51] has attracted much attention from the research community and industry. OpenFlow advocates a clear separation between the data plane and the control plane of switches, and promotes an uniform abstraction of different switch data planes. OpenFlow is an integral piece and a representative achievement in the software defined networking (SDN) movement, which will be described later.

1.3.2 Topology Design

The network topology plays a vital role in determining the cost and performance of a DCN. It also largely determines scalability, extendability and resilience of the network. The search for effective topologies for data center interconnection has motivated many research efforts in recent years [18–24].

One important design goal of DCN topologies is cost-effectiveness. The general theme of work in this area is to “scale out” instead of “scale up”, which means to find topologies that can connect hundreds of thousands of servers using only modest-sized commodity switches [18]. This is motivated by the fact that large, high-end switches are orders of magnitudes more expensive than commodity switches in today’s market. Meanwhile, a good DCN topology should also provide sufficient bandwidth capacity and good fault-tolerance, which is commonly achieved by providing rich path multiplicity between servers, see, for example, many proposed DCN topologies like fat-tree [18], BCube [19], Dcell[22], etc.

On the other hand, much attention has been put to designing topologies that enable convenient data center expansion, meaning that, servers can be added without needing to significantly alter the network structure. BCN [21] and Ficonn [20] belong to this category.

1.3.3 Load balancing

As mentioned before, DCN topologies, such as fat-tree and Bcube, provide large bandwidth capacity by deploying rich path parallelism between servers. However, how to efficiently utilize the bandwidth capacity of these multiple paths remains a challenging issue. Traffic load balancing, that is, the traffic load should be evenly spread across each of the multiple paths, is the key to efficient bandwidth utilization[15].

The currently adopted load balancing approaches in practical data centers are Equal-Cost Multi-Path(ECMP)[16] and Valiant Load Balancing(VLB) [17]. Both follow a simple idea: assign each flow to a path randomly or according to a hash function. Such approaches are often referred to as oblivious routing or randomized routing. However, as shown in [15], oblivious routing leads to unbalanced traffic load distribution, due to the possibility that multiple "elephant" flows, that is, flows with large bandwidth demand, are assigned to the same path.

To reduce the effect of multiple elephant flows colliding on the same path, Multi-path TCP (MP-TCP)[60], a modification to the original TCP protocol, is proposed, which splits a large flow into several smaller flows, each can be transmitted via a separate path, thus reduces the possibility of collision of large flows. Also, some researchers propose reactive flow scheduling, which periodically re-route certain flows based on network condition. The most notable work here is Hedera [15], which introduces a centralized controller that dynamically reschedules flow routes to maximize the network throughput. The controller periodically identifies elephant flows and network congestion, and re-route elephant flows on optimized paths.

Another approach is online flow scheduling, which selects the path for each flow adaptively when the flow starts according to the network condition. An example of online scheduling is to greedily route a flow along the path with least congestion. The challenge for both reactive scheduling and online scheduling is the high overhead for collecting the status of the whole network given the scale of DCNs, which involves research in the network control and management domain.

1.3.4 Control and Management

Managing and configuring a large DCN is a daunting task. As manual configuration is expensive and error-prone, there has been much effort towards more efficient DCN control and management.

One line of research focuses on minimizing the human factor in managing DCNs. For example, an automatic addressing scheme, DAC[26], presents a platform that automatically identifies the addresses for switches and servers based on their topology properties by solving a graph isomorphism problem. Other efforts aim to simplify the installment of new devices and achieve "plug-and-play". For example, Portland [24] assigns a pseudo-MAC address (PMAC) that encodes topology information to a host, and establish a mapping between the actual MAC address and PMAC for a network device through a gossip-like protocol. VL2 [23] uses IP routing and forwarding technologies and TCP's end-to-end congestion control mechanism, but manages to create a virtual lay-2 network environment by employing a directory system which maintains mapping between application-specific address (AA) and Location-specific address (LA).

The most notable development in the DCN control technology is the movement towards software defined networking (SDN) [44–46], led by the Open Networking Foundation (ONF) [43]. The main characteristics of SDN include: (1) SDN advocates a separation of the control Plane from the data plane of the network; (2) The control intelligence of the network is logically centralized; (3) The network infrastructure is abstracted to expose a common set of APIs for cloud applications. SDN brings many appealing features, such as better programmability of the network, increased network reliability and granular traffic control. Several SDN platforms have been proposed by universities and leading companies, such as NOX [44] and Google's B4 [46]. As our research relies on many features of SDN, a more detailed discussion of SDN will be given in later sections.

1.3.5 Novel Architectures

As mentioned before, traditional electrical networks struggle to accommodate the rapidly-increasing data volume in today's data centers, therefore, many researchers focus on constructing novel network architectures by using other transmission technology, such as wireless communication and optical communication, as a complementary augmentation of the electrical networks.

Wireless data center networking are discussed in [47–49], in which wireless "flyways" are established by adding wireless links between server racks to alleviate the traffic congestion problem. This is motivated by the observation that the servers in a data center are densely packed, and wireless devices are very efficient in providing high bandwidth over short ranges. Moreover, Wireless links can be set up without the cost of wiring, which makes it much more convenient to adapt the topology to the requirements of real-time transmissions.

Many also try to leverage optical fiber communication in DCNs. Optic offers significant advantage in power efficiency and bandwidth capacity [40–42]. The resulted hybrid optical/electrical, also referred to as Hybrid packet/circuit (Hypac) switched DCNs, augment the electrical packet switched (EPS) network with a rack-to-rack optical circuit switched (OCS) network. The goal is to combine the best features of both networks: the EPS network can provide flexible and packet-grained transmission, and enables each server to communicate with multiple other servers simultaneously; The OCS network, on the other hand, can deliver high bandwidth transmission with lower power consumption between any pair of server racks with established connections. Hence, Hypac DCNs are able to maintain the flexibility of packet switching, while allocating *on-demand* large bandwidth to the places where it is most needed by dynamically establishing OCS connections.

1.4 Our Research and Contributions

In this section, we explain the motivations for our research, and describe the contributions made in this dissertation.

The overarching objective of our work is to **explore the unique novel features and techniques available in data centers to design cost-effective, multicast-capable DCNs with predictably high performance**. The content of this dissertation can be roughly divided into three overlapping parts. The first part focuses on optical packet switching (OPS) [122–127]. Switches are critical components of a DCN, and OPS has been shown to be one of the most promising candidates in the next-generation switching technology, due to its potential to provide fine-grained transmission with huge bandwidth capacity and energy-efficiency. In this dissertation, we present a comprehensive study on optical packet switching for data center interconnection, including performance modeling, scheduling algorithm design and scheduler architecture design. The goal is to **achieve low-latency, high-throughput, reliable (low-loss) and multicast-capable optical packet switching for data center communication**.

The second part of our research focuses on deploying multicast communication in DCNs [128–134]. Many online applications and back-end infrastructural computations hosted by data centers require *one-to-many* group communication [31, 32]. Examples include redirecting search queries to multiple indexing servers, distributing executable binaries to a group of servers participating in MapReduce alike cooperative computations [7], replicating file chunks in distributed file systems [11], and so on.

However, *network-level multicast* has not been implemented in data centers due to the high network cost and complexity pertaining to management and scheduling of multicast traffic by traditional IP multicast. Without network-level support for multicast, duplicated packets have to be sent to every individual recipient in a group communication separately via unicast, which leads to long communication delay and wastes tremendous bandwidth. In contrast, implementing such group communication by multicast can avoid sending unnecessary duplicated packets in the network and deliver packets to all the recipients in one-shot, thus greatly reduce bandwidth consumption and transmission delay, and significantly improve application throughput. The lack of support for multicast in today's DCNs has impeded many critical data center applications to reach their full potential and imposed a grave hindrance to the cloud adoption for a wide variety of popular applications. This dissertation investigates a wide range of important topics related to the design of multicast-capable DCNs, including multicast scheduling at switch level, cost-effective topology construction, multicast traffic load balancing and bandwidth guarantee. Our objective is to **deploy cost-effective multicast in DCNs with guaranteed performance through exploring unique novel features and techniques in data centers.**

The third part of our research focuses on hybrid packet/circuit (Hypac) switched DCNs [135]. Hybrid packet/circuit (Hypac) switched DCNs, which argument the electrical packet switched (EPS) network with an optical circuit switched (OCS) network, have been proposed to combine the strengths of both types of networks. However, one problem with current Hypac DCNs is that the slow reconfiguration time of the OCS network results in severe performance drawbacks, such as degraded network predictability and deficiency in handling correlated traffic. We propose that such a problem can be mitigated through efficient configurations of both the electrical and optical networks. In our research, we **design efficient network configuration algorithms with the objective to exploit the full potential of Hypac DCNs and achieve predictably high network performance.**

Next, we list the contributions of this dissertation below.

- **Performance Modeling of Hybrid Optical Packet Switches** [122]. We present an efficient analytical model called the *aggregation model* that comprehensively analyzes various performance metrics for a novel type of OPS, called OpCut, under different types of traffic. Through inductive aggregation, the aggregation model can achieve a polynomial complexity to the switch size. We develop the aggregation model for the OpCut switch under both

Bernoulli traffic and ON-OFF Markovian traffic. The effectiveness of our model is validated by extensive simulations. The results show that the aggregation model is very accurate in all tested scenarios.

- **Low-Latency Multicast Scheduling for Optical Packet Switches** [123, 124, 127]. The support of high-throughput, low latency, reliable multicast traffic scheduling at switch level is the first step towards deploying multicast in DCNs. Here, we study multicast scheduling problem for optical packet switches. Given the intrinsic complexity of multicast traffic, the limited optical buffer capacity and the stringent time constraint on scheduling, multicast scheduling for OPS is a very challenging problem. We address these challenges by presenting the design of a novel optical buffer structure, a time-efficient algorithm and a pipelined and paralleled scheduler architecture that achieves O(1)-time scheduling.

- **Exploring Server Redundancy in Constructing Cost-Effective Nonblocking Multicast Fat-tree Data Center Networks.** [128–131]. Fat-tree is the most widely adopted topology in DCNs [18, 23, 24], because it can scale to hundreds of thousands of servers using only cheap commodity switches. However, it is still costly for fat-tree DCNs to support nonblocking (i.e. free of congestion) multicast communication, due to the large number of core switches required [27–30]. Since multicast is an essential communication pattern in many cloud services and nonblocking multicast communication can ensure the high performance of such services, reducing the cost of nonblocking multicast fat-tree DCNs is very important. On the other hand, server redundancy is ubiquitous in todays data centers to provide high availability of services. In this dissertation, we explore server redundancy in data centers to reduce the cost of nonblocking multicast fat-tree data center networks (DCNs). Our findings reveal important properties of multicast fat-tree DCNs and provide valuable theoretical guidelines that enable cost-efficient provisioning of fat-tree DCNs for data center operators.

- **On-line Multicast Scheduling with Bounded Congestion in Fat-tree Data Center Networks** [132–134]. Without an efficient flow scheduling algorithm that appropriately routes flows to achieve traffic load balance, heavy congestion may occur throughout a DCN, which prevents full utilization of link bandwidth and causes unpredictable network performance. Here, we study on-line multicast flow scheduling in fat-tree DCNs, where multicast flow requests arrive one by one without a priori knowledge of future traffic. To address the drastic traffic fluctuation in data centers, we consider a very general traffic model called hose traffic model, where the only assumption is that the total bandwidth demand of traffic that enters (leaves) an ingress (egress) link of each server at any time is bounded by the capacity of its network interface card. We present a low-complexity on-line multicast flow scheduling algorithm for fat-tree DCNs, which can achieve bounded congestion and traffic load balance under any arbitrary sequence of multicast flow requests that satisfy the hose model.

- **Collaborative Network Configuration in Hybrid Electrical/optical (Hypac) DCNs** [135]. In Hypac DCNs, the OCS network can only offer slow-reconfiguring, circuit-grained transmission. Meanwhile, the EPS network is shared in a best-effort fashion and is largely oblivious to the accompanying OCS network. This dichotomy between the OCS/EPS networks results in severe performance drawbacks, such as degraded network predictability and deficiency in handling correlated traffic [42]. Since the OCS/EPS networks have unique strengths and weaknesses, and are best suited for different traffic patterns, coordinating and collaborating the configuration of both networks is critical to eliminate these drawbacks and reach the full potential of Hypac DCNs. Leveraging the useful features of software defined networking, we propose a time-efficient algorithm called Collaborative Bandwidth Allocation (CBA) that configures both networks in a complementary manner, which, given sufficient bandwidth from both networks, can guarantee 100% throughput with a bounded delay.

Fig.1.1 gives an overview on where our work fits in the taxonomy for DCN research. This dissertation combines algorithm design, mathematical modeling, network optimization, analytical and simulation techniques to provide a comprehensive study on the design of high performance DCNs. The outcome of this research will not only benefit numerous cloud computing applications and facilitate cloud adoption for many future applications that rely on group communication, but also

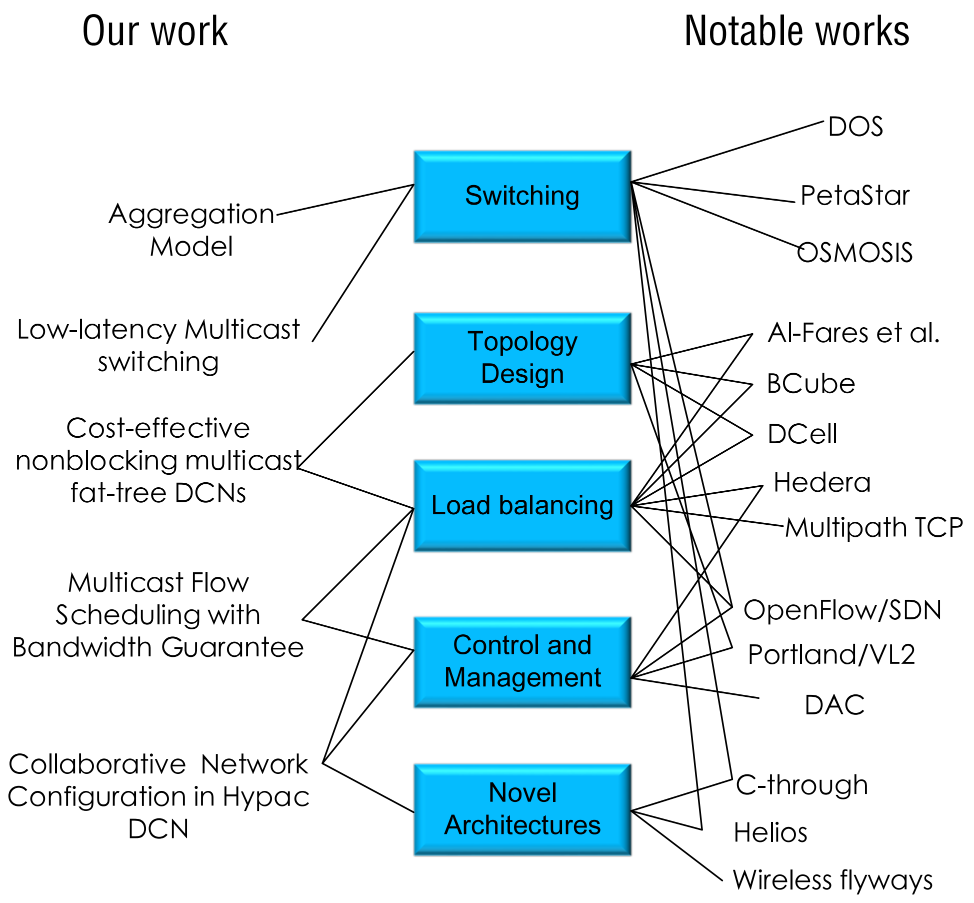


Figure 1.1: A taxonomy for DCN research.

have a significant impact on fundamental design principles and infrastructures for the development of future DCNs.

1.5 Dissertation Outline

The rest of the dissertation is organized as follows. Chapter 2 and chapter 3 present our study on optical packet switches. Chapter 2 presents an analytical model that accurately predicts the performance of the OpCut switch under different traffic patterns. Chapter 3 presents our design of high-throughput, low-loss and scalable multicast traffic scheduling in optical packet switches.

Chapter 4 and chapter 5 focus on deploying network-level multicast communication in fat-tree DCNs. Chapter 4 present our findings on exploring server redundancy in data centers to achieve cost-effective nonblocking multicast communication in fat-tree DCNs. In chapter 5, we propose a time-efficient, on-line multicast flow scheduling algorithm for fat-tree DCNs that can guarantee bounded network congestion under arbitrary traffic patterns. In chapter 6, we present our work on collaborative network configuration in Hypac DCNs. Finally, chapter 7 concludes the dissertation.

Chapter 2

Performance Modeling of Hybrid Optical Packet Switches with Shared Buffer

Optical packet switching is considered as a most promising building block in future DCNs [53]. However, the capacity of practical optical buffer, i.e., Fiber Delay Lines (FDLs), is very limited currently, which makes it difficult to keep packet loss to an acceptable level in all-optical packet switching. Thus, hybrid optical/electronic switch architectures, such as the switch proposed in [62] which we refer to as the OpCut switch, are good alternatives due to their potential to achieve ultra-low packet loss and packet delay. This chapter presents an efficient analytical model called the *aggregation model* that comprehensively analyzes various performance metrics of the OpCut switch under different types of traffic. By inductively aggregating more queues in the buffer into a block, the aggregation model can achieve a polynomial complexity to the switch size. We first develop the aggregation model for the OpCut switch under both Bernoulli traffic and ON-OFF Markovian traffic. We then extend the model to analyzing the performance of the WDM OpCut switch, where there are multiple wavelengths on each fiber. The effectiveness of our model is validated by extensive simulations. The results show that the aggregation model is very accurate in all tested scenarios.

The rest of this chapter is organized as follows. Section 2.1 explains the background and related work on switch performance modeling. Section 2.2describes the structure of OpCut. Section 2.3 gives the basic idea of the aggregation model. Section 2.4 and Section 2.5 present the details of the aggregation model under Bernoulli traffic and ON-OFF Markovian traffic, respectively. Section 2.6 extends the model to the WDM OpCut switch. We validate the effectiveness of the proposed model through extensive simulations and present the simulation results in Section 2.7. Finally, Section 2.8 concludes this chapter.

2.1 Introduction and Related Work

Optical networking has been widely adopted to transport high volume traffic in core networks due to the huge bandwidth of optics. Currently deployed optical networks primarily use optical circuit switching (OCS), which may not fit the dynamic bursty traffic of typical Internet applications due to low resource utilization. To overcome this problem, much research has been conducted in the past few years towards the development of optical burst switching (OBS) and optical packet switching (OPS) networks, which offer more flexible alternatives. Compared to OBS, OPS networks can achieve better scalability and higher bandwidth by switching data on a packet-by-packet basis. A major challenge for OPS networks is how to resolve output contention in OPS switches, which occurs when multiple optical packets simultaneously go to the same output of an OPS switch, because the limited capacity of optical buffer built by FDLs. Various contention avoidance and resolution schemes have been proposed to deal with this problem [61], yet none of those schemes is able to reduce packet loss to a reasonable level under heavy traffic load.

To address these challenges, a hybrid optical/electronic packet switch architecture [62, 63] combining optical cut-through with shared electronic buffer was proposed as a promising candidate for practical deployment. It utilizes the large capacity of electronic buffer to resolve contention while letting packets cut through the switch optically as much as possible. This switch architecture was shown to have the potential to achieve ultra-low packet loss and average packet delay with a minimal number of O/E/O conversions. In this chapter, we will systematically study the analytical performance of this switch architecture. In the rest of the chapter, we will simply call it *OpCut* switch.

There has been much work on the performance modeling of various types of switches in the literature. Based on the media packets are transmitted, switches can be divided into two categories: electronic switches, such as [64–68], and optical switches, such as [69, 72, 73]. Among electronic switches, shared buffer switches [65–68] have received much attention due to its efficient buffer resource allocation. However, since all the queues in a shared buffer are strongly dependent on each other, the performance modeling of shared buffer switches is difficult. Thus, approximation models based on different assumptions on the packet distribution were proposed. For example, the model in [66] assumed that all the queues are independent of each other. In [67], an analytical model that recursively aggregates queues in the shared buffer was proposed, and was shown to provide more accurate predication on switch performance than previous models.

Performance modeling for optical switches has also been well studied and mainly focuses on two types of optical switches: optical burst switches (OBS) [69] and optical packet switches (OPS) [72, 73]. In [69], an OBS switch was modeled by the queueing network, where burst arrivals are described by a Markov process. In [73], the performance of bufferless WDM optical packet switches with limited-range wavelength conversion was studied by finding the number of used wavelengths on each output fiber recursively.

Despite of the myriad amount of work on performance modeling of electronic switches and optical switches, there has been very limited study on the performance modeling of hybrid optical/electronic switches, such as the OpCut switch. In [62], the performance of the OpCut switch with an infinite buffer was analyzed under Bernoulli traffic. Although the model can give some quantitative characteristics of the OpCut switch, the infinite buffer is not realistic, and some important properties, such as cut-through ratio and packet loss caused by buffer overflow, are not considered. Given the potential of the OpCut switch to achieve ultra-low latency and packet loss, we feel a comprehensive analytical model for the OpCut switch that analyzes various important properties under different types of traffic is needed to fully evaluate the performance of the OpCut switch.

Our work is motivated by the lack of such a model. We will present an efficient analytical model called the *aggregation model* that can accurately predict the performance of the OpCut switch with any buffer size under both Bernoulli traffic and ON-OFF Markovian traffic. By inductively aggregating queues in the shared buffer into a block, the aggregation model has only a polynomial complexity to the switch size and buffer size. Our simulation results verify that the aggregation model achieves a near-perfect accuracy in predicating system dynamics, such as packet loss ratio and average packet delay.

2.2 The OpCut Switch

We consider an $N \times N$ OpCut switch as shown in Fig. 2.1. The OpCut switch is "hybrid" in the sense that it is equipped with a recirculating shared electronic buffer. An optical packet that arrives at the switch input is sent to its destined output directly, trying to "cut-through" the switch in optical domain. When there is contention at an output, i.e., multiple optical packets contend for the same output, one packet is chosen randomly to cut-through the switch by the scheduler, while the *leftover packets* are picked up by optical receivers and converted to electronic form then stored in the shared electronic buffer. The buffer is shared by all leftover packets and only drops packets when there is no space left, thus has lower packet loss than other types of buffers of the same buffer size. The leftover packets are multiplexed and stored in the queues in the buffer corresponding to their destinations. N logic FIFO queues, queue 1, queue 2, . . ., queue N, are maintained in the buffer with queue i storing the leftover packets destined for output i, $1 \leq i \leq N$. While the size of each queue is variable, the total size of N queues is always equal to the buffer size B. The packets in the buffer are converted back to optics later by optical transmitters and sent to the switch output. To make the analysis tractable, we assume that there are a sufficient number of optical transmitters/receivers, so that there is no contention for transmitters/receivers, and packet loss is caused only by buffer overflow (in our case, N transmitters/receivers are sufficient). By allowing packets to "cut-through" optically, the OpCut switch has the potential to achieve ultra-low latency and packet loss with a minimal number of O/E/O conversions.

The switch is time-slotted and the operations are performed in a two-phase manner as follows.

- **Phase 1**: Cut-through. Upon receiving incoming packets, all the inputs send requests to all the outputs. Then each output randomly picks one of the requests to grant the transmission of a packet, and other leftover packets will be stored in the buffer.

- **Phase 2**: Buffer management.

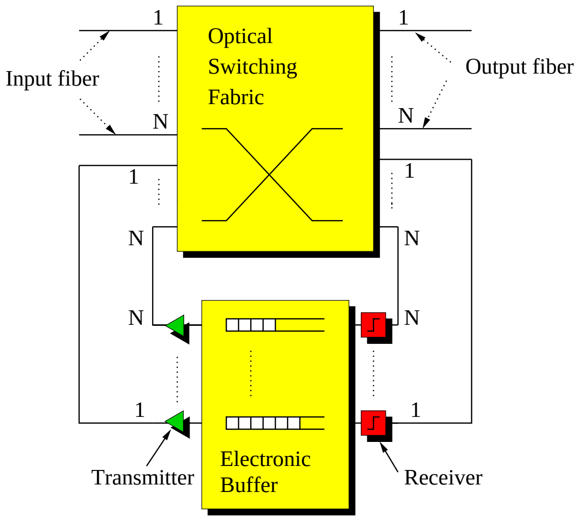


Figure 2.1: An $N \times N$ OpCut switch.

- **Step 1**: Packet transmission. If an output is not occupied in the "cut-through" phase and the queue corresponding to it is nonempty, the head packet buffered in the queue is sent out to the output through an optical transmitter.
- **Step 2**: Receive packets. The leftover packets are put into the queues according to their destinations.
- **Step 3**: Drop packets if necessary. After the previous two steps, buffer is now in the "intermediate state." If the number of packets that have to be buffered exceeds buffer size B, some of them have to be dropped according to the *random drop with pushout policy*, which, if the buffer size is exceeded by V, will drop V packets randomly out of the total $B + V$ packets. Note that, both new packets and old packets already in the buffer are equally likely to be dropped.

2.3 Aggregation Model - An Overview

In this section, we discuss some challenges in performance modeling of the OpCut switch and introduce the aggregation model.

The difficulty of analytically modeling the behavior of the OpCut switch lies in the strong dependency among the queues in the shared buffer. The numbers of pack-

Table 2.1: Notations in Aggregation Model

N	Number of input/output fibers of the switch
k	Number of wavelengths per fiber
B	Size of shared buffer
(X, Y)	State of queue 1 (X) and queue 2 (Y)
(X_0, Y_0)	Initial state of queues 1 and 2 before receiving leftover packets
(X_m, Y_m)	Intermediate state of queues 1 and 2
(X_1, Y_1)	Final state of queues 1 and 2 after packet drop
l_x	Leftover packets in arrived packets for output 1
l_y	Leftover packets in arrived packets for output 2
a	Buffered packets to be sent from queue 1
b	Buffered packets to be sent from queue 2
U	Packets stored in I-queue block
u	Packets stored in $(I+1)_{th}$ queue
U^*	Packets stored in $(I+1)$-queue block
S	Packet arrivals for first I outputs
s	Packet arrivals for $(I+1)_{th}$ output
S^*	Packet arrivals for first $I+1$ outputs
L	Leftover packets in newly arrived packets for first I outputs
l	Leftover packets in newly arrived packets for $(I+1)_{th}$ queue
L^*	Leftover packets in newly arrived packets for first $I+1$ outputs
O	Buffered packets to be sent to first I outputs
o	Buffered packets to be sent to $(I+1)_{th}$ output
O^*	Buffered packets to be sent to first $I+1$ outputs
L_S^I	Leftover packets among S arriving packets destined for first I outputs
$CT_I(O\|S, L, U)$	Probability that O buffered packets are transmitted from I-queue block in state (S, L, U)
(α, β, γ)	State of inputs under ON-OFF Markovian traffic
$CT_I(O\|\alpha, \gamma, L, U)$	Probability that O buffered packets are transmitted from I-queue block in state (α, γ, L, U) under ON-OFF Markovian traffic

ets buffered in the N queues at any moment are a set of random variables strongly depending on each other. The strong dependency comes from inputs, buffer queues and outputs. For example, if there are x packets destined for output 1 in a time slot, there cannot be more than $N - x$ packets destined for other outputs in the same time slot. Similarly, for a buffer of size B, the fact that there are y packets in the i_{th} queue rules out the possibility that there are more than $B - y$ packets in other queues. The dependency also comes from the fact that there cannot be more packets going through an output than its capacity in any time slot, which means that if an output is occupied by a "cut-through" packet, it is out of consideration for leftover packets in the buffer in that time slot. Since the number of joint states of queue lengths is an exponential function of the switch size, there is no scalable method to accurately represent all these states.

Fortunately, the OpCut switch has a very useful property that despite of the strong dependency, queues in the shared buffer interact in such a way that much unnecessary details can be omitted when finding the behavior of the queues. Take a tagged queue for example. It is only concerned with the total number of packets stored in other queues: if there are x packets stored in other queues, its maximum capacity is $B - x$, and it does not need to know how exactly those packets are distributed among other queues. If we already have an accurate model describing the behavior of the $(N - 1)$-queue "block," the characteristic of the tagged queue and thus the entire switch can be determined. This property allows us to analyze the OpCut switch in an inductive way: Finding the behavior of two queues, say, queues for output 1 and output 2, first. Then, aggregate these two queues into a block, whose behavior can be deduced from the behavior of its components. Next, consider this block and the third queue, then repeat the aggregation procedure. This process can be carried on till all the queues are included in a block, by then we have an accurate estimation of the entire switch.

To characterize the interference between the two phases in the OpCut switch, we need to find the distribution of leftover packets and the occupancy condition of outputs, which can be derived from the distribution of the input traffic. Then we consider two arbitrarily chosen queues and their corresponding outputs, say, queues for output 1 and output 2. The states of these two queues can be represented by a a pair of random variables (X, Y), where X is the number of buffered packets in queue 1 and Y is the number of buffered packets in queue 2. Regarding (X, Y) as a two-dimensional Markov chain, we can find the transition rate of (X, Y), thus the steady-state distribution of (X, Y), given the distribution of leftover packets and occupancy condition of outputs. Next, combine the two queues into one block. The state of the block can be described by a variable U that denotes the total number of buffered packets in these two queues, whose transition also has Markov property over time. To fully describe the behavior of the block, the following aspects need to be specified: the steady-state distribution of this block and the probability mass function (p.m.f.) of the total number of buffered packets to be sent out from this block. The former can be obtained by merging the corresponding sub-states of two queues. To describe the latter, we define a conditional probability $CT_2(O|S, L, U)$, which denotes the probability that O buffered packets are transmitted from this block that contains U packets currently, given that there are S arrived packets destined for the first two outputs in this time slot, among which L leftover packets are put into buffer.

After obtaining the characteristics of the two-queue block, we move on to the three-queue case, queue 1 through queue 3. We regard the first two queues as a block, thus the state of these three queues can be represented by two variables (U, u), where u is the number of buffered packets in the third queue. With the information on the two-queue block we found in the previous step, the transition rate and steady-state distribution of this two-dimensional Markov chain (U, u) can be deduced. After the aggregation of the two-queue block and the third queue, we obtain the conditional probability $CT_3(O|S, L, U)$ to describe the new block, where U is the total number of buffered packets in the three-queue block. This process continues till all the queues are included in a single block, then we have an accurate description of the switch behavior. Useful information such as packet loss ratio and average packet delay can be extracted from this N-queue block. Since there are N steps in this model and in each step only two components are considered, as will be seen later, the aggregation model is very efficient and has only a polynomial complexity to the switch size and buffer size. The aggregation model has several variations depending on different types of traffic. In the next two sections, we will give the aggregation model under Bernoulli traffic and ON-OFF Markovian traffic, respectively. The notations used are summarized in Table 2.1.

2.4 The Model for Bernoulli Traffic

In this section, we derive the aggregation model under Bernoulli traffic and analyze the performance of the OpCut switch in terms of cut-through ratio, packet loss ratio and average packet delay.

Bernoulli traffic is a widely used traffic model in various network settings [71, 73]. It can be described as follows.

- The packet arrival at each input is Bernoulli with parameter ρ, i.e., in a time slot, the probability that there is a packet arrival at an input is ρ and there is no correlation between time slots.
- The destination of a packet is uniformly distributed over all N outputs;
- The arrival at each input is independent of each other.

We now give the detailed description of the aggregation model under Bernoulli traffic. We first consider the cut-through phase.

2.4.1 Preliminary - Cut-through

Since the portion of the packets that cut through the switch is an important performance metric of the OpCut switch and only leftover packets failed to cut through will be sent to buffer, we need to first find the cut-through ratio and the characteristics of leftover packets in the cut-through phase.

Under Bernoulli traffic, the arrival at each input is independent of each other and there is no correlation between time slots. Thus, the probability of k packet arrivals among N inputs in a time slot is

$$P(k) = \binom{N}{k} \rho^k (1-\rho)^{N-k} \tag{2.1}$$

If two or more of these packet arrivals destined for the same output, one of the packets will "cut-through" and the rest will be stored in the buffer as leftover packets. The probability of r packets cutting through among k packet arrivals is

$$P(r|k) = \frac{S(k,r)(N)_r}{\sum_{s=1}^{k} S(k,s)(N)_s}$$

where $S(k, r)$ is the Stirling number of the second kind, which denotes the number of ways to partition a set of k distinct elements into r nonempty subsets, and $(N)_r = N!/(N-r)!$.

The *cut-through ratio* C of the OpCut switch is defined as the ratio of packets cutting through to the total number of packet arrivals. The expected cut-through ratio is given by

$$E(C) = \sum_{k=0}^{N} \sum_{r=0}^{k} \frac{rP(r|k)P(k)}{N \times \rho} \tag{2.2}$$

We will simply call it *cut-through ratio* in the rest of the chapter.

In order to describe the probability mass function (p.m.f.) of leftover packets, we define L_S^I as the number of leftover packets among S arriving packets destined for first I outputs. Consider the one queue case first. The p.m.f. of L_S^1 can be determined by

$$P(L_S^1 = l) = \begin{cases} 1 & l = \max\{0, S-1\} \\ 0 & \text{otherwise} \end{cases} \tag{2.3}$$

Now consider the general case when $I \geq 1$. Given that there are S packet arrivals destined for first I outputs, having L leftover packets is equivalent to that $S - L$ packets cut-through. Since there are at most I packets cut through among the first I outputs in a time slot, the p.m.f. of L_S^I is obtained as

$$P(L_S^I = L) = \begin{cases} \frac{S(S,S-L)(I)_{S-L}}{\sum_{k=1}^{I} S(S,k)(I)_k} & S - L \leq I \\ 0 & \text{otherwise} \end{cases} \tag{2.4}$$

2.4.2 Getting Started - Analyzing Two Queues

After obtaining the p.m.f. of leftover packets for all I-queue blocks, $1 \leq I \leq N$, we are now in the position to derive the aggregation model. We begin with the queues for output 1 and output 2, queue 1 and queue 2. Using (X, Y) to represent the number of buffered packets in queue 1 and queue 2. Suppose there are currently X_0 packets in queue 1 and Y_0 packets in queue 2. Since the packet arrival at each input is Bernoulli with parameter ρ and the destinations of packets are uniformly distributed over all outputs, the number of packet arrivals towards output 1 and the number of packet arrivals towards output 2 follow the multinomial distribution. Thus the probability that there are x packet arrivals destined for output 1 and y packet arrivals destined for output 2 is

$$P_{x,y}(x,y) = \frac{N!}{x!y!(N-x-y)!}\left(\frac{\rho}{N}\right)^{x+y}\left(1-\frac{2\rho}{N}\right)^{N-x-y} \tag{2.5}$$

With x packets towards output 1, the number of its leftover packets $l_x = \max\{x-1,0\}$. If output 1 is not occupied in the cut-through phase, i.e., $x = 0$, and queue 1 is nonempty, a buffered packet will be sent from queue 1 to output 1 in this time slot, Thus the number of buffered packets to be sent from queue 1 can be expressed as $a = \min\{X_0, \max\{1-x,0\}\}$. Clearly, $a = 0$ or 1. Similarly, given there are y packet arrivals towards output 2, the number of leftover packets to be stored in queue 2 $l_y = \max\{y-1,0\}$ and the number of buffered packets to be sent to output 2 can be expressed as $b = \min\{Y_0, \max\{1-y,0\}\}$. Also, $b = 0$ or 1. After receiving leftover packets and sending out a buffered packet (if any), queue 1 will have $X_m = X_0 + l_x - a$ packets, and queue 2 will have $Y_m = Y_0 + l_y - b$ packets. Hence, the transition probability of the Markov chain from state (X_0, Y_0) to state (X_m, Y_m), given there are x and y packets destined for output 1 and output 2 respectively, is

$$\begin{aligned}&\Lambda(X_m, Y_m|X_0, Y_0, x, y) = \\ &\begin{cases} 1 & X_m = X_0 + l_x - a, Y_m = Y_0 + l_y - b \\ 0 & \text{otherwise}\end{cases}\end{aligned} \tag{2.6}$$

If there is no buffer overflow, the next state of the Markov chain is (X_m, Y_m). Otherwise, some of the packets will be dropped according to the random drop with pushout policy, and the two queues may store fewer packets than X_m and Y_m. Note that it would be difficult, if not impossible, to know exactly how many packets will be dropped from queue 1 and queue 2, because at this moment, we have no information about other queues. Therefore, we make an assumption called "zero external interference," which states that *"if $X_m + Y_m \leq B$, no packets will be dropped from queue 1 and queue 2; otherwise, $X_m + Y_m - B$ packets will be dropped from* *these two queues."* This assumption is equivalent to only considering queue 1 and queue 2 in the buffer, and the rest or "external queues" will not grow to a size large enough to interfere with queue 1 and queue 2. Validation of this assumption was provided in [67]. According to the random drop with pushout policy, each packet is equally likely to be dropped when buffer overflows. Hence the probability that the switch drops v_x packets from queue 1 and v_y packets from queue 2, where $v_x + v_y = X_m + Y_m - B$, $0 \leq v_x \leq X_m$ and $0 \leq v_y \leq Y_m$ is

$$P_D(v_x, v_y; X_m, Y_m) = \frac{\binom{X_m}{v_x}\binom{Y_m}{v_y}}{\binom{X_m+Y_m}{X_m+Y_m-B}} \tag{2.7}$$

Combining the above discussions, the transition rate of the Markov chain from state (X_0, Y_0) to (X_1, Y_1) consists of the following two cases. Case 1: When $X_1 + Y_1 < B$, the transition rate is

$$\sum_{x,y} \Lambda(X_1, Y_1 | X_0, Y_0, x, y) P_{x,y}(x, y)$$

where $P_{x,y}(x, y)$ is given in Equation (2.5). Since packets stored in the two queues do not exceed buffer size in this case, no packets would have been dropped from queue 1 and queue 2 according to "zero external interference assumption." Hence the probability that the Markov chain will go from (X_0, Y_0) to (X_1, Y_1) is the probability that the number of packets stored in queue 1 changes from X_0 to X_1 and that stored in queue 2 changes from Y_0 to Y_1. Case 2: If $X_1 + Y_1 = B$, by the random drop with pushout policy, some packets may have been dropped from two queues. In other words, the two queues may have visited some intermediate state (X_m, Y_m) before reaching state (X_1, Y_1), where $X_m \geq X_1$, $Y_m \geq Y_1$. The probability that the two queues changes from state (X_0, Y_0) to state (X_1, Y_1) is equal to the probability that state (X_0, Y_0) transits to some intermediate state (X_m, Y_m), then drops exactly $X_m - X_1$ packets from queue 1 and $Y_m - Y_1$ packets from queue 2. Thus, the transition rate when $X_1 + Y_1 = B$ is

$$\begin{aligned}\sum_{(X_m,Y_m)} \sum_{x,y} \quad & \Lambda(X_m, Y_m | X_0, Y_0, x, y) P_{x,y}(x, y) \\ & \times P_D(X_m - X_1, Y_m - Y_1; X_m, Y_m)\end{aligned}$$

After we obtain the transition rates for all states, the steady-state distribution of the Markov chain $\pi(X, Y)$ can be obtained. With the accurate description of characteristics of the two queues, we can aggregate the two queues into a block.

2.4.3 Aggregating Two Queues

In this subsection, we show how to aggregate two queues into a block. Let U denote the number of packets stored in the block. $U = X + Y$, where (X, Y) is called a sub-state of U. Merging all sub-states, the probability that the block is in state U, or $\pi(U)$, is $\sum_i \pi(X_i, Y_i)$, where (X_i, Y_i) is the i_{th} sub-state of U if $X_i + Y_i = U$. Recall that only unoccupied outputs in the cut-through phase can receive buffered packets. To characterize the occupancy status of output 1 and output 2, we also use a conditional probability $CT_2(O|S, L, U)$, which can be interpreted as the probability that O buffered packets can be sent out from this two-queue block, given that there are currently U packets in the block and S packet arrivals destined for the first two outputs, among which L packets are left-over packets. We define (x, y, l_x, l_y, X, Y) as a sub-state of (S, L, U), if $x + y = S$, $l_x + l_y = L$ and $X + Y = U$. Let $\Theta(O|x, y, l_x, l_y, X, Y)$ be the probability that O buffered packets are sent out from the two-queue block in sub-state (x, y, l_x, l_y, X, Y). $\Theta(O|x, y, l_x, l_y, X, Y) = 1$ when $O = a + b$, where a and b are the numbers of buffered packets from queue 1 and queue 2, respectively, otherwise, it is 0. Given sub-state (x, y, l_x, l_y, X, Y), we have $a = \min\{X, \max\{1 - x, 0\}\}$, and $b = \min\{Y, \max\{1 - y, 0\}\}$. Let $P(x, y, l_x, l_y, X, Y)$ be the probability that the block is in sub-state (x, y, l_x, l_y, X, Y). It can be determined by

$$P(x, y, l_x, l_y, X, Y) = \begin{cases} P_{x,y}(x, y)\pi(X, Y) & \\ \qquad l_x = \max\{x - 1, 0\}, l_y = \max\{y - 1, 0\} & \\ 0 \qquad \text{otherwise} & \end{cases}$$

Merging all the sub-states, we have the probability that the block is in state (S, L, U), $P(S, L, U) = \sum_i P(x_i, y_i, l_{x_i}, l_{y_i}, X_i, Y_i)$, where $(x_i, y_i, l_{x_i}, l_{y_i}, X_i, Y_i)$ is the i_{th} sub-state of (S, L, U). Then,

$$CT_2(O|S, L, U) = \sum_i \frac{\Theta(O|x_i, y_i, l_{x_i}, l_{y_i}, X_i, Y_i)P(x_i, y_i, l_{x_i}, l_{y_i}, X_i, Y_i)}{P(S, L, U)}$$

After we obtain the characteristics of the two queue block, we can use them in the next iteration of the aggregation model when we aggregate the two-queue block with the third queue. Next, we will explain the general case that I-queue block is aggregated with the $(I+1)_{th}$ queue.

2.4.4 Iteration - More Queues

Suppose that the first I queues have been aggregated and the characteristics of this I-queue block has been obtained. We now study the first $I+1$ queues by considering the block of the first I queues and the $(I+1)_{th}$ queue.

Assumed that the first $I+1$ queues are in state (U_0, u_0) initially, with U_0 and u_0 being the number of packets in the first I queues and the $(I+1)_{th}$ queue respectively. Since the number of packet arrivals towards the first I outputs and that towards the $(I+1)_{th}$ output also follow multinomial distribution. The probability that there are S packet arrivals for the first I outputs and s packet arrivals for the $(I+1)_{th}$ output is

$$P_{S,s}(S,s) = \frac{N!}{S!s!(N-S-s)!}\left(\frac{I\rho}{N}\right)^S\left(\frac{\rho}{N}\right)^s\left(1-\frac{(I+1)\rho}{N}\right)^{N-S-s}$$

Consider the $(I+1)_{th}$ queue first. When there are s packets destined for the $(I+1)_{th}$ output, the number of leftover packets for queue $(I+1)$ $l = \max\{s-1, 0\}$, and the number of buffered packets to be sent out in this time slot $o = \min\{u, \max\{1-s, 0\}\}$. After receiving leftover packets and sending out the buffered packet, the number of buffered packets in the $(I+1)_{th}$ queue becomes $u_m = u_0 - o + l$.

Next, consider the block of I queues. With the conditional probability $CT_I(O|S, L, U)$ from the last iteration (for example, Equation (2.8) when $I = 2$) and the p.m.f. of leftover packets from Equation (2.4), the transition rate of the Markov chain for the block of $I+1$ queues can be obtained in a similar way to the two-queue case. We again make the "zero external interference assumption," which means that the buffer randomly drops packets only when in the intermediate state $U_m + u_m > B$. The probability that the $I+1$ queues change from state (U_0, u_0) to intermediate state (U_m, u_m), given S and s packets destined for the first I outputs and the $(I+1)_{th}$

output respectively is

$$\Lambda(U_m, u_m|U_0, u_0, S, s) = \begin{cases} \sum_{L,O} CT_I(O|S, L, U_0)P(L_S^I = L) \\ \qquad U_m = U_0 - O + L \text{ and } u_m = u_0 - o + l \\ 0 \qquad \text{otherwise} \end{cases}$$

We now derive the transition rate of Markov chain (U, u) in two different cases. After receiving leftover packets and sending out buffered packets, the I-queue block and the $(I + 1)_{th}$ queue are in an intermediate state (U_m, U_m'). If $U_m + u_m < B$, there are no packets dropped by "zero external interference assumption." Hence, the intermediate state (U_m, u_m) is the final state (U_1, u_1). The transition rate from (U_0, u_0) to (U_1, u_1) is thus obtained as

$$\sum_{S,s} \Lambda(U_1, u_1|U_0, u_0, S, s)P_{S,s}(S, s)$$

If $U_1 + u_1 = B$, some packets may have been dropped from the intermediate state (U_m, u_m) before the $(I + 1)$-queue block reaches state (U_1, u_1). The transition rate in this case can be written as

$$\sum_{U_m,u_m} \sum_{S,s} \qquad \Lambda(U_m, u_m|U_0, u_0, S, s)P_{S,s}(S, s) \times P_D(U_m - U_1, u_m - u_1; U_m, u_m)$$

where $P_D(U_m - U_1, u_m - u_1; U_m, u_m)$ is the probability that $U_m - U_1$ packets are dropped from the I-queue block and $u_m - u_1$ packets are dropped from the $(I+1)_{th}$ queue, which is given by Equation (2.7). After obtaining the transition rate of the Markov chain, the steady-state distribution, $\pi(U, u)$, can be found.

The steady-state distribution of U^*, which is the number of buffered packets in the $(I + 1)$-queue block, can be obtained by merging all the sub-states $\pi(U_i, u_i)$. We can also find the behavior of $I + 1$ queues described by conditional probability $CT_{I+1}(O^*|S^*, L^*, U^*)$. We call (S, s, L, U, u) a sub-state of (S^*, L^*, U^*), if $S + s = S^*$, $L = L^* - \max\{s - 1, 0\}$ and $U + u = U^*$. Similar to the two-queue case, let $\Theta(O^*|S, s, L, U, u)$ be the probability that O buffered packets are sent out from the first $(I + 1)$ queues in sub-state (S, s, L, U, u). Since there can be $o = \min\{u, \max\{1 - s, 0\}\}$ packets sent out from $(I + 1)_{th}$ queue, $\Theta(O^*|S, s, L, U, u)$

is simply the probability that there are $O^* - o$ packets that can be sent out from the first I queues, which is the $CT_I(O^* - o|S, L, U)$ found in the last iteration. Next, let $P(S, s, L, U, u)$ be the probability of that the $I + 1$ queues are in sub-state (S, s, L, U, u).

$$P(S, s, l, U, u) = P(L_S^I = l)P_{S,s}(S, s)\pi(U, u)$$

Merging all the corresponding sub-states of (S^*, L^*, U^*), we have the probability that the $I + 1$ queues are in state (S^*, L^*, U^*)

$$P(S^*, L^*, U^*) = \sum_i P(S_i, s_i, L_i, U_i, u_i)$$

where $(s_i, s'_i, l_i, U_i, u_i)$ is the i_{th} sub-state of (S^*, L^*, U^*). Then,

$$CT_{I+1}(O^*|S^*, L^*, U^*) = \sum_i \Theta(O^*|S_i, s_i, L_i, U_i, u_i)P(S_i, s_i, L_i, U_i, u_i)/P(S^*, L^*, U^*)$$

We now have a complete description of the behavior of the $(I + 1)$-queue block, and continue to the next iteration.

2.4.5 Using the Model

After aggregating all the queues into one block, the steady-state distribution of the entire buffer $\pi(U)$ and the conditional probability $CT_N(O|S, L, U)$ can be obtained. With this information, we are now in the position to derive the packet loss ratio and average packet delay of the switch.

Packet Loss ratio. The packet loss ratio of the OpCut switch denoted by α with respect to traffic load ρ is given by

$$\begin{aligned}\alpha \;=\; & \sum_{S=0}^{N}\sum_{L=0}^{N}\sum_{U=0}^{B}\sum_{O=0}^{N}\pi(U)CT_N(O|S, L, U) \\ & \times P(L_S^N = L)P_S(S)[U + L - O - B]^+\end{aligned} \tag{2.8}$$

where $P_S(S)$ is the probability that there are a total of S, $0 \leq S \leq N$, packet arrivals in a time slot, which is given by Equation (2.1). The notation $[\;]^+$ means $[x]^+ = x$ if $x > 0$, otherwise $[x]^+ = 0$. To see why Equation (2.8) holds, note that given the switch in state U, after receiving l packets and sending out o packets, the buffer is in intermediate state $U_m = U + l - o$. If the number of packets in the intermediate state exceeds the buffer size, exactly $U + l - o - B$ packets will be dropped, otherwise no packets will be dropped.

Average Packet Delay. The delay of a packet in a switch is generally defined as the number of time slots it stays in the buffer before being sent out. In our case, a packet could be sent out or dropped, thus we redefine the delay of a packet as the number of time slots a packet stays in the buffer before being sent out or dropped. If a packet cuts through optically, its delay is 0. As Little's formula holds for general stable systems, it can be applied to analyzing the average packet delay of the OpCut switch, which is stable. To use Little's formula, we first need the average buffer size S

$$E(S) = \sum_{U=0}^{B} \pi(U) \times U$$

By Little's formula, the average packet delay

$$D = \frac{E(S)}{N \times \rho} \tag{2.9}$$

By now we have obtained a complete set of performance metrics for the OpCut switch under Bernoulli traffic in terms of cut-through ratio, packet loss ratio and average packet delay (Equations (2.2), (2.8) and (2.9)). The complexity of the aggregation model under Bernoulli traffic is mainly determined by the size of the state space. Since there are N iterations in the aggregation model, and in each iteration the size of the state space is $O(B^2)$, the complexity of the aggregation model under Bernoulli traffic is $O(NB^2)$.

2.5 The Model for ON-OFF Markovian Traffic

So far we have derived the aggregation model under Bernoulli traffic for single wavelength OpCut switch. In this section, we will show how to apply the idea of aggregation to *ON-OFF Markovian traffic* for single wavelength OpCut switch. ON-OFF Markovian traffic is another widely used traffic model in the literature [74]. Under ON-OFF Markovian traffic, the input alternates between two states "idle" and "busy." If input is in "busy" state, there is a packet arrival in the current time slot, otherwise, there is no packet arrival. Since the transition is Markovian, each input can be modeled as a two-state Markovian chain. When in “idle” state, the probability that it will go to “busy” state in the next time slot is q while the probability that it will stay in “idle” state is $1 - q$. Similarly, the probability that an input will switch from “busy” state to “idle” state is p and the probability that it will stay in “busy” state is $1 - p$. Packets arrived in consecutive busy time slots and sharing the same input and output are called a *burst*. The average load for each input is thus $\rho = q/(p + q)$. In this section, we show that the aggregation model can also be used to accurately analyze the performance of the OpCut switch under ON-OFF Markovian traffic.

Though the general scheme of the aggregation model for ON-OFF Markovian traffic is similar to that for Bernoulli traffic, the dependence among arrivals in consecutive time slots at each input makes it more complex, thus modification to the aggregation model is necessary to adapt it to ON-OFF Markovian traffic. We briefly outline the differences between two models next. Recall that we consider a block of first I queues and the $(I + 1)_{th}$ queue in each iteration of the aggregation model, thus a triple (α, β, γ) can be defined to represent the state of inputs, where α is the number of busy inputs that send packets destined for first I outputs, β is the number of busy inputs that send packets to output $I + 1$, and γ is the total number of busy inputs. Note that there is exactly one packet arriving at each busy input. Thus this triple also tells us the exact number of packets towards first I outputs and the number of packets towards the $(I+1)_{th}$ output. For any I, $1 \leq I \leq N-1$, (α, β, γ) is a three-dimensional Markov chain, of which the transition rate and steady-state distribution $\pi(\alpha, \beta, \gamma)$ can be obtained [67]. Taking the dependence between arrivals in consecutive time slots into consideration, the cut-through ratio of the OpCut switch under ON-OFF Markovian traffic can be derived by subtracting the expected number of leftover packets from the total packet arrivals as follows. Let l_i be the number of leftover packets destined for output i, $1 \leq i \leq N$. Due to the symmetric statistical characteristics, $l_1, l_2, \ldots, l_N$ are random variables with the same distribution, although depending on each other. By probability theory, the expected number of leftover packets $E(L) = E(l_1 + l_2 + \cdots + l_N) = NE(l_1)$. With the steady-state distribution (α, β, γ) when $I = 1$, we have

$$E(l_1) = \sum_{\alpha=0}^{\gamma} \sum_{\beta=0}^{\gamma-\alpha} \sum_{\gamma=0}^{N} \pi(\alpha, \beta, \gamma)[\alpha - 1]^{+} \tag{2.10}$$

By subtracting leftover packets from the total packet arrivals, we have the cut-through ratio

$$E(C) = 1 - \frac{E(L)}{N \times \rho} = 1 - \frac{E(l_1)}{\rho} \tag{2.11}$$

Next, we explain how to modify the current aggregation model to adapt it to the OpCut switch under ON-OFF Markovian traffic.

Taking into consideration the dependency between arrivals in consecutive time slots, the state of the first I-queue block and the $(I+1)_{th}$ queue for $1 \leq I \leq N-1$ can be represented by five variables $(\alpha, \beta, \gamma, U, u)$, where α, β and γ denote the inputs state defined above, and U and u are the numbers of packets in first I queues and the $(I+1)_{th}$ queue, respectively. Since the state of inputs is completely independent of the number of packets currently stored in buffer, the transition rate from state $(\alpha_0, \beta_0, \gamma_0, U_0, u_0)$ to state $(\alpha_1, \beta_1, \gamma_1, U_1, u_1)$ is the product of the following two terms. (1) The transition rate from state $(\alpha_0, \beta_0, \gamma_0)$ to state $(\alpha_1, \beta_1, \gamma_1)$ for the given I. (2) The conditional probability that the $I+1$ queues will go from state (U_0, u_0) to state (U_1, u_1), given that there are α_0 packets for the first I queues and β_0 packets for the $(I+1)_{th}$ packets among the total of γ_0 packet arrivals, which can be found as follows. When $U_1 + u_1 < B$, no packets would have been dropped by "zero external interference assumption." The conditional probability is

$$\Lambda(U_1, u_1 | U_0, u_0, \alpha_0, \beta_0, \gamma_0) = \begin{cases} \sum_{L,O} CT_I(O|\alpha_0, \gamma_0, L, U_0) P(L_{\alpha_0}^{I} = L) \\ \qquad U_1 = U_0 - O + L \text{ and } u_1 = u_0 - o + l \\ 0 \qquad \text{otherwise} \end{cases}$$

where the number of buffered packets sent from the $(I+1)_{th}$ queue $o = \min\{U_0, \max\{1-\beta_0, 0\}\}$, and the number of leftover packets destined for the $I + 1_{th}$ queue $l = \max\{\beta_0 - 1, 0\}$. The conditional probability $CT_I(O|\alpha_0, \gamma_0, L, U_0)$ that O packets will be sent out from the first I queues in the given state $(\alpha_0, \gamma_0, L, U_0)$ can be obtained from the last iteration for $I \geq 2$. When $I = 1$, $CT_1(O|\alpha_0, \gamma_0, L, U_0)$ is 1 if

$O = \min\{U_0, \max\{1-\alpha_0, 0\}\}$ and 0 otherwise. When $U_1 + u_1 = B$, it is possible that the $(I+1)_{th}$ queue dropped packets before reaching the final state (U_1, u_1), thus the state transition probability is

$$\sum_{U_m, u_m} \Lambda(U_m, u_m | U_0, u_0, \alpha_0, \beta_0, \gamma_0) \times P_D(U_m - U_1, u_m - u_1; U_m, u_m)$$

where $\Lambda(U_m, u_m|U_0, u_0, \alpha_0, \beta_0)$ is given by Equation (2.12) and $P_D(U_m - U_1, u_m - u_1; U_m, u_m)$ is given by Equation (2.7). After obtaining the steady-state distribution of the I-queue block and the $(I+1)_{th}$ queue, we can aggregate the first $I+1$ queues into one block, whose state is represented by $(\alpha^*, \gamma^*, L^*, U^*)$, where $\alpha^* = \alpha + \beta$, $U^* = U + u$, $L^* = L + l$. Update the conditional probability $CT_{I+1}(O^*|\alpha^*, \gamma^*, L^*, U^*)$

in a similar way to the Bernoulli case. That is, separate all the sub-states and obtain the conditional probability based on each sub-state, and then sum them up. The difference is that there will be more states due to the introduction of the new variable γ. Repeat the iteration of the aggregation process till we have the steady-state distribution $\pi(\gamma, U)$ of the buffer and the conditional probability $CT_N(O|\gamma, L, U)$, which is the probability that O packets will be sent out from the buffer in this time slot, given that there are γ busy inputs, L leftover packets and U buffered packets of the N-queue block. The packet loss ratio and average packet delay can be derived in a similar way to Bernoulli traffic, where the average packet delay has the same expression as Equation (2.9) and the packet loss ratio is given by

$$\begin{aligned}\alpha &= \sum_{\gamma=0}^{N}\sum_{L=0}^{\gamma}\sum_{O=0}^{N}\pi(\gamma, U)CT_N(O|\gamma, L, U) \\ &\quad \times P(L_\gamma^N = L)[U + L - O - B]^+\end{aligned} \tag{2.12}$$

Note that as we need additional variables to represent the states of inputs in ON-OFF Markovian traffic, compared to Bernoulli traffic case, the state space of the aggregation model under ON-OFF is larger. The complexity of the aggregation model under ON-OFF Markovian traffic is $O(B^2N^4)$.

2.6 Aggregation Model for WDM Optical Cut-through Switch

So far we have considered the case that each fiber of the OpCut switch has a single wavelength. To fully utilize the huge bandwidth of optics, the bandwidth of an optical fiber can be divided into multiple wavelengths where each wavelength carries independent data [70], so that the system capacity is greatly increased compared to the single wavelength system. This is called *wavelength division multiplexing* (WDM). In this section, we show that the aggregation model can also be adapted to analyzing the performance of the WDM OpCut switch. We will mainly focus on the aggregation model for the WDM OpCut switch under Bernoulli traffic. The model for the WDM OpCut switch under ON-OFF Markovian traffic can be similarly derived, but is much more extensive. Thus, we omit it here. The WDM OpCut switch has N inputs/outputs, each of which has k wavelengths. An electronic buffer of size B is shared by all leftover packets, where N queues are maintained, with each corresponding to an output fiber. In the WDM OpCut switch, up to $N \times k$ packets may arrive in a time slot in optical format, one on each wavelength channel. Due to the use of wavelength converters, each output fiber can receive up to k packets in each time slot. At first glance, the WDM OpCut switch might be similar to WDM optical packet switches with shared buffer implemented by FDLs, which have been extensively studied, see, for example, [67, 72]. However, the WDM OpCut switch is fundamentally different from the switches with FDL buffer, because its electronic buffer is random access memory and can also have a large capacity, while the FDLs can only delay packets for a fixed length of time in a pipelined fashion and accommodate very limited number of packets, In addition, the queues in the shared electronic buffer interact in a completely different way from shared FDLs. The idea of aggregation can be adapted to the WDM OpCut switch. To do so, some modifications have to be made in order to accurately analyze the performance of the WDM OpCut switch. Different from the single-wavelength OpCut switch considered so far, if an output fiber has received s packets ($s \leq k$) during the cut-through phase in the WDM OpCut switch, it still can receive up to $k - s$ buffered packets from the corresponding queue later in that time slot. Only when there are more than k packet arrivals towards an output fiber in the same time slot, will the leftover packets be put into the corresponding queue in the shared electronic buffer. This unique characteristics of the WDM OpCut switch makes it necessary to reconsider the cut-through ratio and equations governing the queue behavior in the aggregation model. To find the expected cut-through ratio for the WDM OpCut switch, we first calculate the expected number of leftover packets then subtract it from the total packet arrivals. Define H_i as the number of packets destined for output fiber I, and L_I as the number of leftover packets for the I_{th} queue. The expectation of L_i for any i can be expressed as

$$E(L_I) = \sum_{h=k+1}^{N\dot{k}} (h-k)P(H_i = h)$$

Since packets under Bernoulli traffic have uniformly distributed destinations, H_I is a Binomial random variable $B(N \times k, \rho/N)$. Due to the symmetry of Bernoulli traffic, $E(L_i) = E(L_j), \forall i, j$, thus $E(L) = NE(L_i)$. After obtaining the expected number of leftover packets, the cut-through ratio of the WDM OpCut can be obtained.

$$E(C) = 1 - \frac{E(L)}{N \times k \times \rho} \tag{2.13}$$

Since the WDM OpCut switch can receive up to k packets at each output fiber in each time slot, we need to reconsider the characteristic of leftover packets. To describe the behavior of leftover packets, we define L_S^I as the number of leftover packets among S packet arrivals destined for the first I output fibers. When $I = 1$, the p.m.f. of L_S^1 can be expressed as

$$P(L_S^1 = l) = \begin{cases} 1 & l = \max\{0, s-k\} \\ 0 & \text{otherwise} \end{cases}$$

Then, we move on to the general case when $I \geq 1$. L_S^I can also be derived inductively as follows. First, let S_I denote the number of arriving packets for the first I output fibers. The conditional probability that there are h packets destined for output I among S arriving packets destined for the first I outputs $P(H_I = h|S_I = S)$ can be determined as follows

$$P(H_I = h|S_I = S) = \binom{S}{h}(1/I)^h(1-1/I)^{S-h}$$

The p.m.f of L_S^I for $I \geq 1$ can thus be derived inductively.

$$P(L_S^I = l) = \sum_{h=0}^{S} P(L_{S-h}^{I-1} = l - z)P(H_I = h|S_I = S)$$

where z is the number of leftover packets for the I_{th} queue, and $z = \max\{0, h-k\}$. The equation holds because the number of leftover packets towards the first I queues in a time slot is the sum of the number of leftover packets for the first $I-1$ queues and the number of leftover packets for the I_{th} queue. After obtaining the p.m.f. of leftover packets, the remaining differences between the WDM OpCut switch and the single-wavelength OpCut switch in mathematical terms are the equations governing the number of buffered packets sent from each queue. In the WDM OpCut switch, the number of buffered packets sent from a queue is $o = \min\{u, \max\{k-s, 0\}\}$, where u is the number of buffered packets in the queue, and s is the packet arrivals destined for the corresponding output fiber. S-ince the queues in the shared electronic buffer of the WDM OpCut switch interact with each other in the same way as those in the single-wavelength OpCut switch, the aggregation model can be readily adapted to the WDM case with the above modifications. The equations of the packet loss ratio and average packet delay for the WDM OpCut switch share the same expressions as those for the OpCut switch with a single wavelength, which are given by Equation (2.8) and Equation (2.9). Together with Equation (2.13), we now have a complete set of performance metrics for the WDM OpCut switch. Note that the state space of the aggregation model for the WDM OpCut switch under Bernoulli traffic is equal to the single-wavelength case. Thus, the complexity of this model is also $O(NB^2)$.

2.7 Validation of the Aggregation Model

In this section we will mainly focus on the validation of the proposed aggregation model via simulations. We have conducted extensive simulations for the OpCut switch of various sizes under different traffic load, and compare the simulation results with the analytical results obtained from the aggregation model. In this section, we present numerical results to illustrate how different parameters of the OpCut switch affect the switch performance under Bernoulli traffic and ON-OFF

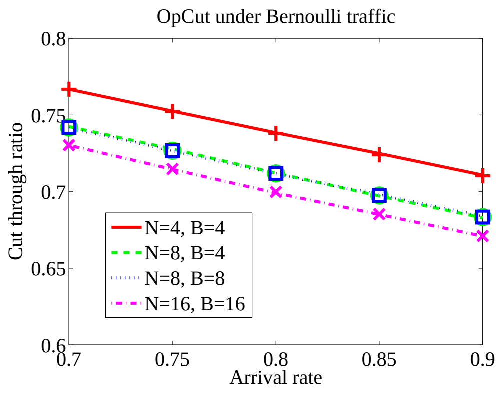
Markovian traffic in terms of cut-through ratio, packet loss ratio and average packet delay.

Since packets loss is a very rare event, we set the number of time slots in our simulation according to packet loss. If packet loss is in the order of 10^{-4} or higher, the simulation is run for 10^7 time slots. If the packet loss is lower than 10^{-4}, the simulation is run till it has encountered 10^3 lost packets. The simulation consists of two parts. The first part studies the performance of the single-wavelength OpCut switch under Bernoulli traffic and ON-OFF Markovian traffic, and the second part studies the WDM OpCut switch under Bernoulli traffic.

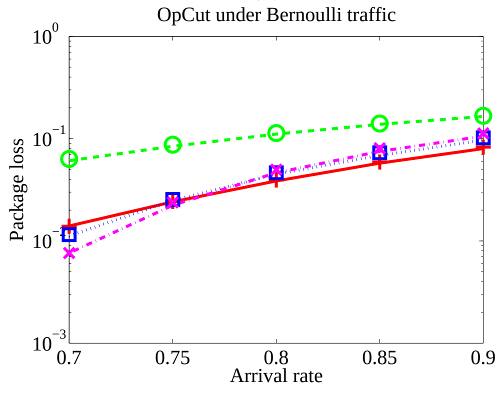
2.7.1 Single-Wavelength OpCut Switch

For the single-wavelength switch, We have conducted simulations for four different switch size N and buffer size B with $N = 4, B = 4$; $N = 8, B = 4$; $N = 8, B = 8$ and $N = 16, B = 16$, and obtained their cut-through ratio, packet loss ratio and average packet delay, respectively. Fig. 2.2 shows the performance with respect to different traffic load for the OpCut switch under Bernoulli traffic. First, note that the results obtained from the aggregation model almost overlap with simulation results in every testing scenario, indicating our model is indeed very accurate for both small and large switches. In addition to being accurate, it should be mentioned that the analytical model is orders of magnitude faster than the simulation: computing one point in Fig. 2.2 may take hours of simulation time, while it takes the aggregation model only a few seconds for the same computation.

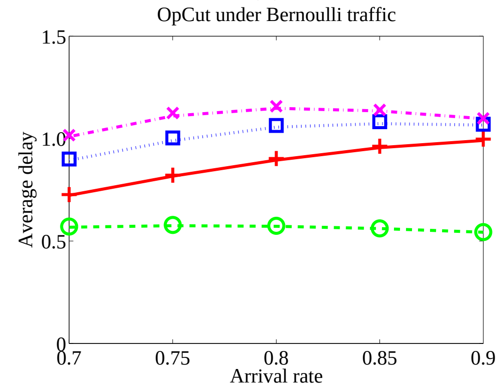
From Fig. 2.2(a), we observe that the cut-through ratio drops as traffic load increases, which is expected since with more packet arrivals at the same time, more contentions would occur at the output, leading to fewer packets cutting through. From Fig. 2.2(b) and (c), we can see that packet loss ratio and average packet delay slowly rise with the increase of traffic load. This is because that with more packets arriving, more leftover packets will be put into buffer, thus packets are more likely to be dropped due to buffer overflow. With longer queues, the average delay a packet experiences also increases. Notice that the average packet delay increases with traffic load at first and stops increasing until it reaches a certain level. The reason is that we count the average delay as the number of time slots for a packet to be sent to the output or dropped. When traffic is light, packets will be sent to buffer after waiting for a certain amount of time since buffer overflow is rare, while



(a)

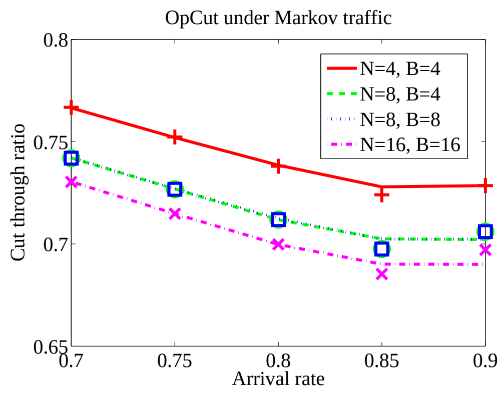


(b)

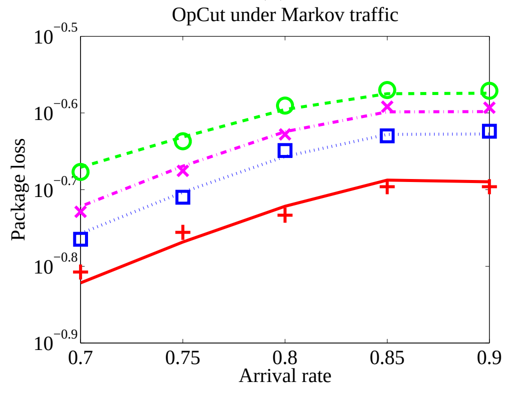


(c)

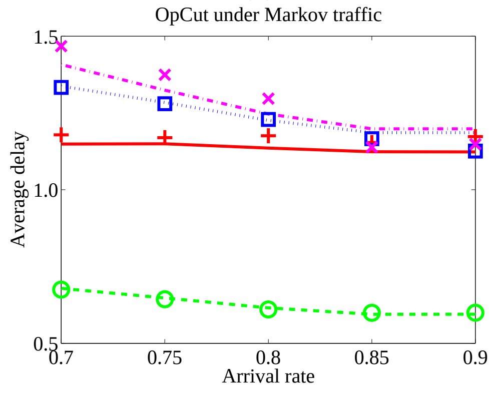
Figure 2.2: Simulation and analytical results for the OpCut switches of different switch and buffer sizes under Bernoulli traffic. (a) Cut-through ratio. (b) Packet loss ratio. (c) Average packet delay.



(a)



(b)



(c)

Figure 2.3: Simulation and analytical results for the OpCut switch of different switch sand buffer sizes under ON-OFF Markovian traffic. (a) Cut-through ratio. (b) Packet loss ratio. (c) Average packet delay.

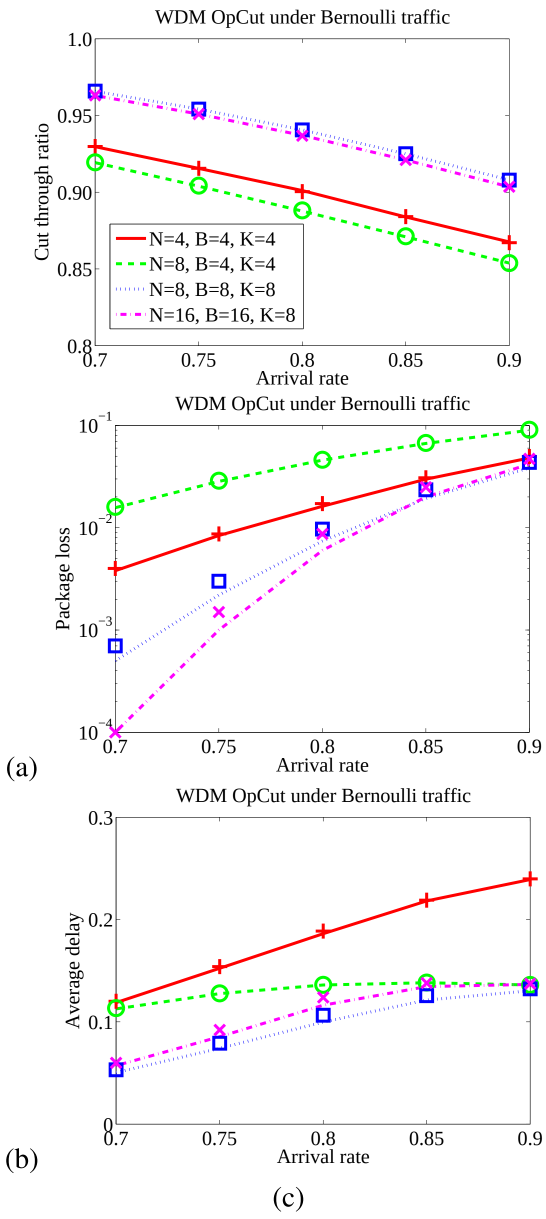


Figure 2.4: Simulation and analytical results for the WDM OpCut switches of different sizes (N) with different buffer space (B) and number of wavelengths (k) under Bernoulli traffic. (a) Cut-through Ratio. (b) Packet Loss Probability (c) Average Packet Delay.

packets are more likely to be dropped when traffic load is heavy. Overall, we can observe that packet loss ratio and average delay are kept at a very low level (packet loss ratio is approximately 10^{-2} and average packet delay is less than one time slot) under all traffic loads. This indicates that the OpCut switch can achieve ultra-low packet loss and average packet delay even under heavy traffic load.

Comparing the performance of the OpCut switches with the same switch size but different buffer size in Fig. 2.2(b) and (c) (for example, $N = 8, B = 4$ and $N = 8, B = 8$), we can see that a larger buffer size reduces packet loss ratio for a very limited amount. This is because that a large percentage of packets cut-through directly and only leftover packets will be sent to buffer, thus the buffer usage is kept at a low level even under heavy traffic load. This also indicates that the OpCut switch is a stable system and buffer overflow is a rare event. Larger buffer actually increases average delay when traffic load is heavy, which is due to that the average delay is calculated as the average number of time slots that a buffered packet has to wait till being dropped or sent to the output. Since a larger buffer leads to fewer dropped packets, the average delay increases.

Fig. 2.3 shows the performance with respect to different traffic loads for the OpCut switch under ON-OFF Markovian traffic. The average burst length, which is the consecutive number of packet arrivals at each input, is set to 5 in our simulation.

From Fig. 2.3, we can again observe a near-perfect agreement between simulation results and analytical results of the model for all switch sizes and traffic loads. Note that in the figure cut-through ratio decreases with traffic load, while packet loss and average packet delay increases with traffic load, for a similar reason to the Bernoulli traffic case. Comparing the performance of the OpCut switch under Bernoulli traffic and Markovian traffic, we observe that ON-OFF Markovian traffic has higher packet loss and longer average packet delay under the same traffic load. For example, when $N = 4, B = 4$ and $\rho = 0.8$, packet loss ratio under ON-OFF Markovian traffic is higher than that under Bernoulli traffic by about 10^{-1}, and average delay is longer by half a time slot. The increase in packet loss is due to buffer overflow caused by the burstiness of Markovian traffic. This difference of packet loss between two types of traffic becomes more evident when traffic load is heavier. This is because that when traffic is light, buffer is under-utilized, and bursty traffic usually will not cause buffer overflow. In contrast, when traffic load is heavy, buffer storage is close to its limit, thus bursty packets are more likely to be dropped. The reason why packets have longer average delay under ON-OFF Markovian traffic is that instead of evenly distributed among queues as Bernoulli traffic, leftover packets tend to concentrate among a few queues during a certain period of time due to the dependency of packet arrivals in consecutive time slots. Hence, under ON-OFF Markovian traffic, the queues in the shared buffer tend to be unbalanced, leading to longer average packet delay.

2.7.2 WDM OpCut Switch

In this subsection, we verify the effectiveness of our model for the WDM OpCut switch under Bernoulli traffic. We consider the WDM OpCut switch of different switch and buffer sizes, as well as different number of wavelengths on each input/output: $N = 4, B = 4, k = 4$; $N = 4, B = 4, k = 8$; $N = 8, B = 8, k = 4$ and $N = 16, B = 16, k = 4$. Fig. 2.4 plots the three performance metrics of the WDM OpCut switch: cut-through ratio, packet loss ratio and average packet delay, respectively. Both analytical results and simulation results are presented with respect to traffic load. First, we note a good agreement between the analytical model and the simulation for the WDM OpCut switch of different sizes and number of wavelengths, indicating the aggregation model works well in modeling the performance of the WDM OpCut switch. Comparing the performance of the single-wavelength switch with that of the WDM switch under Bernoulli traffic, we can see that the WDM switch performs better in terms of all three performance metrics, We also compare the performance of the WDM switch with $N = 4, B = 4, k = 4$ with that with $N = 4, B = 8, k = 8$. In general, a larger number of wavelengths k on each fiber improves the performance of the switch. The reason is that despite that the number of arriving packets increases with k under the same traffic load, more wavelengths mean larger capacity at each output, which allows more packets to be sent out from each output in a time slot, leading to better performance.

2.8 Conclusions

In this chapter we have presented an analytical performance model called the aggregation model for the WDM OpCut switch, an optical packet switch with shared electronic buffer. The aggregation model resolves the strong dependency among queues in the shared buffer by inductively aggregating more queues into a block, achieving a low polynomial complexity to the switch size. We applied the aggregation model to the single-wavelength OpCut switch under both Bernoulli and ON-OFF Markovian traffic. Then we extended the model to the WDM OpCut switch, where there are multiple wavelength channels on each input/output. Effectiveness of the aggregation model is validated by comparing the simulation results with analytical results. It is shown that the aggregation model can achieve near-perfect accuracy in approximating various performance metrics of the OpCut switch in all testing scenarios regardless of switch size and traffic type.

Chapter 3

Low-Latency Multicast Scheduling in Optical Packet Switches

This chapter presents our results on high-throughput, low-latency and scalable multicast traffic scheduling in all-optical packet switches. The previously discussed hybrid electronic/optical interconnects require optical-to-electronic-to-optical (OEO) conversion, which leads to undesirable packet delay, power consumption and additional cost in high speed switching. On the other hand, fiber-delay-line (FDL) [78] provides a viable solution to store optical packets due to its transparency to traffic bit-rate and low power dissipation. We first propose a novel FDL buffer called multicast-enabled fiber-delay-lines (M-FDLs), which can provide flexible delay for copies of multicast packets using only a small number of FDL segments. We then present a *Low Latency Multicast Scheduling (LLMS) Algorithm* that considers the schedule of each arriving packet for multiple time slots. We show that LLMS has several desirable features, such as a guaranteed delay upper bound and adaptivity to transmission requirements. To relax the time constraint of LLMS, we further propose a pipeline and parallel architecture for LLMS that distributes the scheduling task to multiple pipelined processing stages, with N processing modules in each stage, where N is the size of the interconnect. Finally, by implementing it with simple combination circuits, we show that each processing module can complete the packet scheduling for a time slot in $O(1)$ time. The performance of LLMS is evaluated extensively against statistical traffic models and real Internet traffic traces, and the results show that the proposed LLMS algorithm can achieve superior performance in terms of average packet delay and packet drop ratio.

The rest of this chapter is organized as follows. Section 3.1 presents the introduction and related work. Section 3.2 presents the architecture of the adopted all-optical multicast interconnect and optical buffer. Section 3.3 describes the details of the all-optical multicast scheduling (LLMS) algorithm. Section 3.4 presents the pipeline and parallel technique and the corresponding hardware implementation that reduces the time complexity of LLMS. Section 3.5 presents the performance evaluation results. Finally, Section 3.6 concludes the chapter.

3.1 Introduction and Related Work

Driven by emerging applications requiring high-bandwidth transmission from one source to multiple destinations, such as video conference, video-on-demand (VoD) and IP-based Television (IPTV) [75], optical multicast switching has attracted much research effort. A series of all-optical switching architectures and techniques have been proposed to support multicast at the interconnect/router level, such as wavelength-assisted switching [76, 77], Broadcast-and-Select (BS) switching [79, 80], and wavelength-division-multiplexing (WDM) switching [81–83], etc. However, despite the considerable amount of work on multicast-capable optical packet switching architectures, relatively little attention has been paid to multicast packet scheduling in such interconnects, which is critical for high-speed all-optical packet interconnects. Motivated by this observation, in this chapter we study multicast scheduling in all-optical packet interconnects/switches.

Since a practical "optical RAM" able to mimic the buffers used in electronic interconnects is still not available currently, how to resolve *output contention*, which occurs when multiple optical packets simultaneously go to the same output, poses a serious challenge for multicast scheduling in optical packet interconnects. Various contention resolution techniques have been proposed [61]. Bufferless approaches such as wavelength conversion and deflection routing [61] resolve contentions by sending conflicted packets to different wavelengths or other outputs. However, these approaches have been found ineffective for avoiding packet loss under congested network conditions and demanding a lot of network resources.

Multicast scheduling in optical packet interconnects with FDL buffer is fundamentally different from the well-studied multicast scheduling in electronic interconnects [84, 85, 89, 90], for the reason that all the approaches for electronic interconnects rely on electronic RAM to resolve output contention, while FDLs can merely delay packets for a fixed period of time. Two major challenges must be properly addressed in multicast scheduling with FDL buffer. First, since FDL is very bulky in general, only a few can be used in a single optical interconnect. Therefore, efficient optical buffer is needed to avoid the performance degradation resulted from limited buffering capability of FDL. Second, scheduling in optical packet interconnects requires electronic processing that involves calculating the delay for each packet in the FDL buffer and configuring the switching fabric, where a complex scheduler may pose a bottleneck in high-speed switching. For example, a scheduler with $O(N)$ time complexity, where N is the interconnect size, requires the electronic processing component to run N times faster than the port speed normalized by the packet length on the fiber. Suppose we have an optical packet switch with 100 Gbit/s port speed and the packet length is 64 bytes. Then, the number of packets arriving at each input port would peak at around 200 million per second (i.e., each time slot lasts around 5 ns). To process packets at such a rate, a scheduler with $O(N)$ time complexity would have to work at a clock frequency much higher than state-of-art FPGAs can accommodate even for small switches with $N = 8$ ports. This means that we will face a serious scalability problem as the port count and port speed increase. It is therefore critical to design a scheduler of lower time complexity for high speed optical packet interconnects.

Most existing multicast scheduling schemes require $O(N)$ time complexity. Wavelength-assisted routing [76, 77] is a commonly used multicast scheduling scheme, in which each multicast packet is sent to a multicast module, a FDL loop device used to generate copies of the packet and provide necessary delay. However, wavelength-assisted routing based schemes are generally quite complex and provide only limited multicasting ability, as each multicast module cannot be shared by multiple packets simultaneously. Moreover, wavelength-assisted routing cannot provide delay guarantee since a packet may have to be recirculated many times in the multicast module before being sent out. Output buffering [91] is another widely adopted scheme. In [91], a buffer consisting of an $N \times B$ switch and B FDL segments is placed at each output of an $N \times N$ interconnect. The length of these FDL segments increases linearly, in which the shortest segment is able to delay optical signal for one time slot and the longest one can delay optical signal for B

time slots. During each time slot, the scheduler assigns packets to FDL segments with proper delay in each output buffer, such that they will exit the optical buffer at different times. Though output buffering delivers optimal performance in terms of average delay and packet drop ratio, the stringent hardware requirement makes output buffering unscalable for large interconnects.

To avoid performance bottleneck caused by slow scheduling in high speed electronic and optical switches, many previous work focused on designing scheduler architectures with low time complexity through parallel and/or pipeline processing [92–94]. For example, [92] presents a multi-processing scheduler with $O(N)$ time complexity for input-queued electronic switches, which uses multi-input-queue (MIQ) and parallel arbitration to speedup the scheduling process. A scheduler for output-queued switches was proposed in [93], which achieves $O(\log^2 N)$ time complexity using a parallel prefix-sum operation. Its time complexity was further reduced to $O(1)$ in [91, 94] through a pipeline processing architecture consisting of $(\log_2 N + 1)$ pipeline stages. However, the pipeline processing incurs $O(\log_2 N)$ delay overhead.

In this chapter, we systematically address the above challenges in multicast scheduling for high speed optical packet interconnects. Our contributions can be summarized as follows.

- We propose an efficient optical buffer called multicast-enabled FDLs (M-FDLs) that enables flexible packet duplication and controllable delay using much shorter FDL segments than the incremental buffer [91]. Such optical buffer is very helpful to multicast scheduling in optical packet interconnects.

- Using the M-FDLs buffer, we present an algorithm called *Low Latency Multicast Scheduling (LLMS) Algorithm* for all-optical packet interconnects. By considering the schedule of each arriving packet for multiple time slots, LLMS allows more efficient packet transmission than scheduling algorithms that resolve output contention for a single time slot. LLMS can also immediately detect the congestion and promptly drop packets with overlong delay to let upper layer protocols quickly respond to the network condition. Such a feature is desirable for delay-sensitive multicast applications. LLMS is able to achieve the performance very close to the optimal performance of output buffering [91] in terms of average delay and packet drop, while requiring

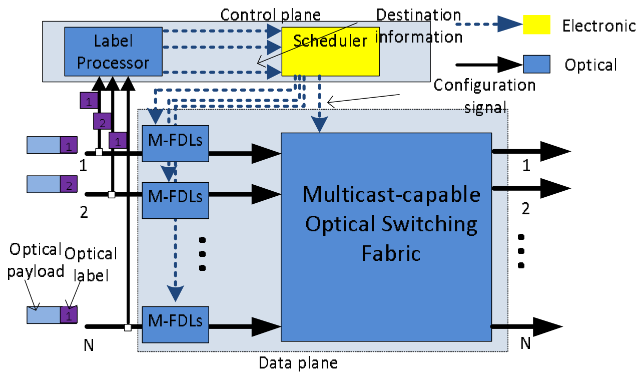


Figure 3.1: Architecture of a single-wavelength, input-buffered $N \times N$ optical multicast packet interconnect.

orders of magnitude shorter FDL segments.

- We propose a pipeline and parallel processing architecture that distributes the scheduling task to multiple pipelined stages, with N processing modules operating in parallel in each stage. Combined with a simple combination circuit implementation, the time complexity for each processing module can be reduced to $O(1)$. The proposed architecture enables the switch to schedule packets at the line rate, and solves the dilemma that the processing speed of electronic scheduler struggles to catch up with the ever-increasing port speed in optical packet interconnects. Also, it does *not* incur additional pipeline overhead, which is very desirable compared to the scheduler proposed in [91, 94] that has $O(\log_2 N)$ pipeline overhead.

- The LLMS algorithm can be easily extended to provide differentiated Quality-of-Service (QoS), which presents a desirable extra feature. We show that the prioritized LLMS scheduling, though based on a simple preemptive strategy, achieves good QoS differentiation in traffic and, more importantly, can be implemented by the pipeline and parallel processing architecture with very little modification.

3.2 Interconnect Architecture and Buffer Management

In this section, we briefly describe the adopted interconnect architecture and the operation of the proposed optical buffer, multicast-enable FDLs (M-FDLS).

3.2.1 Interconnect Architecture

We consider a simple single-wavelength, input-buffered optical multicast packet interconnect, whose high level view is depicted in Fig. 3.1.

We assume the interconnect operates in a time-slotted manner and uses optical packets of the same duration with low bit rate headers to facilitate processing at the scheduler. Each optical packet consists of two parts: payload and label (or header). The optical label contains the destination outputs of the packet, and is much shorter than the optical payload. It is also encoded at a low fixed bit rate to allow easy optoelectronic conversion and electronic processing. The payload duration is fixed to a time slot, such that its data volume is proportional to the user-defined bit-rate ranging from Mbs per second to hundreds of Gbs per second. In an all-optical packet switched network, the payload of each packet transmits across the network transparently, and is only electronically recovered at end points.

The adopted interconnect consists of the optical switching fabric and M-FDLs as input buffers in the data-plane, and optical label processors and electronic scheduler in the control plane. When an optical packet arrives at an input port, the input port aligns the incoming packet related to the switch master clock in order to synchronize packet flows, which is necessary for packet header recovery. Then, the packets label is stripped off and sent to the label processor, which can be performed passively by the optical correlation technique. The label processor then converts all-optical headers to electronic form, and sends them to the electronic scheduler, which calculates the schedule for each packet. Based on scheduling results, control signals are issued to the FDL buffers and switching fabric to properly configure the interconnect. Finally, updated headers are reinserted, and the packets are sent out of the switch.

Similar to many existing work in the literature, such as European KEOPS switch [79, 80], we adopt the broadcast-and-select optical switching fabric. Since coupler-

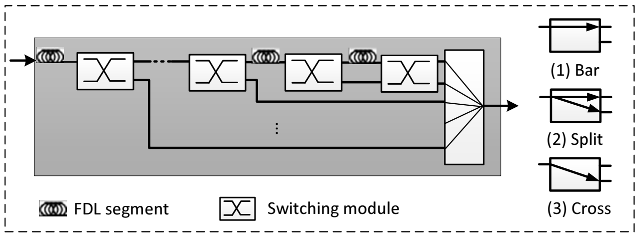


Figure 3.2: Multicast-enabled FDLS (M-FDLs). Left: Structure of M-FDLs. Right: Three possible states of switching modules: bar, split and cross.

s and SOAs inevitably introduce signal distortion, after packets are delivered to corresponding output ports, they will go through proper regeneration (e.g., reamplification and reshaping) to reduce signal degradation [79].

3.2.2 Buffer Management

Next, we present a novel optical buffer called multicast-enable FDLs (M-FDLs) that provides flexible delay for each incoming multicast packet. Fig. 3.2 shows the buffer structure. The M-FDLs buffer consists of cascaded *unit-length* FDL segments. Each unit-length FDL segment can provide a delay of T, which is the duration of a time slot. To provide flexible delays ranging from T to dT, a total of d FDL segments are needed.

The FDL segments are connected by 1×2 *switching modules*. To support controllable delay and flexible packet duplication for multicast packets, each switching module can be set to one of three possible states: when it is in "bar" state (the default state), packets simply go through it and move to the next FDL segment; when it is in "split" state, a copy of packet will be sent to the interconnect for transmission through an optical multiplexer, while the packet continues to move forward in the M-FDLs; when it is in "cross" state, packets will move out of the M-FDLs completely and be sent to the interconnect for transmission.

Duplicating packets in a switching module causes considerable split power loss, which is an important concern in the design of M-FDLs. Here, we borrow some ideas from the design of switching cells in multicast optical cross point switches (OXS) [95], and present an implementation of switching modules that uses active vertical couplers (AVC) to achieve lossless packet duplication.

As shown in Fig. 3.3, a switching module is built using two AVCs (called

AVC1 and AVC2) perpendicularly intersecting each other, which are formed using a light-amplifying active waveguide layer grown on top of the passive waveguide. When a switching module is at "bar" state, optical signal simply passes through the bottom passive waveguide with negligible insertion loss and SNR degradation, as shown in Fig. 3.3(a). When a switching module is at "cross" state, optical signal is coupled into the upper active waveguide. By adjusting the injecting carrier density at a proper level, the signal power is completely transferred from AVC1 to AVC2, amplified in the process, and as a result, the packet exits the M-FDLs buffer completely, as shown in Fig. 3.3(b). Finally, when a switching modules set to "split" state, the optical signal is coupled into active waveguide with about equal parts of input signal power at the end of both couplers, as shown in Fig. 3.3(c). The gain of the active waveguide is set to be sufficient to overcome the split loss, therefore realizing lossless packet duplication.

Even though insertion loss caused by packets splitting is no longer an issue, each splitting operation decreases the optical signal-to-noise ratio (OSNR) of a packet. However, the noise accumulation is very slow, since a high OSNR of 25 dB can still be achieved after more than 40 stages of splitting [95]. This is sufficient for current multicast optical packet switches, because a packet can be split at most N times, where N is the switch size, and as mentioned before, the size of multicast optical switching fabrics is limited in practice.

We now use an example to illustrate how the M-FDLs works. Assume that a multicast packet arrives at time slot t, and is scheduled to deliver a copy of it to some of its destination outputs in the $(t + i)^{th}$ time slot and to the rest of its destination outputs in the $(t + j)^{th}$ time slot ($i < j \leq d$). At the beginning of the $(t + i)^{th}$ time slot, the packet moves to the i^{th} coupler, and the scheduler sets the i^{th} coupler to "split," such that a copy of the packet will be sent to the switching fabric. At the beginning of the $(t + j)^{th}$ time slot, the scheduler sets the j^{th} coupler to "cross" state, thus the packet moves out of the M-FDLS and be transmitted completely.

Compared with existing FDL buffers for multicast packet switching, M-FDLs has some clear advantages. On one hand, M-FDLs does not have the problem of limited multicast processing capability in the recirculating loop buffer in the wavelength-assisted routing scheme [76, 77], since each M-FDLs can be shared by all the incoming multicast packets in a pipelined fashion. On the other hand, M-FDLs can achieve the same buffer depth using much shorter FDL segments,

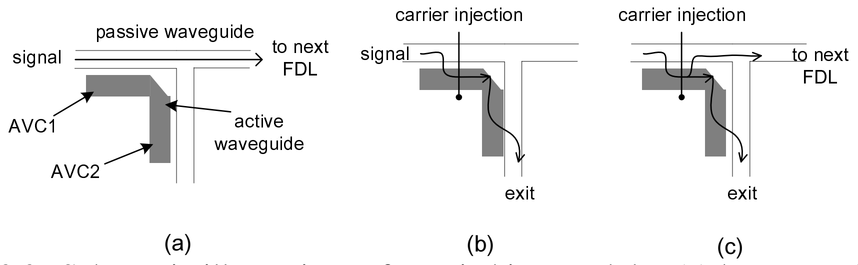


Figure 3.3: Schematic illustrations of a switching module: (a) bar state; (b) cross state; (c) split state.

compared to the incremental buffer used in the output buffering scheme [91]. For example, as mentioned earlier, to achieve flexible delays ranging from $1T$ to $30T$, an incremental optical buffer in [91] requires FDL segments with a total length that can delay packets for 465 time slots, while each M-FDLs only requires 30 unit-length FDL segments, which is a substantial saving. Based on the input-buffered optical multicast packet interconnect and M-FDLs buffer described above, we will present the Low Latency Multicast Scheduling (LLMS) Algorithm in the next section.

3.3 Low Latency Multicast Scheduling (LLMS)

3.3.1 Preliminaries

In this subsection, we introduce some commonly used terms in multicast scheduling. In multicast scheduling, the vector of destinations of a multicast packet is called its *fanout*. For clarity, an arriving *input packet* is usually distinguished from its corresponding *output copies*, i.e., the copies of the input packet destined for the outputs in its fanout.

The most straightforward multicast solution was the use of copy networks, in which all output copies are delivered by unicast. However, since optical switching fabric such as Broadcast-and-Select (BS) has a *intrinsic multicasting capability*, i.e., the ability to transmit packets from one input port to multiple output ports simultaneously, treating multicast as multiple unicasts wastes bandwidth and prolongs packet delay.

There are several service disciplines to transmit multicast packets from input ports to output ports, which can be roughly divided into two categories: *one shot* and *fanout splitting*. With one shot, all the output copies of an input multicast packet must be sent to the corresponding output ports in one time slot, whereas in fanout splitting, a multicast packet could be delivered to the outputs in multiple time slots, and in each time slot, only some of the outputs in its fanout receive the packet copy. It has been shown that one shot discipline may severely limit the throughput of the interconnect. Thus, we adopt fan-out splitting discipline.

3.3.2 General Description

In this subsection, we give a general description on how the Low Latency Multicast Scheduling (LLMS) algorithm works.

As mentioned earlier, the optical packet interconnect we consider is input-buffered, where a M-FDLs is placed in each input port. Each M-FDLs is capable of providing flexible delay within a range as well as producing duplicated copies for each entering packet. On the other hand, there can be at most one packet exiting from each output in each time slot. Hence, the schedule in each time slot must be contention-free, that is, no more than one packets are sent towards the same output port in a time slot. Therefore, the main objective of the proposed scheduling algorithm is to interleave arriving packets onto contention-resolved schedules in one or more future time slots, such that they are delivered with low transmission latency.

According to the operations of the adopted interconnect, the label of an arriving packet will be sent to the scheduler to be processed the moment the packet enters the corresponding M-FDLs. Every packet has to go through the first FDL segment inside the corresponding M-FDLs buffer. When a packet reaches the first coupler, it has the option to send a copy to the interconnect or exit the buffer. In this way, all arriving packets will be delayed for at least a period of T (where T is the time slot length), which allows the scheduler to make proper decisions. We adopt a centralized scheduler, which keeps track of all the arriving packets in the current time slot and all the packets currently inside M-FDLs. The scheduler makes the following decisions for each arriving packet: (1) in which future time slot will the packet be sent to the switching fabric; (2) if it is to enter switching fabric, which outputs the packet will be delivered to, such that no output contention could occur. Then, according to the schedule, the scheduler sends coordinated control signals to the switching fabric and M-FDLs for contention-free packet transmission. For example, suppose a packet is scheduled to be delivered to a subset of its destination output ports t time slots after it enters the M-FDLs. Then, right before the packet reaches the t^{th} coupler of the M-FDLs, the corresponding coupler duplicates the packet and a copy is sent to the switching fabric, which is properly configured to deliver the packet to the corresponding output ports.

To ensure low packet latency, the scheduler adopts a greedy strategy that delivers the output copies of a packet as early as possible. If a packet cannot be delivered to all its destination output ports before it reaches the end of the M-FDLs, the remaining output copies will be dropped. We should also consider the limitation of the switching fabric, which can send up to one packet from each input port to outputs. Therefore, the scheduler should prevent multiple packet copies from entering the same input port simultaneously, which can be done by setting the constraint that at most one packet copy is allowed to exit each M-FDLs in a time slot. Next, we describe the implementation details of the LLMS algorithm.

3.3.3 Implementation Details

In this subsection, we describe in detail the Low Latency Multicast Scheduling (LLMS) algorithm.

Consider an interconnect of size $N \times N$ as shown in Fig. 3.1, and assume that packets arrive at the beginning of each time slot. LLMS considers the schedule for the next D time slots by keeping D scheduling vectors, indexed by $1, 2, \ldots, D$, with each vector corresponding to the scheduling results in a future time slot. Note that D cannot be larger than the maximum delay each M-FDLs can provide. For example, a scheduling vector of index t is denoted by S_t, which is used for keeping track of scheduling results of the time slot that is t time slots after the current time slot. S_1 is used to record scheduling results of the next time slot. A scheduling vector has N entries, indexed by $1, 2, \ldots, N$, with each corresponding to an output. The o^{th} entry of S_t is denoted by $S_t(o)$. If a copy of some packet for output o is scheduled to be transmitted in the t^{th} time slot, we say that output copy o of the packet is *assigned* to entry $S_t(o)$.

Each entry can be represented by a four-tuple $(full, input, location, split)$, where the one-bit field $full$ is set to 1 if this entry has been assigned, otherwise it is 0. $input$ is used to record the corresponding input index of the packet in that entry. $location$ is used to record the index of the scheduling vector that the copy is assigned to *initially*, while the $split$ field is used to configure the state of couplers in M-FDLs. For example, suppose an output copy of some packet is assigned to the t^{th} scheduling vector, then the $location$ field of the assigned entry is set to t. If the packet still has leftover output copies that remain to be scheduled, then the $split$ field of the assigned entry is set to 1, which indicates that a copy will be created by setting the t^{th} coupler to "split" after t time slots while the packet continues to move along the M-FDLs afterwards. Otherwise, the $split$ field is set to 0, indicating that the t^{th} coupler will be set to "cross" and the packet will exit the M-FDLs after t time slots.

Since the switching fabric can only transmit up to one packet from each input to output ports within a time slot, at most one packet can come out of the same M-FDLs in one time slot. Also, each output can receive at most one packet in each time slot to avoid output contention. Therefore, an entry of index o in a scheduling vector is said to be *eligible* for an output copy of some packet from input i if and only if all the following three conditions are met.

1. The entry is not full, i.e., no packet has been previously assigned to this entry.
2. The output copy is destined for the o^{th} output.
3. No packets from input i have been previously assigned to this scheduling vector, such that at most one packet from the same input is scheduled to be transmitted in the same time slot.

To ensure that the third condition is satisfied, we use D one-bit mask vectors of length N, each corresponding to one scheduling vector. The i^{th} entry of the t^{th} mask vector is denoted as $M_t(i)$, which is set to 1 if some packet from the i^{th} input has been assigned to the t^{th} scheduling vector. The scheduler will check mask vectors before assigning output copies to ensure no packet from the same input has been previously assigned to this scheduling vector, and all the copies with the same input index in each scheduling vector are copies of the same packet.

In order to reduce the delay each packet experiences in buffers, the basic operation of the scheduler is to find the *earliest possible* eligible entries for arriving packets in each time slot. To prevent packets of an input port with a smaller index

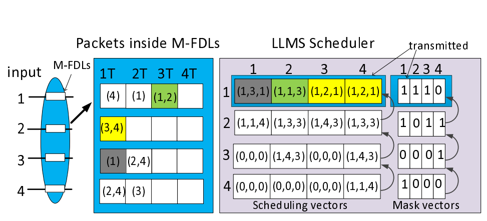
Table 3.1: Low Latency Multicast Scheduling (LLMS)

```
Input to LLMS: Arriving packets $P$, scheduling vectors $S$,
mask vectors $M$, priority register $pr$
// Packet Transmission
Configure couplers and switching fabric according to
the $1_{st}$ scheduling vector $S_1$
For $t = 2$ to $D$ Do
    $S_{t-1} = S_t$;
    $M_{t-1} = M_t$;
EndFor
Clear $S_D, M_D$;
// find the earliest eligible entries for the arriving packets
For packet $P_i$ from input $i$,
$i = [pr, pr \mod N + 1, (pr + 1) \mod N + 1, \cdots ,$
$(pr + N - 2) \mod N + 1]$. Do
  For scheduling vector $S_t, t = [1, 2, \ldots, D]$ Do
    For each output $o$ in $P_i$'s fanout, Do
      If $S_t(o)$ is empty and $M_t(i) == 0$
        Assign the entry to the output copy of $P_i$;
      EndIf
    EndFor
    If some copies of $P_i$ get assigned in $S_t$
      set $M_t(i) = 1$;
    EndIf
  EndFor
  If there are still outputs in $P_i$'s fanout left undelivered
      Drop these output copies of $P_i$;
  EndIf
EndFor
```

from always being scheduled earlier, the round robin policy is used to allocate priority to the packets arriving at different inputs to be scheduled in each time slot. We choose round robin policy for two reasons. Firstly, as the packet that is scheduled first will generally be transmitted with shorter delay than others, we need to ensure that all packets receive similar performance regardless of their inputs. Round robin policy has been widely applied in switch scheduling and shown to be effective in maintaining fairness. Secondly, as will be shown later, round robin can be readily implemented in hardware using a shuffle network, which does not incur extra time cost.

To indicate which input has the highest priority, a register pr is used. We update pr according to a cyclic-priority rule, i.e., the value of pr changes to $(pr + 1) \mod N$ at the end of each time slot. The scheduler checks the optical label of each packet in the order of their input index $[pr, pr \mod N + 1, (pr + 1) \mod N + 1, \cdots, (pr + N - 2) \mod N + 1]$, and tries to assign their output copies to the earliest eligible entries among D scheduling vectors. For example, consider a switch with size $N = 4$ and current priority $pr = 2$. Then, the arriving packet from input 2 is scheduled first. Next, packets from inputs $3, 4$ and 1 will be scheduled one by one. pr is updated to 3 in the next time slot, then $4, 1, 2 \ldots$ in the subsequent time slots. When an output copy cannot be assigned after searching all D scheduling vectors, it will be dropped by the scheduler. Note that some copies of the packets arriving later can be scheduled to be transmitted prior to copies of the earlier packets if there is no output contention, to reduce the average delay. It is easy to see that the time complexity of LLMS is $O(N^2D)$ in the worst case, where N is the interconnect size and D is the number of scheduling vectors.

At the beginning of each time slot, the scheduler configures the corresponding couplers in the M-FDLs buffers and switching fabric for packet transmission, according to the first scheduling vector S_1. Next, the scheduler shifts all the scheduling vectors and mask vectors forward by one position, i.e., the content of S_t is moved to $S_{t-1}, t = 2, 3, \ldots, D$. Then, the last scheduling vector and mask vector is emptied, because packets existing in M-FDLs buffers would have been transmitted within next $D - 1$ time slots. Finally, the scheduler starts the scheduling process as described above for the current time slot. For example, assume that entry $S_1(2)$ has the value of $(1, 1, 3, 1)$, which indicates that the packet entered the M-FDLs three time slots ago from the 1_{st} input and now reaches the 3^{rd} coupler (because


(a)

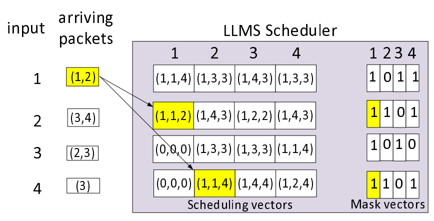

(b)

Figure 3.4: A scheduling example for a 4×4 interconnect. (a) The output copies corresponding to S_1 are transmitted (marked by blocks of the same color), then the scheduler rotates the scheduling vectors and mask vectors. (b) At the beginning of the next time slot, the scheduler schedules all arriving packets.

the scheduling vectors are shifted forward by one position each time slot). The scheduler sets the 3^{rd} coupler to "split," and connects input 1 with output 2 at the beginning of the time slot, such that a copy of the packet is delivered to output 2. The coupler will be reset to the default state "bar" after the packet goes through. Clearly, the delay of any transmitted output copy is bounded by the number of scheduling vectors D in LLMS. The detailed description of LLMS is given in Table 3.1.

3.3.4 A Scheduling Example

A scheduling example of LLMS algorithm for a 4×4 interconnect is shown in Fig. 3.4. The number of scheduling vectors D is set to 4. Packets are denoted by their fanouts, e.g., the packet destined for outputs 3 and 4 is denoted as $(3, 4)$. The packets in M-FDLs are denoted by the time they have been delayed, and the maximum delay each M-FDLs provides is $4T$, where T is the length of a time slot. Entries in scheduling vectors shown in the figure are 3-tuple recording the $(full, input, location)$ information of the assigned packets. The $split$ field is omitted here, as it does not participate in the scheduling process.

The initial content of scheduling vectors and mask vectors at the beginning of the current time slot is depicted in Fig. 3.4(a). Note that LLMS allows the packets arrived later to be transmitted before the packets arrived earlier, thus eliminates the Head-of-Line (HOL) blocking. For example, packet (1) from input 3 arrived one time slot later than packet $(2, 4)$, yet is scheduled to be transmitted earlier. Such a feature enables more efficient buffer management and reduces packet delay.

In the meanwhile, it is worth mentioning that in-order transmission of packets in the same flow (i.e., packets sharing the same input and fanout) is guaranteed in LLMS. For example, suppose packet A arrives at some input port with a previous packet B in the same flow still in the M-FDLs buffer. We can prove by contradiction that any output copy of A cannot be scheduled before B in LLMS, given that the output copy of B has not been dropped. Assume that an output copy of packet A is assigned an eligible entry that is earlier than the corresponding output copy of B. Since each eligible entry for packet A would also be eligible for B as they belong to the same flow, such a scheduling result contradicts with the scheduling procedure of LLMS, which always assigns the *earliest eligible* entries for each packet among the scheduling vectors. Therefore, we can see that in-order transmission is guaranteed in LLMS, as it is impossible to deliver a later packet prior to its predecessor in the same flow.

The scheduler then configures the interconnect and M-FDLs according to the first scheduling vector, and rotates the scheduling vector and mask vector forward by one position, as shown in Fig. 3.4(a). Packets transmitted completely will be removed from the buffer, while those with remaining fanout stay in the M-FDLS and will be delayed by another T. Packet (1) from input 3 (grey blocks) and packet $(3, 4)$ from input 2 (yellow blocks) will be transmitted completely, while packet $(1, 2)$ from input 1 (green blocks) will send one copy to output 2, and stays in the M-FDLs for future transmission. Assume the current priority indicator pr is 1. The scheduling results of the arriving packets are shown in Fig. 3.4(b). Take packet $(1, 2)$ (yellow block) as an example. Its two output copies are scheduled for transmission in the 2^{nd} and 4^{th} scheduling vectors, respectively. The packet will reach the second coupler when the 2^{nd} scheduling vector is rotated to the front, and accordingly the scheduler will change the 2^{nd} coupler to "split" and connects input 1 with output 1, such that a copy of the packet will be delivered to output 2. The packet will move out of M-FDLs in four time slots when all its output copies are transmitted.

3.3.5 Prioritized LLMS

In practice, Internet traffic consists of flows with different Quality-of-Service (QoS) requirements. For example, some video-on-demand service providers offer good

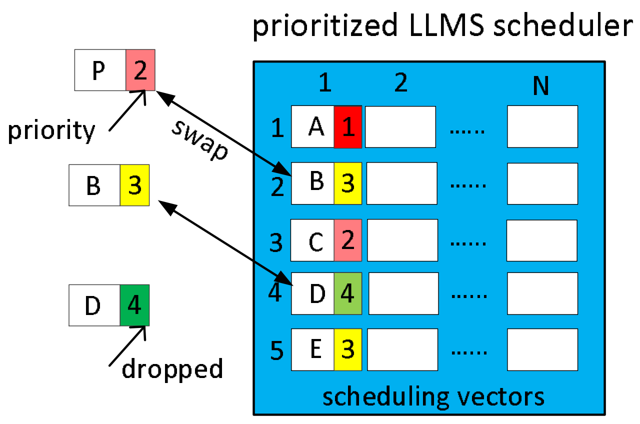


Figure 3.5: An example for the prioritized LLMS ($D = 5$).

QoS guarantee for premium members, while trying to serve free viewers in a best-effort manner. Therefore, it is desirable for interconnects to be able to provide differentiated QoS for flows of varied priorities. In this subsection, we propose a simple yet efficient modification to LLMS, denoted as prioritized LLMS, that is able to schedule each packet according to the priority of the corresponding flows.

Similar to the original LLMS, the prioritized LLMS also tries to find the earliest *eligible* entry among the scheduling vectors for output copies of each packet. Recall that an eligible entry has to be empty in the original LLMS, as denoted in the first of the three conditions. In the prioritized LLMS, we modify the condition to be that an entry is eligible for an arriving packet if it is empty *or* has been assigned to some packet with lower priority. Correspondingly, if the eligible entry found for an arriving packet is empty, then the packet is assigned in the same way as the original LLMS. Otherwise, if the eligible entry has been assigned to some packet with lower priority, the scheduler will *swap* the two packets, that is, assigns the entry to the packet with higher priority, then continues to schedule the packet with lower priority among the rest of the scheduling vectors. Low priority packets will be dropped if no eligible entry can be found.

Next, we use a simple example to illustrate the operation of the prioritized LLM-S in Fig. 3.5. As the prioritized LLMS operates similarly to the original LLMS, we only show their differences. We assume that each packet carries signaling bits representing the priority of the corresponding flow, with higher priority denoted by a smaller number, as shown in Fig. 3.5. At the beginning of a time slot, the prioritized LLMS scheduler tries to schedule an arriving packet P destined for output 1 with priority 2. The earliest eligible entry found for packet P has already been assigned to an earlier packet B with priority 3, thus the scheduler swaps packet P and B, then continues to schedule packet B among the rest of the scheduling vectors. Similarly, the scheduler swaps packet B with a packet with lower priority D, then tries to schedule D from the scheduler vectors left. As an eligible entry for packet D cannot be found, the packet is thus dropped.

It can be observed that packets from flows with high priority will generally be assigned to the front of scheduling vectors, whereas low priority packets occupy the back of the scheduling vectors. Correspondingly, the packets from high priority flow will be scheduled with lower latency and drop ratio. Therefore, with simple modification to the original LLMS algorithm, the prioritized LLMS can provide differentiated QoS according to the priorities of traffic flows efficiently.

3.4 Pipeline and Parallel Architecture

In this section, we first present a pipelining technique that distributes scheduling tasks to a sequence of sub-schedulers, each of which can finish its scheduling task in $O(N^2)$ time. Then, we show the procedures within each sub-scheduler can be further distributed to N processing modules that operate in parallel, which can be built using simple combination gates. We also give the combination circuit implementation for each processing module. We show that with such an implementation, each processing module in the proposed pipeline and parallel architecture achieves $O(1)$ time complexity.

The most time consuming part in LLMS involves a nested loop of three layers, when the scheduler tries to find the earliest eligible entries among D scheduling vectors for at most N arriving packets, each with a fanout of cardinality up to N. Also, it takes $O(ND)$ time to shift all scheduling vectors. To schedule packets among D scheduling vectors, we construct D sub-schedulers (SS), indexed by $1, 2, \dots, D$, and concatenate them to a directional cascaded ring, as shown in Fig. 3.6.

In each time slot, the arriving packets are processed through a sequence of sub-schedulers along the ring in a pipelined fashion. Each $SS_t, t \in [1, 2, \dots, D]$, has one built-in scheduling vector S_t and mask vector M_t, and takes as inputs the re-

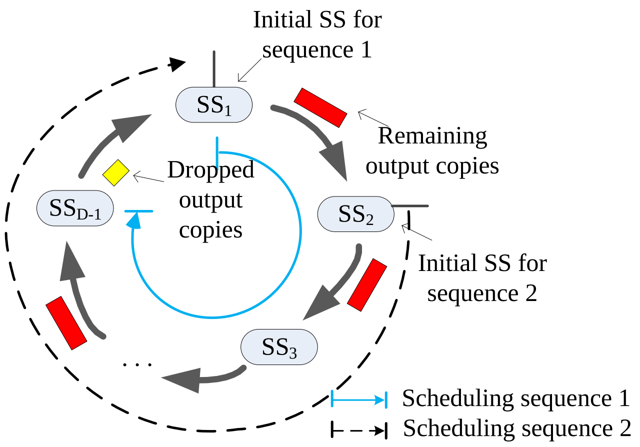


Figure 3.6: Ring of cascaded schedulers. The solid line and dashed line indicate the sequence of sub-schedulers packets go through in different time slots.

maining output copies of the processed packets that have not been scheduled. Each SS is responsible for the scheduling of the processed packets according to the built-in scheduling vector and mask vector, then passes the remaining copies as outputs to the next SS. The output copies that cannot be scheduled after going through all the D SS will be dropped. We denote the first sub-scheduler in the sequence as the *initial* SS. Starting from the initial SS, the t^{th} SS in the sequence is responsible for the scheduling for the time slot which is t time slots after the current time slot.

To relax the time constraints of LLMS, all sub-schedulers operate in a pipelined fashion. Fig. 3.7 illustrates the pipeline example for an optical multicast interconnect with $D = 6$ scheduling vectors, in which the time each SS takes to schedule the packet arrivals in one time slot is denoted as a *step*. The packet arrivals in the k^{th} time slot are denoted as P^k. Assume SS_1 is chosen as the initial sub-scheduler in the first time slot. SS_1 is responsible for the assignment of packet arrivals P^1 to its scheduling vector. When it finishes, the remaining output copies that cannot be assigned by SS_1 are passed to SS_2 as inputs. As shown in Fig. 3.7, the scheduling of P^1 is completed after all the sub-schedulers have been visited.

According to LLMS, the scheduling process for the current time slot cannot begin till all the packet arrivals from the previous time slot have been scheduled and the scheduling vectors have been shifted. As all the sub-schedulers must be visited during the scheduling process for one time slot, each step has to be completed within $1/D$ time slot. However, a key observation is that only the results of the

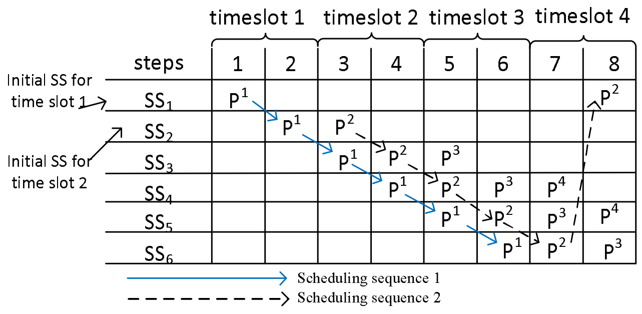


Figure 3.7: Pipeline operation for the first three time slots ($D = 6$).

first scheduling vector S_1 need to be ready for interconnect configuration in each time slot. After interconnect configuration, the information stored in S_1 will no longer be needed. This observation makes it possible to pipeline the scheduling for consecutive time slots to further reduce the time constraints of LLMS, as explained next.

In LLMS, all the scheduling vectors and mask vectors need to be shifted forward by one position at the beginning of each time slot, which involves massive data transfer among the vectors. To simplify the operation, we clear the initial SS then simply "shift" the scheduling sequence clockwise by one position at the beginning of every time slot, that is, choose the one next to the initial SS as the initial SS in the next time slot. For example, as depicted in Fig. 3.6, assume SS_1 is the initial SS for the scheduling sequence in the current time slot, and the arriving packets go through the sequence of SS's along the solid line. In the next time slot, SS_2 is chosen as the initial SS. In this way, SS_1 becomes the last SS in the sequence and all other SS's are one position closer to the initial SS, indicated by the dashed line in Fig. 3.6. The simple rotation avoids massive data transfer and only takes $O(1)$ time, while producing equivalent results to that of shifting all the vectors forward.

Moreover, due to the fact that only the result of the initial SS needs to be ready for transmission at the beginning of the next time slot, we do not have to wait till the scheduling for the previous time slot to complete before starting the scheduling for the next time slot. In other words, the scheduling in consecutive time slots can also be pipelined. At the beginning of the 2^{nd} time slot, the scheduler configures the interconnect according to the scheduling vector of SS_1, then clears the register storing the scheduling vector and mask vector in SS_1. Next, it shifts the initial SS of the processing sequence for the 2^{nd} time slot to SS_2. Instead of waiting

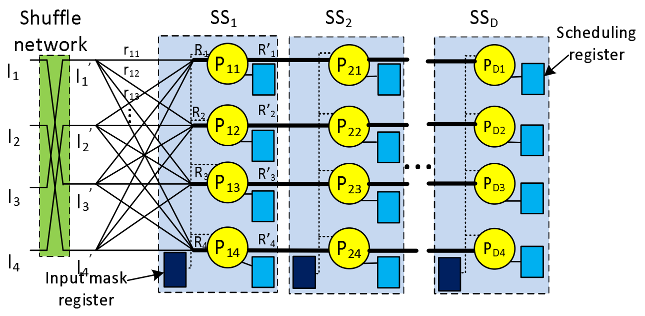


Figure 3.8: Pipeline and parallel architecture ($N = 4$).

for the completion of the scheduling for the 1^{st} time slot, the scheduling for the 2^{nd} time slot can start as soon as SS_2 is available, as shown by the dashed line in Fig. 3.7. Similarly, the scheduler configures the interconnect according to SS_2 and initiates the processing sequence from stage 3 at the beginning of the 3^{rd} time slot. To avoid conflict (i.e., the packet arrivals from two time slots enter the same SS), each SS needs to complete a step within a half time slot, which is a constant factor independent of the interconnect size N and the number of scheduling vectors D.

In each step, each SS has to schedule up to N arrival packets among the N entries in its scheduling vector, which takes $O(N^2)$ time. In high-speed switching, this requirement may be too stringent when N is large. Therefore, we also propose a processing architecture, which distributes the processing task of each SS to N processing modules operating in parallel. We show that combined with the pipelining technique and combination circuit implementation, the proposed processing architecture can reduce the time complexity of LLMS to $O(1)$.

The architecture for a 4×4 interconnect is shown in Fig. 3.8, which consists of D cascaded sub-schedulers, with N processing modules in each SS. Each processing module $P_{kn}, (n = 1, 2, \dots, N)$ in SS_k has a scheduling register V_{kn} storing the n^{th} entry of the corresponding scheduling vector, and all the processing modules in SS_k share one register recording the input mask vector.

At the beginning of each time slot, input port i sends a binary request $r_{ij}(j = 1, 2, \dots, N)$ to the j^{th} processing module in the initial SS according to the fanout information of the incoming packet, that is, $r_{ij} = 1$ if the incoming packet is destined for output j, otherwise $r_{ij} = 0$. The set of requests towards output j, i.e., $r_{ij}(i = 1, 2, \dots, N)$, are denoted as R_j. For processing module P_{kj}, a request r_{ij} is said to be *eligible* if scheduling register V_{kj} is empty and no previous packet from input i has been assigned to stage k. The basic operation for each processing module is to assign the eligible request with the *smallest* input index to its scheduling register.

In each time slot, processing module P_{kj} in the initial stage k takes request set R_j as the input, and outputs the set of remaining requests R'_j after processing, which is then fed to the next stage in the sequence as input. This process repeats until all the D stages have been visited. If there are still requests left unassigned, the corresponding output copies will be dropped. Since inputs with a smaller index are always scheduled at higher priority, we also adopt a simple shuffle network as in [91] to dynamically change the priority of each input port. In this way, all input ports have an equal chance to get its requests assigned first.

For each processing module, it takes $O(N)$ time to search among N requests, which is a non-trivial task. Next, we show this operation can be implemented by the following simple combination circuit with $O(1)$ time complexity. For processing module P_{kj}, we have following variables.

1. Request vector, denoted as $r_{ij}, i = 1, 2, \ldots, N$, is the input to the processing module; the remaining request vector after processing, denoted as r'_{ij}, is the output to the processing module;

2. $full$ bit of the entry in the scheduling register, denoted as F, where $F = 1$ if the entry of the scheduling vector is occupied, otherwise it is 0; updated $full$ bit after processing is denoted as F';

3. Input mask vector, denoted as M_i, where $M_i = 1$ if a packet from input i has been assigned to stage k previously, otherwise it is 0; updated input mask vector after processing is denoted as M'_i;

4. Scheduling results denoted as Q_i, $i = 1, 2, \ldots, N$. $Q_i = 1$ if request r_{ij} is assigned to the scheduling register in the processing module, otherwise it is 0;

For a processing module P_{kj}, Q_i is set to 1 only when all the following three conditions are met:

1. The scheduling register must be empty, i.e., $F = 0$;

2. None of the requests from inputs with an index smaller than i can be assigned to the scheduling register, that is, $r_{xj} \cap \overline{M_k(x)} = 0$ for $x \in \{1, 2, \cdots, i-1\}$;

3. The request from input i must be able to get assigned to the scheduling register, $r_{ij} \cap \overline{M_k(i)} = 1$.

Thus we can give the logic expression for Q_i as follows, from which r'_{ij}, F' and M'_i can be easily derived.

$$Q_i = \overline{F} \cap \overline{(r_{1j} \cap \overline{M_1})} \cap \cdots \overline{(r_{i-1,j} \cap \overline{M_{i-1}})} \cap (r_{ij} \cap \overline{M_i})$$

With pipeline and parallel processing, the time complexity of each processing module to implement LLMS becomes $O(1)$, and the hardware cost (i.e., the number of logic gates) of the proposed architecture is $O(N^2D)$. It is worth mentioning that the proposed architecture can implement the prioritized LLMS algorithm in Section 3.3.5 with a small modification: each processing module can use a comparator to compare the priority of incoming requests with that of currently assigned to its scheduling register, and simply swap the assigned one with the incoming request if the incoming one has higher priority.

Note that, the parallel pipelined architecture for output buffering scheme in [91] also achieves $O(1)$ time complexity. However, each packet has to go through multiple processing stages before it can be scheduled in the processing architecture, which introduces large pipelining overhead. In comparison, the scheduling results are immediately available at the initial stage of the processing sequence in the proposed scheduling architecture, therefore no scheduling overhead is introduced.

As mentioned before, the size of multicast capable optical switching fabrics is limited (no larger than 64×64 currently [87]). Therefore, one or a small number of high end FPGAs are sufficient to implement the proposed scheduler in practice. To see the hardware feasibility of the proposed architecture under high-speed switching, let us look at a 64×64 optical packet interconnect with 40 Gb/s port speed. Assume the packet length is 128 bytes and there are $D = 16$ scheduling vectors. Then the processing architecture requires approximately 65,000 logic gates and 80 MHZ clock frequency, which can be easily accommodated by the state-of-art FPGAs.

3.5 Performance Evaluations

We have conducted extensive simulations to evaluate the performance of LLMS, which consists of two parts. In the first part of the simulation, we evaluate the effect of the number of scheduling vectors D (i.e., the maximum delay) on the packet drop ratio, which is defined as the percentage of dropped output copies among the total output copies of all packets arrived during the simulation period, and the average delay, which is calculated by the average interval between the arrival and departure of all successfully transmitted output copies.

Due to the similarity between the adopted switching architecture and the input queued (IQ) electronic interconnect, we compare the performance of LLMS with several well-known multicast scheduling algorithms for IQ electronic interconnects, including FIFOMS [84], MCMS [89] and MLRRMS [90] algorithms. In our simulation, we also include a simple FIFO scheduling algorithm on the output queued interconnect (OQFIFO) as a performance benchmark. Scheduling algorithms for electronic interconnects usually assume infinite buffer size because of the available large size electronic buffer, which means that no packets will be dropped in the scheduling process, therefore only average delay performance will be evaluated when comparing LLMS with these algorithms. To evaluate the performance of LLMS in the terms of both average delay and packet drop ratio, we also compare LLMS with the output buffering scheme for optical packet interconnects [91]. These algorithms can be briefly described below.

FIFOMS is an iterative multicast scheduling algorithm. In an iteration, each unmatched input scheduler selects the HOL packet in each VOQ with the smallest timestamp and sends the requests to the corresponding outputs. The process continues till there is no possible match between inputs and outputs. FIFOMS was shown to be superior to many well-known scheduling algorithms in terms of packet latency.

MCMS considers the scheduling of the HOL packet in each input queue for multiple time slots (the number of time slots considered is set to 64 in our simulation). It was demonstrated that the latency performance of MCMS outperforms most of previous scheduling algorithms such as WSPLIT, revision scheme and windows-based algorithms.

MLRRMS schedules the HOL packets in each input queue first. If there are any output and input left idle, the scheduler then tries to send the packets behind the HOL packet in each idle input to idle outputs. This process continues until either the maximum look-ahead depth d is reached or all packets in the queues are examined, or no idle outputs are found in the schedule. MLRRMS can effectively alleviate HOL blocking, thus achieves high switch throughput. However, one major constraint of MLRRMS is that the internal memory access rate of input buffers must run d times faster than switch port bit rate. Such a constraint makes MLRRMS impractical to implement in high speed switches. Nevertheless, we include its simulation results to evaluate LLMS. The maximum search depth d is set to 1 in the simulation as in [90], which means that the scheduler examines the first two packets in each queue.

The output queued interconnect is known to have optimal delay performance when no packets are dropped, but requires N times faster switching ability and memory speed. Despite its much stronger hardware requirement, in our simulation, we include a simple FIFO scheduling algorithm on the output queued interconnect (**OQFIFO**) as a performance benchmark to demonstrate the good performance of LLMS.

The output buffering (**OUTBUF**) scheme places an output buffer consisting of an $N \times D$ high speed optical interconnect and D FDLs with incremental delay in front of each output port. When there are multiple packets destined for the same output, the scheduler determines the proper FDL segment that each packet should go to, such that they would exit the FDL buffer without contention. When operating in time-slotted manner, OUTBUF can emulate the OQFIFO scheduling with the constraints that each output queue is of size D, thus can achieve optimal performance in both average delay and packet drop ratio. However, as shown earlier, OUTBUF requires a prohibitive amount of FDL segments. We use it as a performance benchmark to test how close our algorithm can be to the optimal results in the terms of packet drop ratio and average delay.

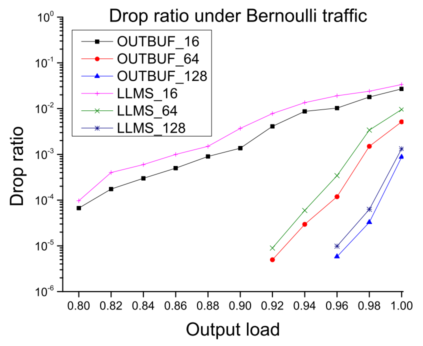
In the second part of the simulation, we study the performance of the prioritized LLMS algorithm. Similar to [88], we consider two priority classes in the traffic, with the arrival ratio of high priority traffic to low priority traffic $\rho_1 : \rho_2$ set to $1 : 3$, i.e., 25% of packets are high priority packets and the remaining are low priority packets. The effectiveness of prioritized LLMS is demonstrated by comparing the packet drop ratio and average delay of traffic belonging to different priority classes.

In each simulation run, there is a sufficient warmup period (typically one fourth of the total simulation time) to obtain stable statistics. The simulation runs for a fixed amount of simulation time (10^6) unless the scheduling algorithms become unstable (i.e., the interconnect reaches a stage where it cannot sustain the offered load). For cases in which the packet drop ratio is in the order of 10^{-4}, we extend the simulation period to 10^7 time slots for more reliable results. In order to compare the performance of the algorithms in various networking environments, simulation are conducted for different interconnect sizes (8×8, 16×16 and 32×32) under several different types of traffics, including Bernoulli traffic, gathered traffic, unicast traffic, and real Internet traffic. For statistical traffic patterns (i.e., Bernoulli traffic, gathered traffic and unicast traffic) used, all inputs are assumed to have identical packet arrival process, and for Internet traffic simulation, a different trace file is fed into each input to simulate the packet arrival process. Since we observe similar results for all interconnect sizes, only the results for the 16×16 interconnect are shown.

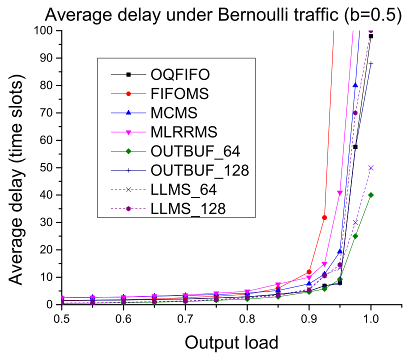
3.5.1 Performance under Bernoulli Traffic

Bernoulli traffic is one of the most commonly used traffic models in the simulation of scheduling algorithms. Bernoulli traffic can be described using two parameters, λ and b. λ is the probability that an input port is busy in a time slot, i.e., the rate packets arrive at some input port at the beginning of a time slot. Given an arriving multicast packet, determining whether the packet is destined to a specific output can be considered as a *Bernoulli trial*. The trial is called a "success" if the output is a destination of the packet. As there are N outputs, finding the set of destination outputs of a packet requires N independent and identical Bernoulli trials, with b being the success probability in each trial. The number of destinations outputs of the packet follows a binomial distribution with expected value bN. b is set to 0.5 in the simulation. For an $N \times N$ interconnect, the average fanout of a multicast packet is bN and the output load μ is λbN.

We first compare on the packet drop ratio of both LLMS and OUTBUF under Bernoulli traffic in Fig. 3.9(a) for different values of D. For LLMS, D means the number of scheduling vectors, whereas D is the longest delay of each output buffer in OUTBUF. As can be seen from the figure, packet drop only occurs at very high traffic loads (over 0.8) under Bernoulli traffic, which can be explained as follows.

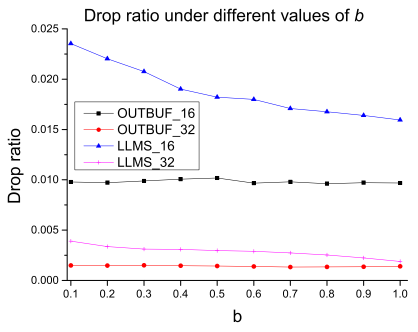


(a)

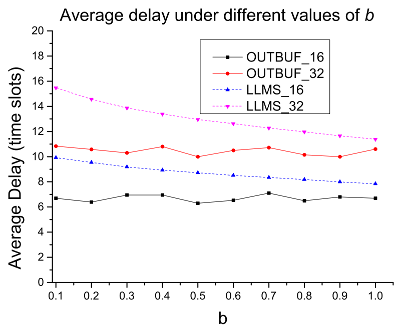


(b)

Figure 3.9: Performance for a 16×16 interconnect under Bernoulli traffic with $b = 0.5$. (1) packet drop ratio; (b) average delay.



(a)

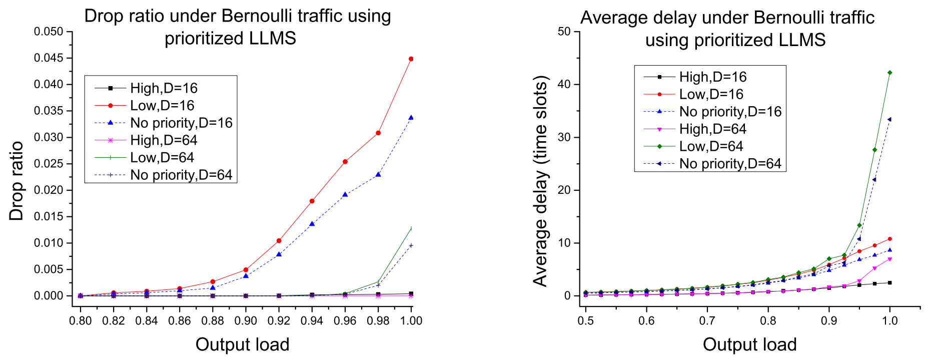


(b)

Figure 3.10: Performance for a 16×16 interconnect under the same output load (0.95) with different values of b. (1) packet drop ratio; (b) average delay.

When the traffic load is light, the scheduling vectors are relatively empty, and the expected number of arriving packets competing for the scheduling vectors in each time slot is small. Therefore, most output copies can be assigned in the vectors and few packets will be dropped. On the other hand, when the traffic load is heavy, the scheduling vectors are mostly occupied, and the expected number of arriving packets in each time slot is larger, leading to increased packet drop ratio.

The drop ratio reduces drastically when we increase D from 16 to 64, but only a slight improvement is observed when we further increase D to 128, indicating that most packets can be scheduled for transmission within 64 time slots. LLMS closely matches OUTBUF in the terms of packet drop ratio when D is sufficiently large (e.g., $D = 64$). When D is small (e.g., $D = 16$), the packet drop ratio of LLMS



(a) (b)

Figure 3.11: Performance comparison of traffic of different priorities for a 16×16 interconnect under Bernoulli traffic using prioritized LLMS. (1) packet drop ratio; (b) average delay.

is slightly higher than that of OUTBUF, which indicates that LLMS can achieve near-optimal packet drop ratio under Bernoulli traffic. The reason why LLMS has higher packet drop ratio than OUTBUF is that a packet cannot be assigned to an entry in a scheduling vector even when it is empty, if a previous packet from the same input is scheduled to be transmitted in the same vector, which leads to more dropped packets under LLMS compared with OUTBUF.

Fig. 3.9(b) compares the average packet latency of LLMS under Bernoulli traffic with other algorithms. It can be seen that, as the traffic load increases, MLRRM-S, MCMS and FIFOMS become saturated due to HOL blocking, which coincides with the theory that the IQ interconnect cannot maintain sustainability under al-l admissible multicast traffic conditions [86]. At the same time, we can see that the proposed LLMS algorithm closely matches the performance of OUTBUF, and significantly outperforms all other algorithms when the traffic load is heavy.

When the traffic load approximates 1, LLMS achieves better delay performance than OQFIFO. The reasons are two folds: first, LLMS can detect and promptly drop output copies with overlong latency instead of keeping them in the buffer, thus significantly reduces the average packet latency; second, the outputs not used by buffer transmission are open for cut-through scheduling with zero latency, further reducing the average packet latency.

We also observe that LLMS_64 performs better than LLMS_128 in terms of average packet latency. The reason is that with more scheduling vectors, the scheduler is less prone to drop packets and more tolerant to packet latency. For example, an output copy of some packet that expects to have a delay of 70 time slots would be dropped by LLMS_64, while it would be scheduled for transmission by LLMS_128. Such trade-off between the packet drop ratio and the average packet latency can be easily adjusted by changing the number of scheduling vectors, making LLMS highly adaptive to various transmission requirements.

Given the same output load (0.95), Fig. 3.10 compares the performance of LLMS and OUTBUF under different values of b. We observe that b has negligible impact on the performance of OUTBUF, which is expected because the output load stays the same with different b. Meanwhile, packet drop ratio and average delay under LLMS slightly increase when b is smaller, which can be explained as follows. Given that at most one packet can exit an M-FDLs buffer in each time slot, a scheduled packet would block other packets arriving at the same input from using the scheduling vectors it occupies, which we refer to as input blocking. Under the same output load, the input blocking level increases when b is smaller, because of the larger packet arrival rate at each input, which causes increased packet drop ratio and average delay.

Fig. 3.11 compares the performance of prioritized LLMS algorithm for traffic belonging to different priority classes. The performance of non-prioritized LLMS is also presented for comparison purpose (dash lines). We consider prioritized LLMS with the number of scheduling vectors D set to 16 and 64, respectively. We can see from Fig. 3.11(b) that all high priority packets are delivered with negligible drop ratio under all tested output loads. The difference in average delay performance between high priority traffic and low priority traffic is also very significant, as shown in Fig. 3.11(b). Meanwhile, low priority traffic only suffers a marginal increase of packet drop ratio and average delay, compared to the non-prioritized LLMS. These results confirm that prioritized LLMS is very effective in providing service differentiation to traffic of different priorities.

3.5.2 Performance under Gathered Traffic

As multicast applications, such as IPTV and Video on Demand (VoD), often generate sustained and long-lasting flows that may gather among fewer active input ports and engage more output ports at a given switch, we also examine the performance of the LLMS scheduler under gathered traffic. The gathered traffic scenario is known to be difficult to schedule for input-queued switches, and has been widely adopted in the simulation studies on multicast scheduling [96]. We adopt a similar setting as [96] in our simulation, where the number of active input ports M is set to 5 and the number of active output ports N is fixed at 16. We assume the fanout of a multicast packet is chosen uniformly at random among all possible fanout sets with the average fanout $h_m = 8$. Let λ be the rate of packet arrival at each input, then output load $\mu = \lambda M h_m / N$. Next, we explore the sensibility of the scheduling performance to output load μ.

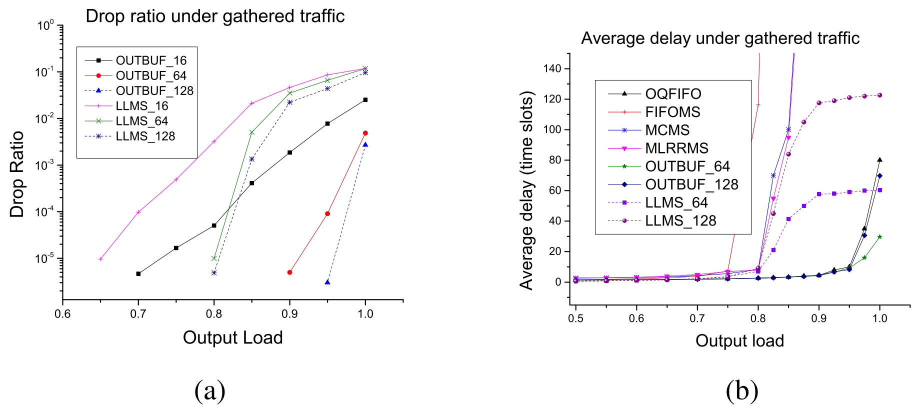


Figure 3.12: Performance for a 16×16 interconnect under gathered traffic with probability destined for each output $b = 0.5$. (1) packet drop ratio; (b) average delay.

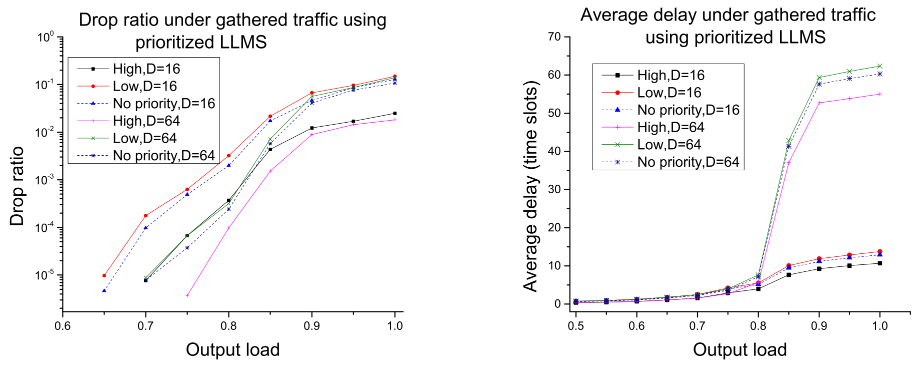
Fig. 3.12 shows the performance for a 16×16 interconnect under gathered traffic with probability destined for each output $b = 0.5$. From the figure, we can see that both output-queued scheduling algorithms (OQFIFO and OUTBUF) have comparable performance under the gathered traffic to the Bernoulli traffic scenario in terms of packet drop ratio and average delay, while other algorithms relying on input queue perform notably worse. The underlying reason is that input-queued scheduling algorithms, like FIFOMS and MCMS, can only send one packet for transmission from each input port within a time slot. Since traffic is concentrated among a few active input ports in gathered traffic, it is likely that the output copy of a buffered packet cannot be transmitted even though the corresponding input port and output port is idle, because it is blocked by the HOL packet in the same queue that fails to be scheduled for transmission. Among all the input-queued scheduling algorithms, MLRRMS has the best delay performance under gathered traffic, which is because it will try to send packets behind the HOL packet of each queue to fill as many idle output as possible, which reduces the time each packet has to stay in buffer.

Meanwhile, we observe that LLMS is able to achieve significantly better performance than other input-queued algorithms in terms of average delay, and achieve comparable performance to OUTBUF when output load is below 0.9. Due to the limitation imposed by the input-queued structure, MLRRMS, FIFOMS and MCMS saturate before the output load reaches 0.85 under gathered traffic, which means that the number of queued packets will keep increasing until buffer overflow. In such cases, it could take quite a long time for the upper layer protocol to detect congestion at the switch and drop packets if the buffer size is large. In contrast, the proposed $LLMS$ enables swift congestion detection and promptly drops packets with overlong delay shortly after arrival, which allows the upper layer protocols to adjust to network condition in time.

Fig. 3.13 shows that prioritized LLMS is quite effective at performance differentiation in gathered traffic too: packet drop in high priority traffic occurs only under high output load ($\mu = 0.85$), and is $1/10$ that of low priority traffic when output load $\mu = 1$. The average delay of high priority traffic is also much lower than that of low priority traffic. On the other hand, we also observe that there are about 2% high priority packets being dropped under gathered traffic when output load is 1, which is higher than Bernoulli traffic scenario, where there is nearly no packet drop in high priority traffic. Such difference is also caused by the fact that arriving packets are concentrated in a few input ports, which could more easily overwhelm the capacity of each M-FDLs buffer, leading to higher packet drop ratio and longer average delay.

3.5.3 Performance under Unicast Traffic

Since a substantial portion of traffic in the Internet is unicast, where each packet is forwarded to a single destination, in this subsection, we show that LLMS is also capable of dealing with unicast traffic efficiently. For a unicast packet, it has an equal probability ($1/N$) being destined for each output port. It is easy to see that when the traffic only consists of unicast packets, the output load μ is equal to arrival



(a) (b)

Figure 3.13: Performance comparison of traffic of different priorities for a 16×16 interconnect under gathered traffic using prioritized LLMS. (1) packet drop ratio; (b) average delay.

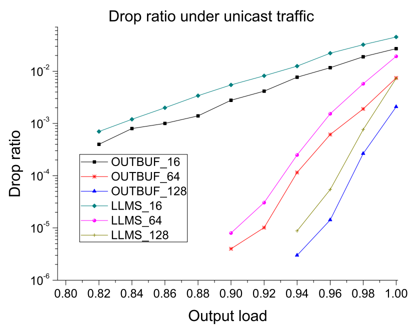
rate λ.

The effect of D on the packet drop ratio and average delay of both LLMS and OUTBUF under unicast traffic for a 16×16 interconnect is illustrated in Fig. 3.14. We can see that, with $D = 64$, packet drop only occurs at very high traffic load (over 0.95) and can be kept at very low level. We also show that the average packet latency of LLMS under unicast traffic in Fig. 3.14(b). We can observe that MLRRMS, MCMS and FIFOMS become saturated before the traffic load reaches 1. In comparison, LLMS consistently achieves low latency under all traffic loads, and closely matches OUTBUF in terms of both average packet delay and packet drop ratio, indicating that it can also achieve near-optimal performance under unicast traffic. Fig. 3.15 demonstrates that the prioritized LLMS achieves similar effectiveness to Bernoulli traffic in providing QoS differentiation under unicast traffic.

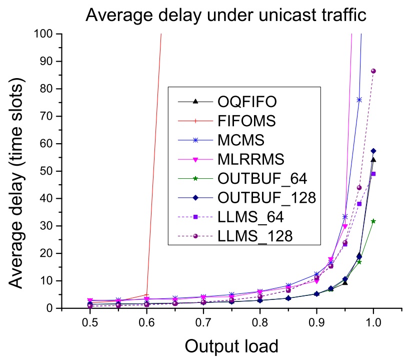
3.5.4 Performance under Internet Traffic

Due to the complexity of Internet traffic, it is very difficult, if not impossible, to completely capture its characteristics using statistical traffic models. For this reason, we have also tested the proposed LLMS algorithm under real Internet traffic traces obtained from the backbone network link monitors.

The anonymized traffic traces used here were obtained from the CAIDA's passive $OC192$ network link monitors [97]. All trace files consist of one line per IP

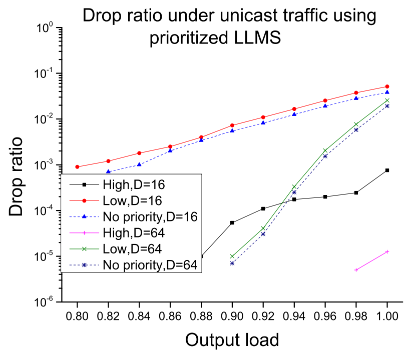


(a)

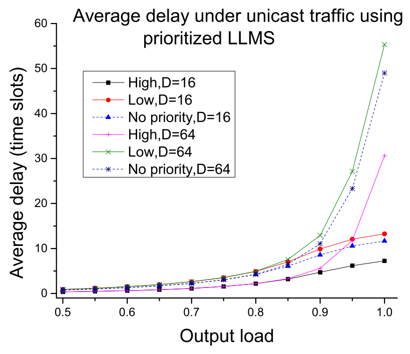


(b)

Figure 3.14: Performance for a 16×16 interconnect under unicast traffic. (1) packet drop ratio; (b) average delay.



(a)



(b)

Figure 3.15: Performance comparison of traffic of different priorities for a 16×16 interconnect under unicast traffic using prioritized LLMS. (1) packet drop ratio; (b) average delay.

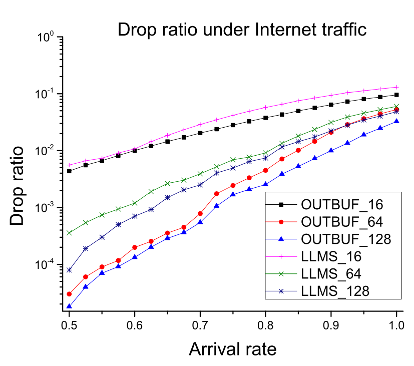
packet arrival in the form of <*packet_ index, time_stamp, protocol, source_IP_address, destination_IP_address*>, where *time_stamp* of each packet records the exact time when the packet is intercepted by the link monitor.

In the simulation, we feed each input with a separate trace file, in which all packets are assumed to have a fixed size of 64 bytes and fit into one time slot. Note that due to lack of regulation in real Internet traffic, it is likely that certain outputs are heavily oversubscribed, that is, they receive more packets than that can be transmitted over the simulation period. Also, it is impossible to regulate the traffic load in each output. Therefore, different from the simulation under statistical traffic models, where the traffic is admissible (no oversubscription at outputs) and the scheduling algorithms are evaluated against the output traffic load, we test the algorithms against the packet arrival rate at the inputs in real Internet traffic as in [50].

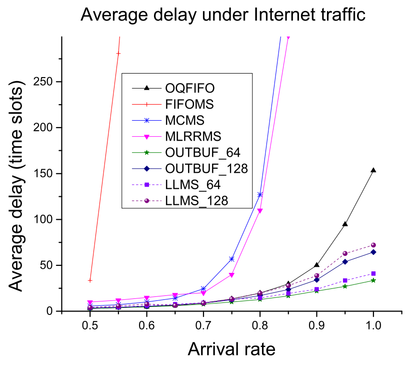
To adjust the arrival rate, we process each trace file using a similar method as in [50], elaborated as follows. First, we calculate the average throughput of a trace by dividing the total traffic volume by the time span of the trace. Then, we set the length of each time slot according to the throughput of the trace to achieve a desired arrival rate. For example, if the throughput of a trace is 500 Mb/s and we want to achieve the arrival rate of 0.8, then the time slot length is set as $((64 \times 8)/500M) \times 0.8 = 0.8192\mu$s. Finally, we find a mapping from packets to different time slots according to the time stamp of packets. If the time stamp of multiple packets fall in the same time slot, they are placed in consecutive time slots. After the above procedures, each processed traffic trace is used to simulate the packet arrival process in an input port.

With the absence of the forwarding table, determining how to map the destination IP address of packets to output ports of the interconnect is not a trivial task. Since forwarding tables in the routers are updated relatively infrequently, we can assume that it stays invariable during the simulation period. We use the destination IP address of a packet to determine whether it is a unicast packet or a multicast packet. If the destination IP of a packet is a class D address, then it is a multicast packet. Otherwise, it is a unicast packet. For unicast packets, we use a simple hash function to determine the output port for each packet, which returns the modulus of summation of the four IP address fields in each destination IP to the interconnect size N. For example, a packet with destination IP address $243.124.121.4$ will be sent to port $(243 + 124 + 121 + 4) \mod N + 1$. For a multicast packet, we assume its fanout f (i.e., the number of its destination outputs) is uniformly distributed between 1 and N, then we randomly choose f output ports as its destination outputs.

From Fig. 3.16(a), we can see that there is a noticeable increase in the packet drop ratio compared to previous traffic patterns. The reason is three-fold. First, we use the packet arrival rate at the input instead of the output load in this simulation. As real Internet traffic consists of both unicast flows and multicast flows, the output load is considerably higher than the arrival rate. Second, the real Internet traffic consists of many flows, in which packets arrive in consecutive time slots

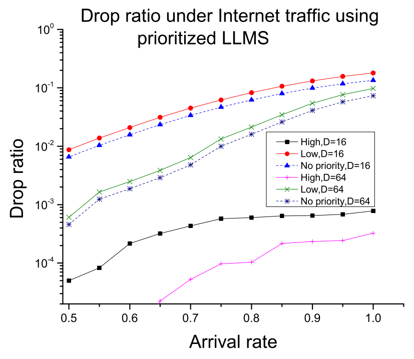


(a)

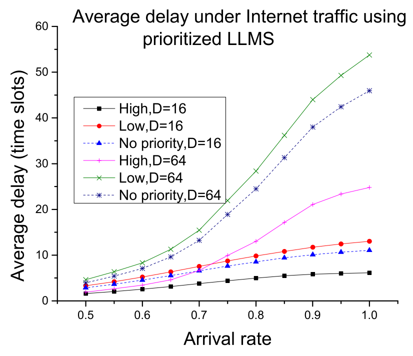


(b)

Figure 3.16: Performance for a 16×16 interconnect under Internet traffic. (a) packet drop ratio; (b) average delay.



(a)



(b)

Figure 3.17: Performance comparison of traffic of different priorities for a 16×16 interconnect under Internet traffic using prioritized LLMS. (1) packet drop ratio; (b) average delay.

and share the same output ports. Such traffic bursts are more likely to cause packet drop. Finally, real Internet traffic could be inadmissible during the simulation period, as some output ports could be oversubscribed. Nevertheless, for a switch with $N = 16$, which is typical size for optical interconnects [79], a M-FDLs buffer with approximately $D = 128$ FDL segments for each input port is sufficient to keep the packet drop ratio at a reasonably low level under the most congested Internet traffic condition.

As for average packet latency shown in Fig. 3.16(b), FIFOMS, MCMS and MLRRMS, perform considerably worse under real Internet traffic than under other traffic models. FIFOMS saturates before the packet arrival rate reaches 0.55, while both MCMS and MLRRMS saturate before the arrival rate reaches 0.9. Meanwhile, we can see that LLMS manages to achieve low average packet latency for the same reason as that under statistical traffic models. Again, LLMS closely matches OUTBUF in both packet drop ratio and average delay, indicating that it is able to achieve near optimal performance under Internet traffic.

From Fig. 3.17, we can see that prioritized LLMS can deliver high priority traffic with negligible drop ratio and much lower delay under Internet traffic, which has practical implications. For example, suppose a typical network congested with a large volume of IPTV or VoD traffic, prioritized LLMS is able to deliver high priority packets of applications sensitive to latency and packet drop, e.g., an on-line conference, across the network with little interference, which is highly desirable.

3.6 Conclusions

In this chapter, we have studied multicast scheduling problem for input-buffered optical multicast interconnects. We first proposed an efficient optical buffer called multicast-enabled FDLs (M-FDLs), which can provide flexible delay for output copies of each multicast packet while requiring only a small number of FDL segments. We also designed a Low Latency Multicast Scheduling (LLMS) Algorithm, the main features of which can be summarized as follows: (1) guarantee a fixed delay upper bound for all transmitted packets; (2) achieve close to optimal average delay and packet drop ratio even under the most congested traffic conditions; (3) enable fast congestion detection and promptly drop output copies with long delay; (4) require much shorter FDL length compared to existing multicast scheduling schemes for all-optical packet interconnects; (5) easily extendable to provide QoS differentiation. We also presented a pipeline and parallel processing architecture and the combination circuit design for LLMS, where it takes each processing module $O(1)$ complexity to finish scheduling in each time slot. The proposed processing architecture does not incur any latency overhead and enables packet scheduling at the line rate, which is essential to optical packet interconnects with ever-increasing port speed. Finally, we evaluated the performance of LLMS through extensive simulation using both statistical traffic models and real Internet traffic traces.

Chapter 4

Exploring Server Redundancy in Constructing Cost-Effective Nonblocking Multicast Fat-tree Data Center Networks

In this chapter, we investigate how to deploy nonblocking multicast cost-effectively in fat-tree DCNs by exploring server redundancy in data centers. First, we present a multirate network model that accurately describes the communication environment of the fat-tree DCNs. We then show that the sufficient condition on the number of core switches required for nonblocking multicast communication under the multirate model can be significantly reduced when the fat-tree DCNs are 2-redundant, i.e., each server in the data center has exactly one redundant backup. We also study the general redundant fat-tree DCNs where servers may have different numbers of redundant backups depending on the availability requirements of services they provide, and show that a higher redundancy level further reduces the cost of nonblocking multicast fat-tree DCNs. Finally, to complete our analysis, we consider a practical faulty data center, where one or more active servers may fail at any time. We give a strategy to re-balance the active servers among edge switches after server failures so that the same nonblocking condition still holds.

The rest of the chapter is organized as follows. Section 4.1 gives the introduction and background. The related work are discussed in Section 4.2. Section 4.3

briefly introduces the background of fat-tree DCNs and the multirate network model adopted. Section 4.4 presents the sufficient multicast nonblocking condition on the number of core switches in redundant fat-tree DCNs under the multirate model. Section 4.5 generalizes the results to faulty data centers. Section 4.6 compares the cost of the nonblocking condition for fat-tree DCNs with different sizes. Finally, Section 4.7 concludes the chapter.

4.1 Introduction

Reducing the cost of DCNs is a first order design concern for maximizing data center profits. Much research effort has been devoted to the search for cost-efficient and scalable data center network fabrics in recent years [25, 98–100], among which fat-tree, a special instance of Clos networks, is widely adopted as the topology for DCNs [18, 23], for the reason that it enables to build large-scale communication networks from many small commodity switches rather than fewer larger and more expensive switches. Due to the huge price difference between commodity and non-commodity switches, such property of fat-trees provides a strong economical incentive.

One of the primary concerns in a fat-tree DCN is that whether it can provide *full bisection bandwidth*, i.e., the capability to connect an arbitrary server with any other server(s) in the network at the full bandwidth of their local network interface. To achieve such congestion-free communication, a fat-tree DCN needs to be nonblocking, that is, it must be able to establish an arbitrary connection without causing any contention in the network at any time. Extensive research [27–30] has been conducted on finding the most cost-effective solution to achieve nonblocking communications in such Clos-type networks. While these studies indicate that nonblocking unicast communication can be achieved with a reasonably small number of core switches in a fat-tree network, nonblocking multicast fat-tree still has higher cost than its unicast counterpart, mainly due to the non-uniformity of multicast traffic pattern that requires a large number of core switches. Since multicast is an essential communication pattern in many data center services, such as redirecting search queries to a set of index servers or replicating file chunks in distributed systems, and nonblocking multicast communication can provide guaranteed high performance of such services, it is critical to reduce the cost of nonblocking multicast fat-tree DCNs.

On the other hand, some types of data centers, referred to as *high availability (HA) data centers*, have extremely stringent availability requirements, which are commonly used for key data storage, financial data distribution and banking systems, etc. [101, 102]. The consequence of downtime for these services is catastrophic, thus 6-nines (99.9999%) or almost error free is a common availability target for HA data centers [3]. To achieve near-perfect service availability, HA data center implementations build server redundancy into the data center to eliminate single points of failure [101–103]. Active servers (i.e., servers that are currently running applications) in such data centers have fully redundant instances provided as standby backups, which can be brought online immediately when their associated active servers fail. For example, when an active server with a particular application loses network access or crashes, the application will be unavailable until the problem is fixed. Data centers with server redundancy implemented can remedy this situation by immediately restarting the application on one of its backup servers without requiring administrative intervention, thus shielding the external observation of the failure. Server redundancy, though it incurs extra hardware cost, is necessary in guaranteeing overall system availability in data centers.

Given the existence of server redundancy in HA data centers, in this dissertation we consider how to utilize such existing redundancy to reduce the cost of nonblocking multicast fat-tree DCNs. To achieve fully nonblocking multicast communication, a fat-tree DCN needs to be sufficiently provisioned to handle all types of multicast traffic. As all the simultaneous connections from a single edge switch in a fat-tree DCN will compete for the same group of core switches when establishing connection paths, the demand for core switches is the most stringent when some edge switch is *congested*, i.e., all the servers associated with the edge switch are active simultaneously and requesting broadcast connections at the full bandwidth capacity of their local network interfaces. Such congestion at edge switches and the non-uniformity of multicast traffic pattern account for the high cost of nonblocking multicast fat-tree DCNs. We will show that data centers with server redundancy can effectively mitigate such congestion by evenly distributing active servers and standby backup servers among edge switches, thus limiting the maximum possible number of simultaneous connections from any edge switch.

First, we will present a multirate network model that accurately describes the communication environment in fat-tree DCNs. Initially, we assume that there is no server failure in data centers to make the problem tractable. Our analysis shows that in this case the sufficient condition on the number of core switches required for nonblocking multicast communication can be significantly reduced when the fat-tree DCN is 2-redundant, i.e., each active server has one redundant backup server. We then generalize the result to practical fat-tree DCNs where servers may have different numbers of redundant backups depending on the availability requirements of the services they provide, and show that a higher redundancy level further reduces the cost of nonblocking multicast fat-tree DCNs.

Next, we will consider data centers where one or more active servers may fail at any time. It can be observed that server failure causes uneven congestion in edge switches, which may make a redundant data center that is nonblocking during normal operation blocking in case of failure. We show that such uneven congestion can be re-balanced by reassigning servers among edge switches. To optimally solve the server reassignment problem, we propose a graph modeling method and show that the problem can be transformed to finding the minimum cost network flow in the graph. We also prove that the reduced nonblocking condition still holds in case of server failure with server reassignment. Finally, we compare the sufficient multicast nonblocking condition on the number of core switches in fat-tree DCNs with different sizes and redundancy levels, and show that substantial cost saving can be achieved through exploring server redundancy.

4.2 Related Work

Multicast capability of three-stage Clos networks has been investigated in the literature. The most significant results on multicast nonblocking conditions of C-los networks are summarized as follows. Masson [104] and Hwang [105] gave the sufficient conditions on strictly nonblocking and rearrangeable nonblocking multicast Clos networks. Yang and Masson [29] gave a sufficient condition for wide-sense nonblocking multicast Clos networks denoted as $v(m, n, r)$, with $m > 3(n-1)\frac{\log r}{\log \log r}$, which yields the currently best available design for this type of multicast network. In [30], this condition was also shown to be necessary under several commonly used routing strategies.

The analysis of Clos networks for circuit switching telephone traffic adopts a

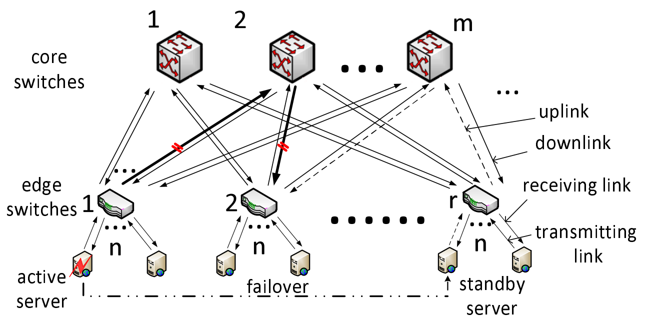


Figure 4.1: A fat-tree data center network. When an active server crashes, its applications (whose connections are represented in bold lines) fail over to its backup server (dashed lines) immediately.

single-rate network model, in which each link is dedicated to a single connection at one time. However, most modern networks operate in a packet switching manner, where packets from different connections are allowed to share a link through multiplexing techniques. To accurately describe such networks, Melen and Turner [28] presented a multirate network model, which can support connections with different data rates and a connection can consume an fraction of the bandwidth of the link carrying it, instead of exclusively occupy the link. They also gave the strictly nonblocking conditions for multirate unicast Clos networks. Liew and Chan [106] improved the nonblocking condition for unicast Clos networks under both the discrete bandwidth model and the continuous bandwidth model. Later, Yang [107] generalized the results on single-rate multicast networks to the multirate model, and gave the wide-sense nonblocking condition for multirate Clos networks under multicast traffic. For a comprehensive summary on nonblocking Clos networks, readers may refer to [27].

4.3 Preliminaries

In this section, we first introduce the background of fat-tree DCNs and server redundancy in data centers. Then, we present the multirate multicast network model. We also give some definitions, notations and observations that will be useful in our analysis of the nonblocking condition for multirate multicast fat-tree DCNs.

4.3.1 Fat-tree DCNs

The fat-tree network is widely adopted as a topology for data center interconnection. Fig. 4.1 illustrates the topology of a level-2 fat-tree. Each edge in the fat-tree network consists of two directed links. The lower level consists of $r = (n + m) \times (n + m)$ edge switches and the upper level consists of $m = r \times r$ core switches, where r is the number of edge switches, each connecting n servers, and m is the number of core switches. We will use the notion $ftree(m, n, r)$ to denote such a fat-tree. As can be seen in Fig. 4.1, given the number of edge switches r and the number of servers associated with each edge switch n, the connection capacity of a fat-tree network $ftree(m, n, r)$ depends on the number of core switches m. Deploying more core switches will certainly improve the connection capacity of the fat-tree network, but will also increase the cost.

As mentioned in [18], limited by the switch size, a level-2 fat-tree can support 5K to 8K hosts. To connect more hosts, we can replace a core switch with a level-2 fat-tree network of smaller switches, expanding the network to a level-3 fat-tree. Since a high-level fat-tree is recursively built from the basic level-2 fat-tree blocks, and any property of a level-2 fat-free still pertains to a high-level fat-tree, we focus on level-2 fat-tree DCNs. In a level-2 fat-tree DCN $ftree(m, n, r)$, an edge switch connects to each core switch through two directed links. We denote the links from edge switches to core switches as *uplinks* and the links from core switches to edge switches as *downlinks*. Such a fat-tree network can interconnect $N = nr$ server-s, with each server connected to an edge switch also through two directed links, denoted as *transmitting link* and *receiving link*.

A fat-tree DCN structure based on a special instance of fat-tree called level-3, k-ary tree was presented in [18], which can be built using only commodity switches with k GigE ports. In such a fat-tree DCN, servers are placed in different *pods*, and all the severs inside a pod are connected by a level-2 $ftree(k/2, k/2, k/2)$ network. As inter-pod bandwidth is likely to be the bottleneck of a fat-tree DCN, a common approach is to adopt inter-switch links with higher bandwidth than that of server-to-switch links to reduce congestion. For example, the fat-tree DCN in [23] uses 1G server-to-switch links and 10G switch-to-switch links in order to provide large connection capacity between edge switches.

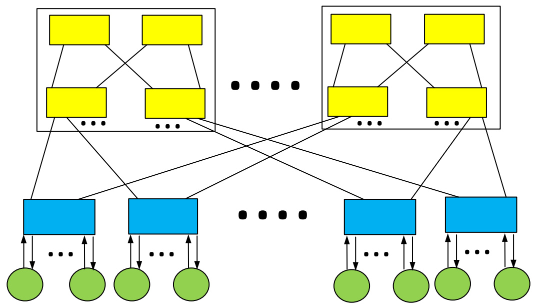


Figure 4.2: A level-3 fat-tree can be built by replacing each core switch with a nonblocking level-2 fat-tree network.

4.3.2 Server Redundancy in HA Data Centers

To provide high service availability, HA data centers build redundancy into every component of the data centers, including server, network, power and cooling, etc. In this subsection, we briefly introduce server redundancy in HA data centers.

Practical HA data centers usually adopt an active/standby server redundancy model [101, 102]. In such a model, each *active server*, i.e., the server that is currently running applications, has one or more idle *standby servers* provided in case of failure. When an active server fails, one of its associated standby servers is brought online, and takes over all the applications from the active server, such a process is called *failover*. Fig. 4.1 also shows an example of failover.

A variety of maintenance tasks are required to ensure successful and timely failover in case of server failure, among which two most important ones are server monitoring and data synchronization. Server monitoring is implemented by the "heartbeat" mechanism, in which each server sends a message in a given interval confirming that it is alive. Data synchronization between each active server and its standby servers is achieved through data disk sharing via a storage area network (SAN), or data replication techniques such as EMC Symmetrix Remote Data Facility [101]. Note that, traffic from both heartbeat and data replication is transmitted through a dedicated private network that is separated from the DCN [101, 102].

Each active server and its corresponding standby backups are *identical* instances of each other, meaning that, they share homogeneous software/hardware configurations and data storage to ensure successful failover. In addition, they are required to be *independent* to avoid being taken off-line simultaneously, i.e., they should not share the same edge switch, power supply, etc. In our analysis, we say a data center network is *2-redundant*, if every server has another redundant instance provided in the data center network, under the constraints that they must be associated with different edge switches. By the same token, we say a data center network k*-redundant*, if every server has another $k - 1$ independent redundant instances. In many practical data centers, servers have different numbers of redundant instances depending on the availability requirement of services provided [108]. We say such data center networks *general redundant*, or simply *redundant* for brevity.

4.3.3 Multirate Network Model

In this subsection, we introduce the multirate multicast network model.

First, it is reasonable to assume all switches in a fat-tree DCN have multicast capability, that is, the ability to establish a connection from an arbitrary input port to a group of output ports simultaneously. As a fat-tree network can be recursively constructed, each switch requiring multicast capability can be replaced by a multicast fat-tree network of the same size. Therefore, eventually, we may only need to build multicast capability into small commodity switch modules, which involves only minor increases in hardware complexity.

In the multirate network model, a connection can take only a fraction of the total bandwidth of a link, and the available bandwidth left on the link can be used for other connections. For example, assume two connections a and b share a link of bandwidth 1 by time division multiplexing (TDM), meaning that they take turns to send packets. If the packets from a occupy the link $\frac{3}{4}$ of time and b takes other $\frac{1}{4}$ of time, mathematically it is equivalent to saying that "the bandwidth of flow a is 0.75 and the bandwidth of flow b is 0.25."

As mentioned earlier, in order to reduce congestion, many fat-tree DCNs adopt inter-level links with higher bandwidth than links directly connected to servers. To simplify notations, we set the bandwidth of transmitting/receiving links of each server to 1, and denote the bandwidth of inter-switch links normalized by the server link bandwidth as S. We define the *weight* of a link as the sum of the transmission bandwidth of all connections passing through the link. Also, a link is called ω-*idle*, where $0 < \omega \leq 1$, if there is *at least* ω available bandwidth left on the link. Note that the number of servers associated with an edge switch, n, generally ranges from hundreds to thousands due to the large scale of today's data centers. Also, it is impractical to use core switches with very high port bandwidth (e.g., an 128-port 10GigE switch would cost orders of magnitude higher than an 128-port GigE switch [18]), hence, we assume that S is much smaller than n in fat-tree DCNs.

A *multicast connection request with weight* ω is a connection request from a source server to a set of destination servers that requires ω transmission bandwidth. If a connection is destined for more than one destination server on some edge switch, then it is only necessary for the source server to have one connection path to that switch, within which the path can be multicast to as many servers as necessary. Therefore, a multicast connection request can be described in terms of connections between a source server and a set of edge switches along with the transmission bandwidth requirement. Formally, a multirate multicast connection request from a server $i \in I = \{1, 2, \ldots, nr\}$ is denoted as (I_i, ω), where $I_i \subset \{1, 2, \ldots, r\}$ denotes the subset of edge switches to which server i is to be connected in the multicast connection. The weight of the multicast connection, denoted by ω, $0 < \omega \leq 1$, is the transmission bandwidth required by the connection.

We consider a discrete bandwidth model as in [106], in which the weight ω of all connections belongs to a given finite set $B = \{b, 2b, \ldots, Db\}$. To simplify the notation, we shall always assume that $b = 1/D$ is an integer in our analysis, as the case that $1/b$ is not an integer can be derived similarly. The multirate network reduces to a circuit switching network when $b = 1$. We call the number of destination edge switches of (I_i, ω) the *fanout* of the connection and denote it as $|I_i|$. Note that since only the connection paths routed through core switches would affect the nonblocking condition, we can omit the connection paths within the same edge switch in our analysis. Hence, the maximum possible fanout of any multicast connection in a fat-tree DCN is $r - 1$. Meanwhile, we refer to a set of multicast connections as a *multicast assignment*, if the total weight of each transmitting/receiving link in the network does not surpass its bandwidth capacity. *A nonblocking multicast network is a network that can realize all possible multicast assignments without congestion.*

To fulfill a connection request (I_i, ω), the connection paths will be routed through a set of core switches via ω-idle uplinks, then reach the corresponding destination edge switches I_i via a set of ω-idle downlinks. Therefore, we need to look at the network state from the perspective of both uplinks and downlinks. For a given connection request (I_i, ω), we refer to the set of core switches connected by ω-idle uplinks from the edge switch associated with server i as the *available* core switches.

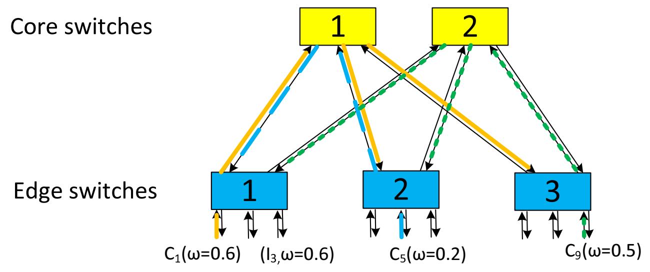


Figure 4.3: A $ftree(2, 3, 3)$ multirate fat-tree network. There are three existing connections in the network $C_1(\omega = 0.6)$, $C_5(\omega = 0.2)$ and $C_9(\omega = 0.5)$ marked by lines of different colors and styles.

Also, to characterize the connection state of a core switch j, $1 \leq j \leq m$, we define the term *destination set*, $M_{j,\omega} \subseteq \{1, 2, \ldots, r\}$, as the set of edge switches that core switch j connects to through downlinks with weight *greater* than $1 - \omega$. In other words, the connection paths to the edge switches in $M_{j,\omega}$ *cannot* be established through core switch j with respect to connection request (I_i, ω).

To better illustrate these notations, we show a small multirate fat-tree network $ftree(2, 3, 3)$ in Fig. 4.3. We set the bandwidth capacity of all the links in the network to 1. There are three existing connections distributed among edge switches $C_1(\omega = 0.6)$, $C_5(\omega = 0.2)$ and $C_9(\omega = 0.5)$, marked by lines of different colors. Now, suppose there is a connection request $(I_3, \omega = 0.6)$ from server 3. We can see that only core switch 2 is *available* for the request, as only the uplink to core switch 2 is 0.6-idle. Also, we can see that the destination sets of core switches are $M_{1,0.6} = \{2, 3\}$ and $M_{2,0.6} = \{1, 2\}$, respectively.

In the next section, we will show that the cost of nonblocking multicast fat-tree DCNs can be significantly reduced through exploring server redundancy in data centers.

4.4 Nonblocking Condition for Multicast Fat-tree DCNs in Redundant Data Centers

In this section, we derive the sufficient multicast nonblocking condition on the number of core switches required in redundant fat-tree DCNs under the multirate model. To make the analysis tractable, we assume that no server failure could occur in the data center in this section. As will be seen in the next section, in case of server failure, through server reassignment, the results in this section will still hold. We first study an ideal case that the fat-tree DCN is 2-redundant, that is, every server has one independent redundant instance provided in the data center network. We then generalize the results to practical fat-tree DCNs, where servers may have different numbers of redundant instances depending on the availability requirement of services provided. We also extend the results to multicast traffic with restricted fanout.

4.4.1 Sufficient Condition for 2-Redundant Fat-tree DCNs

In this subsection, we show that the sufficient multicast nonblocking condition on the number of core switches required can be significantly reduced when the fat-tree DCN is 2-redundant.

In a 2-redundant fat-tree DCN, each server has another independent redundant instance in the network. Given that they are identical instances, i.e., they share the same hardware/software configurations, we can assign either one of the two instances as active server and the other as standby backup. The following lemma shows that in arbitrary 2-redundant fat-tree DCNs, we can always find a way to assign active servers, such that the standby servers and active servers are evenly distributed among edge switches.

Lemma 4.1. *In any 2-redundant fat-tree DCN, it is always possible to assign active servers to edge switches, such that each edge switch has $\frac{n}{2}$ active servers when n is even, or at most $\lceil \frac{n}{2} \rceil$ active servers when n is odd, where n is the number of servers associated with each edge switch.*

Proof. We can model the edge switches and their associated servers in a 2-redundant fat-tree DCN $ftree(m, n, r)$ by a *Server Distribution Graph (SDG)*, $G(V, E)$, according to the following procedures. First, denote edge switch i as node $v_i \in V$ in the SDG, for $1 \leq i \leq r$. For each pair of redundant servers with one server associated with edge switch i and its redundant instance associated with edge switch j, we add an edge $e_k \in E$ between node v_i and node v_j. Note that any two servers associated with the same edge switch cannot be the redundant instance of each other, as each active server and its standby backup server must be independent. Therefore, the resulting SDG for any 2-redundant data center is a multigraph (i.e., multiple edges between two nodes may exist) with each node having a degree of exactly n (i.e., n-regular). An SDG is not necessarily connected, and each component of the graph is also an n-regular multigraph.

We then give an algorithm called *Eulerian Traversal* on the SDG to assign active servers, and show that the lemma holds for arbitrary 2-redundant fat-tree DCNs. First, we consider the case where n is even. In this case, every node in the SDG has an even degree. According to the property of Eulerian graphs, each component of the SDG contains an Eulerian cycle, which is a cycle that traverses every edge in a connected graph exactly once. It is easy to find an Eulerian cycle in $O(|V| + |E|)$ time for any Eulerian graph $G(V, E)$.

After obtaining an Eulerian cycle in each component of the SDG, we proceed as follows.

- For each Eulerian cycle, pick an arbitrary node, say, v_{i_1}, on the circle as the starting point.
- Traverse the circle from v_{i_1} in either clockwise or counterclockwise direction.
- Assign the edge starting from a node in the traversal to that node. For example, for a traversal $(v_{i_1} \overset{e_{i_1}}{\to} v_{i_2} \overset{e_{i_2}}{\to} \cdots \overset{e_{i_{k-1}}}{\to} v_{i_k} \overset{e_{i_k}}{\to} \cdots v_{i_1}$, where v_{i_k} and e_{i_k} are the k^{th} encountered node and edge, respectively, edge e_{i_k} is assigned to node v_{i_k}.

In total, each node v_i will be encountered $n/2$ times during the traversal according to the algorithm, as every node in the graph has a degree of n and each encounter accounts for a degree of 2. In addition, each encounter increases the number of edges assigned to node v_i by 1. Thus, node v_i has a total of $n/2$ edges assigned to it. We then choose the servers corresponding to the assigned edges in the SDG as active servers on edge switch i and the rest of servers as standby servers. Hence, the lemma holds when n is even.

We now prove that the lemma also holds when n is odd. In a 2-redundant fat-tree DCN $ftree(m, n, r)$, each server has exactly one independent redundant instance, thus there must be an even number of edge switches (i.e., r is even) given n is odd. Therefore, the SDG in this case is an n-regular multigraph with an even number of nodes. Grouping two nodes into a pair arbitrarily, we can find $r/2$ pairs of nodes in the SDG. Add a *virtual edge* between the two nodes in each pair, then the graph obtained is an $(n+1)$-regular multigraph. Since $n+1$ is even, the Eulerian traversal algorithm can be applied to the graph. We then ignore all the virtual edges and assign active servers for each edge switch in the same way as that when n is even. Clearly, there are at most $\lceil \frac{n}{2} \rceil$ active servers in each edge switch. $\square$

Lemma 4.1 shows that we can always evenly distribute active servers and stand-by servers among edge switches when the fat-tree DCN is 2-redundant. In a fat-tree DCN, all the simultaneous connections from an edge switch will compete for the same set of core switches when establishing connection paths. Hence, the demand for core switches is the most stringent when some edge switch is *congested*, i.e., all associated servers connected to the switch are active, and requesting broadcast connections with full bandwidth capacity of their transmitting links simultaneously. Such congestion at edge switches accounts for the large number of core switches for nonblocking multicast communication in fat-tree DCNs. Since at most half of the servers associated with each edge switch can be active simultaneously according to Lemma 4.1, the congestion at edge switches can be effectively mitigated. Notice that whether n is even or odd has negligible difference for a reasonably large fat-tree network, thus we will assume n is even in the following sections for clarity.

Suppose there is a new multicast request (I_i, ω) in the $ftree(m, n, r)$ DCN, which is currently providing a set of multicast connections. Next, we focus on the network condition under which the given request can be satisfied. Suppose the new multicast connection request (I_i, ω) is requesting to connect to r' destination edge switches, where $r' = |I_i|, 1 \leq r' < r$ and $\omega \in B$. Connection request (I_i, ω) will be routed to some core switches via a set of ω-idle uplinks, then routed towards the destination edge switches through a set of ω-idle downlinks. We need to look at the network state from the perspective of both uplinks and downlinks. To make the problem tractable, we set a routing constraint that every connection in the network can be routed through at most x $(1 \leq x < r)$ core switches. As will be shown later, such constraint is indeed feasible given a sufficient number of available core switches.

First, we consider the number of uplinks that cannot be used by the connection request, or equivalent, the number of core switches that are *not* available to the given connection request. Let $J_2(\omega, x)$ denote the maximum number of inaccessible core switches regarding the given request (I_i, ω) in a 2-redundant fat-tree DCN, under the condition that every connection in the network is routed through at most x $(1 \leq x < r)$ core switches. We have the following lemma regarding $J_2(\omega, x)$.

Lemma 4.2. *Given a new multicast connection request (I_i, ω) in an arbitrary 2-redundant $ftree(m, n, r)$ DCN, we have*

$$J_2(\omega, x) = \left\lfloor \frac{(n/2 - \omega)x}{S - \omega + b} \right\rfloor .$$

Proof. Without loss of generality, assume the new connection request is from the first active server on the corresponding edge switch. By Lemma 4.1, there are $n/2$ active servers on the edge switch. Since the new connection request (I_i, ω) must be valid, that is, the transmitting link of the corresponding source server and the receiving links of the corresponding destination servers must be ω-idle, the most congested case would be that the transmitting link of the first active server currently carries connections of a total weight $1 - \omega$, and each of the remaining $n/2 - 1$ active servers having multicast connections with full bandwidth of its transmitting link. Since we limit each connection to be routed through at most x core switches, the maximum total bandwidth consumption of all the uplinks by the existing connections out of edge switch i is $(n/2 - \omega)x$. If an uplink carries at least $S - \omega + b$ traffic, it does not have sufficient bandwidth for the new connection request, hence, the corresponding core switch is inaccessible for the new connection request. Consequently, the number of uplinks out of the edge switch that carry a weight of at least $S - \omega + b$ cannot be more than $\left\lfloor \frac{(n/2-\omega)x}{S-\omega+b} \right\rfloor$. That proves the lemma. □

We have considered the network state from the perspective of uplinks. Similarly, we can look at the network state from the perspective of downlinks, as shown in the following corollary.

Corollary 4.1. *Given a new multicast connection request (I_i, ω) in an arbitrary 2-redundant $ftree(m, n, r)$ DCN, for each destination edge switch k in set I_i, there are at most $J_2(\omega, 1)$ downlinks to edge switch k that cannot be used by the connection request.*

Proof. For the same reason as in Lemma 4.2, the total weight of all the connections carried on the receiving links of all active servers in any destination edge switch in I_i is at most $n/2 - \omega$. However, For a given multicast connection, a source server can connect to multiple core switches, whereas a destination server can receive packets from exactly one core switch, hence, $x = 1$. Therefore, we can use $J_2(\omega, 1)$ to denote the number of downlinks with weights greater than $S - \omega$ to each destination edge switch. □

Yang [29] gave the the necessary and sufficient condition that connection request (I_i, ω) can be satisfied through a set of x available core switches without congestion, as shown in the following lemma.

Lemma 4.3. *We can satisfy a multicast connection request* (I_i, ω)*, using some* x $(x \geq 1)$ *available core switches, say,* $j_1, j_2, \ldots, j_x$*, from among the available core switches, if and only if*

$$I_i \cap (\bigcap_{k=1}^{x} M_{j_k,\omega}) = \emptyset.$$

Lemma 4.3 indicates that to satisfy a connection request (I_i, ω), a connection path from the source server to each of the destination edge switches must be available through at least one of the x available core switches. Yang and Masson [29] gave a method to find no more than x available core switches that satisfy the condition in Lemma 4.3. Using a similar technique, we can derive the following lemma.

Lemma 4.4. *Given a new connection request* (I_i, ω) *with fanout* r'*,* $1 \leq r' < r$*, in a 2-redundant* $ftree(m, n, r)$ *DCN, if there exist more than* $m' = J_2(\omega, 1)r'^{1/x}, 1 \leq x \leq r'$ *available core switches, then there will always exist* x *core switches through which this new connection request can be satisfied.*

Proof. Suppose we have m' available core switches for a connection request (I_i, ω). Since we are only concerned with the destination edge switches in the connection request, we intersect the destination sets of these m' core switches with I_i, such that each destination set only includes the edge switches in I_i. The destination sets after intersection are still denoted as $M_{j,\omega}, j = 1, 2, \ldots, m'$ for convenience.

We apply a *minimum cardinality* rule when choosing the set of core switches to satisfy the connection request. In the first iteration, we find the core switch with the destination set of minimum cardinality, denoted as $M_{j_1,\omega}$. From Corollary 4.1, we know that there are at most $J_2(\omega, 1)$ downlinks from core switches to each destination edge switch that cannot be used by the connection request. Therefore, there are at most $J_2(\omega, 1)r'$ elements in all $M_{j,\omega}$s. The cardinality of the chosen destination set cannot be more than the average cardinality of all the destination sets, thus

$|M_{j_1,\omega}| \leq \frac{J_2(\omega,1)r'}{m'}$. In the next iteration, we concentrate on the destination switches that cannot be reached through $M_{j_1,\omega}$. To do this, we intersect $M_{j_1,\omega}$ with each of the destination sets and obtain another m' sets, denoted as $M^{(1)}_{j,\omega}, j = 1, 2, \ldots, m'$. It is clear that there are at most $J_2(\omega, 1)|M_{j_1,\omega}|$ elements in all these sets. We again choose the set with minimum cardinality $M_{j_2,\omega}$. By the same token, we have

$$|M_{j_2,\omega}| \leq \frac{J_2(\omega,1)|M_{j_1,\omega}|}{m'} \leq \left(\frac{J_2(\omega,1)}{m'}\right)^2 r'.$$

In general, in the k^{th} iteration ($1 \leq k \leq x$),

$$|M_{j_k,\omega}| \leq \left(\frac{J_2(\omega,1)}{m'}\right)^k r'.$$

In order to satisfy the connection request using no more than x core switches, we must have

$$|M_{j_x,\omega}| \leq \left(\frac{J_2(\omega,1)}{m'}\right)^x r' < 1.$$

By solving the inequality, we obtain that

$$m' > J_2(\omega, 1)r'^{1/x}.$$

The lemma is thus proved. □

We are now ready to give the sufficient nonblocking condition on the number of core switches m for a 2-redundant multicast fat-tree DCN $ftree(m, n, r)$.

Theorem 4.1. *A 2-redundant $ftree(m, n, r)$ DCN is nonblocking for any multicast assignments under the multirate model, if*

$$m > \min_{1 \leq x < r} \max_{\omega \in B} \{J_2(\omega, x) + J_2(\omega, 1)(r-1)^{1/x}\} \tag{4.1}$$

Proof. From Lemma 4.2, we know that $J_2(\omega, x)$ is the maximum possible number of core switches that are not available for a new connection request (I_i, ω), given that every connection is routed through at most x core switches. Also, Lemma 4.4 shows that a connection request (I_i, ω) with fanout r' will be satisfied through x available core switches, given that there are more than $J_2(\omega, 1)r'^{1/x}$ available core switches. Since the fanout of any connection in the fat-tree DCN is at most $r - 1$, we have that for a given x, if

$$m > J_2(\omega, x) + J_2(\omega, 1)(r-1)^{1/x}$$

we can satisfy the new connection request. Considering all possible values of x and ω, we obtain the nonblocking condition for 2-redundant multicast fat-tree DCNs. $\square$

In addition, we can represent the nonblocking condition in Theorem 4.1 in terms of basic network parameters, as shown in the following corollary.

Corollary 4.2. *A 2-redundant $ftree(m, n, r)$ DCN is nonblocking for any multicast assignments under the multirate model, if*

$$m > \min_{1 \le x < r} \left\{ \left\lfloor \frac{(n/2-1)x}{S+b-1} \right\rfloor + \left\lfloor \frac{(n/2-1)}{S+b-1} \right\rfloor (r-1)^{1/x} \right\} \tag{4.2}$$

Proof. Based on our assumption, n is much larger than S in fat-tree DCNs. Thus, $J_2(\omega, x)$ is maximized when $\omega = 1$, which proves the corollary. $\square$

For given n and r, we can use Theorem 4.1 to find an optimum x such that a minimum m can be determined for nonblocking multicast communication in arbitrary 2-redundant fat-tree DCNs. Next, we extend the result to general redundant fat-tree DCNs.

4.4.2 Sufficient Condition for General Redundant Fat-tree DCNs

In this subsection, we extend the result to general redundant fat-tree DCNs, in which servers may have different numbers of redundant instances.

From Lemma 4.1, we know that it is always possible to distribute active servers and standby servers evenly in arbitrary 2-redundant fat-tree DCNs. Next, we show that the lemma can be extended to arbitrary k-redundant ($k > 2$) fat-tree DCNs. We begin with the assumption that $k = 2^i$.

Lemma 4.5. *In an arbitrary k-redundant ($k = 2^i$) fat-tree DCN, it is always possible to assign active servers, such that each edge switch has at most $\frac{n}{k}$ active servers.*

Proof. Notice that Lemma 4.1 is a special case of the above lemma when $i = 1$. We denote the server assignment for arbitrary 2-redundant fat-tree DCNs in Lemma 4.1 as *Assignment(1)* problem, and the server assignment for arbitrary k-redundant $(k = 2^i)$ fat-tree DCNs as *Assignment(i)* problem. We prove the lemma by recursively using the Eulerian traversal algorithm. For better illustration, we also show an example of server assignment problem in a simple 4-redundant fat-tree DCN in Fig. 4.4.

For a server a in a k-redundant fat-tree DCN, there are k identical instances (including itself) independent of each other. We denoted the k instances as $a_1, a_2, \ldots, a_k$ (see Fig. 4.4(a) for an example). Since $k = 2^i$, we can group two identical instances together in a pair arbitrarily, and obtain 2^{i-1} pairs. Without loss of generality, we denote the obtained server pairs as $\{a_1, a_2\}, \ldots, \{a_{k-1}, a_k\}$.

For assignment purpose, we first assume that the two servers in each of the 2^{i-1} pairs are identical instances to each other, but servers from different pairs are not identical. For example, as shown in Fig. 4.4(b), a_1 and a_2 are treated as identical instances to each other (yellow nodes), but they are not considered identical to a_3 and a_4 (green nodes). Under this assumption, we obtain an *Assignment(1)* problem, where each server has another identical instance. This problem can be solved by constructing a SDG and using the Eulerian traversal algorithm, as in Fig. 4.4(c). After that we have each edge switch with $n/2$ associated active servers by Lemma 4.1. In the next iteration, we try to assign active servers only from the active servers selected in the previous iteration. In this way, each server now has $\frac{k}{2}$ identical instances (including itself), and the problem becomes an *Assignment(i-1)* problem. For example, in Fig. 4.4(d), if we only consider the active servers chosen, the problem would become an *Assignment(1)* problem.

Clearly, *Assignment(i)* problem can be solved in i iterations. Note that each iteration reduces the number of active servers on each edge switch by half. Therefore, there are $\frac{n}{2^i} = \frac{n}{k}$ active servers in each edge switch after i iterations. The lemma is thus proved. [1] □

Lemma 4.5 shows that a higher redundancy level further reduces the congestion at edge switches in a fat-tree DCN. In fact, the condition that $k = 2^i$ in Lemma 4.5

[1] We assume n is divisible by k implicitly. The proof for the case that n is not divisible by k is a trivial extension from the original proof, thus is omitted.

can be removed. Let $J_k(\omega, x)$ denote the maximum number of inaccessible core switches for the new request (I_i, ω) in an arbitrary k-redundant fat-tree DCN, under the condition that every connection is routed through at most x core switches. We can have the following lemma regarding $J_k(\omega, x)$.

Lemma 4.6. *Given a k-redundant $ftree(m, n, r)$ DCN, and a new multicast request (I_i, ω), we have*

$$J_k(\omega, x) = \left\lfloor \frac{(\frac{n}{2^i} - \omega)x}{S - \omega + b} \right\rfloor,$$

where $i = \arg\max_j \{j | 2^j \leq k\}$.

Proof. It is straightforward to see that the server distribution graph (SDG) of a k-redundant fat-tree DCN always contains a subgraph that is the SDG of some 2^i-redundant fat-tree DCN, $i = \arg\max_j \{j | 2^j \leq k\}$. In other words, we can always arbitrarily choose 2^i out of k identical instances for each server in a k-redundant data center, then only assign active servers from these servers. The resulting problem would be an $Assignment(i)$ problem. By Lemma 4.5, the lemma holds. □

In practice, depending on the availability requirement of services provided, different servers in data centers usually have a different number of redundant instances [108]. Next, we will generalize the result to general redundant fat-tree DCNs where servers have different redundancy levels.

Theorem 4.2. *A general redundant $ftree(m, n, r)$ DCN is nonblocking for any multicast assignments under the multirate model, where every server is* at least *k-redundant, if*

$$m > \min_{1 \leq x < r} \left\{ \left\lfloor \frac{(\frac{n}{2^i} - 1)x}{S + b - 1} \right\rfloor + \left\lfloor \frac{(\frac{n}{2^i} - 1)}{S + b - 1} \right\rfloor (r - 1)^{1/x} \right\} \tag{4.3}$$

where $i = \arg\max_j \{j | 2^j \leq k\}$.

Given Lemma 4.6, the above theorem can be similarly proved as Corollary 4.2.

Theorem 4.2 shows that the sufficient nonblocking condition for general redundant multicast fat-tree DCNs depends on the least redundant servers. It is worth pointing out that Theorem 4.2 can be used flexibly in building practical fat-tree DCNs. For example, if it is not cost-effective to provide redundant backups for every server, data center owners have the option to select a set of highly redundant servers that host critical services, and make sure the fat-tree DCN is sufficiently equipped to guarantee nonblocking multicast communication for these servers based on Theorem 4.2. As long as high priority services are allowed to overtake low priority services when realizing connections, the network is guaranteed to be nonblocking for the selected set of servers (or equivalently, the selected set of services).

Given the values of n and r in a general redundant $ftree(m, n, r)$ DCN, we could use Theorem 4.2 to find the minimum m to build a nonblocking multicast fat-tree DCN. However, it is also of interest to determine a bound on m as an explicit function of n and r. The following Corollary addresses this issue.

Corollary 4.3. *A general redundant $ftree(m, n, r)$ DCN is nonblocking for any multicast assignments under the multirate model, where every server is* at least *k-redundant, if*

$$m > 3\left\lceil \frac{(\frac{n}{2^i}-1)}{S+b-1}\right\rceil \frac{\log r}{\log\log r} \tag{4.4}$$

where $i = \arg\max_j\{j|2^j \leq k\}$

Proof. The condition shown in Inequality (5.3) can be relaxed to

$$m > \min_{1\leq x<r}\left\{\left\lceil \frac{(\frac{n}{2^i}-1)}{S+b-1}\right\rceil (x+r^{1/x})\right\}. \tag{4.5}$$

For any constant $u > 0$, we let $x = \frac{\log r}{u\log\log r}$ in (5.4). Then,

$$r^{1/x} = r^{\frac{u\log\log r}{\log r}} = (\log r)^{\frac{1}{u}}.$$

Letting $u = 2$, (5.4) can be written as

$$m > \left\lceil \frac{(\frac{n}{2^i}-1)}{S+b-1}\right\rceil \left[2\frac{\log r}{\log\log r} + (\log r)^{\frac{1}{2}}\right].$$

Since $\frac{\log r}{\log\log r}$ is of higher order than $(\log r)^{\frac{1}{2}}$, we have that

$$m > 3\left\lceil \frac{(\frac{n}{2^i}-1)}{S+b-1}\right\rceil \frac{\log r}{\log\log r}$$

is sufficient for nonblocking multicast communication in fat-tree DCNs. □

From the above proof, we can see that the bound given in Corollary 4.3 is in the same order as the original nonblocking condition in Theorem 4.2. Since a single-rate, circuit switching fat-tree network can be viewed as a special instance of the multirate network model, our nonblocking condition can also be applied to circuit switching fat-tree networks. If we set the the value of b and S to 1, as well as disregard server redundancy ($i = 0$) in Inequity (4.4), the multirate fat-tree network would reduce to a simple circuit switching network, and the condition in Corollary 4.3 would become $m > 3(n-1)\frac{\log r}{\log\log r}$, which is consistent with the bound for nonblocking multicast Clos networks obtained in [29].

4.4.3 Extensions

In this subsection, we generalize the above results to restricted fanout multicast assignments.

Notice that traffic in data center networks is usually "skewed," i.e., a server may have frequent communication with a small group of destination servers while rarely communicating with others [8]. Hence, it is also meaningful to consider the sufficient nonblocking condition for multicast assignments with restricted fanout, in which each multicast connection can have connection paths to at most d, $1 \leq d < r$, edge switches. We will state the corollaries to the above results that address such generalizations without a proof, as they can be easily derived from the theorems and corollaries above.

Corollary 4.4. *A general redundant $ftree(m, n, r)$ DCN is nonblocking for any multicast assignments under the multirate model, if every server is* at least *k-redundant and each multicast connection has at most d fanout, ($1 \leq d < r$), the network is nonblocking if*

$$m > \min_{1 \leq x < d} \left\{ \left\lfloor \frac{(\frac{n}{2^i} - 1)x}{S + b - 1} \right\rfloor + \left\lfloor \frac{(n/2 - 1)}{S + b - 1} \right\rfloor d^{1/x} \right\}. \tag{4.6}$$

In particular, this condition can be written as

$$m > 3 \left\lceil \frac{(\frac{n}{2^i} - 1)}{S + b - 1} \right\rceil \frac{\log d}{\log\log d} \tag{4.7}$$

where $i = \arg\max_j \{j | 2^j \leq k\}$.

It is interesting to note that for the restricted fanout case, when $d = 1$ we have the special case of unicast capability of fat-tree DCNs. The above result then takes the form of sufficient strictly nonblocking condition on unicast capability of fat-tree DCNs in the presence of server redundancy.

Corollary 4.5. *Setting $d = 1$ in Corollary 4.4 yields*

$$m > 2 \left\lfloor \frac{\frac{n}{2^i} - 1}{S + b - 1} \right\rfloor . \tag{4.8}$$

where $i = \arg\max_j \{j | 2^j \leq k\}$.

Compared with [106], which states that the nonblocking condition on m for a multirate unicast Clos network is $m > 2 \left\lfloor \frac{n-1}{S+b-1} \right\rfloor$, Corollary 4.5 shows that the nonblocking condition on m for unicast fat-tree DCNs can be reduced through exploring server redundancy as well.

4.5 Server Reassignment in Faulty Redundant Data Centers

So far we have assumed that no failure could occur in the data center for tractability. In this section, we consider faulty data centers, where one or more active servers could fail at any time. First, we show that although server failure causes uneven congestion among edge switches, which could invalidate the nonblocking conditions established previously, such uneven congestion can be re-balanced by reassigning servers among edge switches. Then, we propose a graph modeling approach for redundant faulty data centers, and show that the *optimal server reassignment* problem in case of server failure can be transformed to finding the *minimum cost network flow* in the corresponding auxiliary graph. Finally, we prove that a server reassignment can always be found for arbitrary redundant faulty data centers, such that the previous nonblocking condition still holds.

4.5.1 Uneven Congestion Caused by Server Failure

There are various sources of failures in a data center, including links, switches and servers. Maintaining nonblocking communication under a single core switch failure is fairly simple: one extra core switch is sufficient to ensure that the condition in Theorem 2 still holds upon the failure. Here, we focus on server failure, which is the very purpose why server redundancy is provided. As indicated in [109], server failure has become the norm rather than exception in large scale data centers. When one or more active servers fail in a redundant data center, they immediately fail over to their standby backups, which become active and replace the failed servers. The failover mechanism masks the external observation of server failure, thus guarantees high availability of data centers. However, it can also cause uneven congestion in some edge switches.

For better illustration, we give a simple example in Fig. 4.5, which shows a 2-redundant fat-tree DCN consisting of four edge switches, with each edge switch connecting to four servers. Each active server is denoted by a colored node, and its backup server is denoted by a blank node with the same letter. Now, assume that active server e in the 3^{rd} edge switch fails, as shown in Fig. 4.5(a). Its backup server located at the 4^{th} switch would have to become active in order to replace the failed server. In this case, three servers associated with the 4^{th} switch are active, which leads to heavy congestion in that switch. As indicated in Lemma 4.1, when there is no server failure in a 2-redundant data center, only half of the servers associated with each edge switch are active, which is the key to reducing multicast nonblocking condition. Therefore, the uneven congestion brought by server failure could cause congestion in a fat-tree DCN even if it satisfies the previous nonblocking condition.

As mentioned above, when backup servers replacing failed active servers are located in an already congested edge switch, the uneven congestion in the faulty data center could invalidate the nonblocking condition. However, a key observation is that when an active server fails over to its backup, it aggravates the congestion at the edge switch associated with the backup, but reduces the congestion at the edge switch of the failed server. For example, only one server connected to the 3^{rd} switch is active after the server failure, as shown in Fig. 4.5(a). On the other hand, it takes relatively little effort to shift the operation of an active server to its backup, as all active servers and their backups are identical instances that require constant synchronization in order to guarantee successful failover. Hence, we can re-balance the uneven congestion through reassigning servers, that is, shifting the operation and traffic load of certain active servers to their backups. For example, as shown in Fig. 4.5(b), after shifting the operation of active server h in the 2^{nd} switch and active server g in the 4^{th} switch to their backups, each edge switch again has two

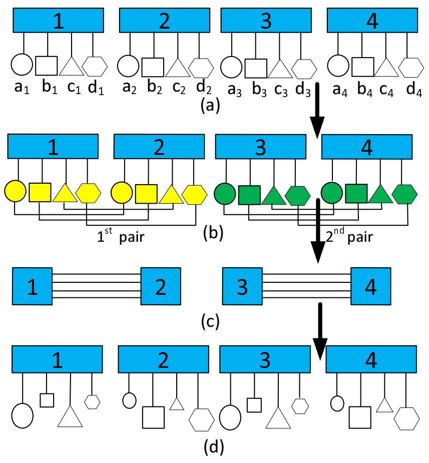


Figure 4.4: Server assignment in a simple 4-redundant fat-tree DCN. (a) Server distribution, in which each server has 4 independent identical instances (including itself) denoted by the same shape; (b) Arbitrarily pair 4 identical instances of each server. Assume servers from different pairs are not identical instances, the resulting problem becomes *Assignment(1)*; (c) Constructing the SDG;(d) Solve the problem using Eulerian traversal algorithm, then only consider the active servers (denoted by larger shapes in the figure) chosen for next iteration.

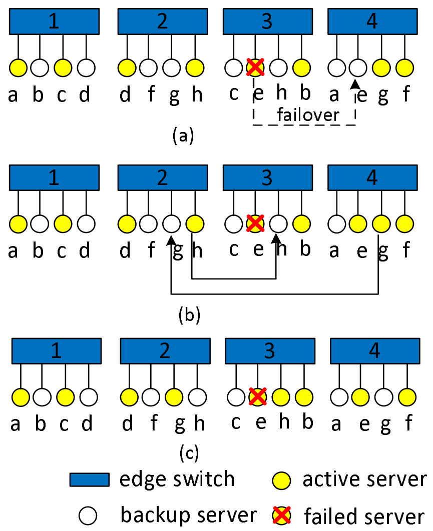


Figure 4.5: A faulty 2-redundant fat-tree DCN. Each active server is denoted by a colored node, and its backup server is denoted by a blank node with the same letter. (a) Active server e in the 3rd edge switch fails over to its backup in the 4th switch. (b) Shifting the operation of the active server h in the 2nd switch and active server g in the 4th switch to their backups. (c) Each edge switch again has two active servers.

active servers and the DCN is nonblocking as shown in Fig. 4.5(c).

The above example shows that it is possible to reassign a set of servers in a redundant faulty data center, such that the obtained nonblocking condition still holds in case of server failure. Note that, each transaction is essentially a failover process, which is different from virtual machine (VM) migration. In VM migration, the memory image of a VM must be transferred to a server (possibly chosen from a pool of alive servers) before the VM can be restarted, which incurs large overhead. On the other hand, as active servers and their backups are constantly synchronized to ensure instant failover in an HA data center, each transaction incurs much less service downtime and overhead, given that the cost of failover is amortized through periodical synchronization. Meanwhile, most server failures are transient, lasing from seconds to minutes [109], and only a few failures are permanent. The data center operator can choose to initiate server reassignment only when the benefit of reassignment outweighs the overhead.

4.5.2 Optimal Server Reassignment

In this subsection, we see how to obtain such reassignment optimally and efficiently. We call the process of shifting the operation of an active server to its backup a *transaction*. Then a *server reassignment* can be defined as a set of transactions, through which the uneven congestion caused by server failure is re-balanced, in the sense that the number of active servers in each edge switch is no more than that given in Lemma 4.5, when there is no server failure in the data center. Since each transaction would incur some overhead and briefly disrupt the corresponding applications, it is desirable that a server reassignment has as few transactions as possible. Therefore, we define an *optimal server reassignment* as a server reassignment that consists of the *minimum* possible number of transactions. Finding the optimal server reassignment in a redundant faulty DCN is nontrivial, especially in a large-scale data center network with the possibility of multiple servers failing at the same time. Next we give a graph modeling approach for redundant faulty DCNs, and show that the optimal server reassignment problem can be transformed to finding the *minimum cost network flow* in the auxiliary graph.

A redundant faulty fat-tree DCN $ftree(m, n, r)$ can be modeled as a directed graph $G'(V, E)$ according to the following procedures. First, denote edge switch i as node $v_i \in V$, for $1 \leq i \leq r$. If x active servers in edge switch i have backup

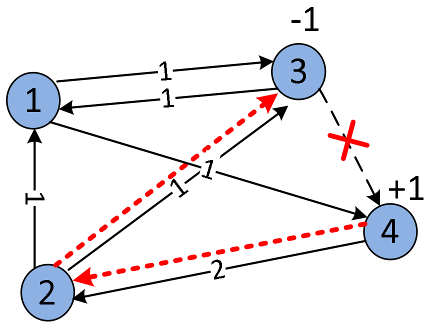


Figure 4.6: The auxiliary graph $G'(V,E)$ for the 2-redundant faulty data center in Fig. 4.5. The capacity of each edge is denoted by the number inside. After the failover, edge $e_{3,4}$ is removed, with node 4 and node 3 assigned $+1$ and -1 demand, respectively. A feasible flow from source node 4 to sink node 3 can be found by sending one unit of flow through edge $e_{4,2}$ and edge $e_{2,3}$.

servers located at edge switch j, we add a directed edge $e_{i,j}$ from node i to node j with its *capacity* $u_{i,j}$ set to x. Then, for each node $v_i \in V$, we assign an integer b_i representing the available demand at that node, which is set to 0 by default. When a failed active server in edge switch i fails over to its backup in edge switch j, we reduce b_i and $u_{i,j}$ by 1, and increase node demand b_j by 1 at the same time. Clearly, the change in b_i denotes the change in the number of active servers connected to switch i caused by server failure, and the total demand of all the nodes in the auxiliary graph $G'(V,E)$ is always 0, i.e., $\sum_{i\in V} b_i = 0$. If $b_i > 0$, then node i is a *source* node, and if $b_i < 0$, node i is a *sink* node. Otherwise, if $b_i = 0$, node i is an *intermediate* node. Fig. 4.6 shows the auxiliary graph $G'(V,E)$ for the 2-redundant fat-tree DCN in Fig. 4.5.

Suppose each edge $e_{i,j}$ carries $f_{i,j}$ units of flow in $G'(V,E)$. Then a network flow is called *feasible* if the required flows from all the source nodes to the sink nodes are delivered, and the following *flow bound* constraint (5.4) and *flow balance* constraint (4.10) are satisfied.

$$0 \le f_{i,j} \le u_{i,j}, \forall e_{i,j} \in E \tag{4.9}$$

$$\sum_{j:e_{i,j}\in E} f_{i,j} - \sum_{j:e_{j,i}\in E} f_{j,i} = b_i, \forall v_i \in V \tag{4.10}$$

Associate each edge $e_{i,j}$ with a cost $c_{i,j}$, then the *minimum cost flow* problem is to find a feasible flow, if any, with minimum cost, that is,

$$\textbf{minimize} \sum_{e_{i,j} \in E} c_{i,j} f_{i,j}$$

Next, we show that server reassignment problem in a redundant faulty data center can be transformed to finding a feasible network flow in the auxiliary graph $G'(V, E)$.

Lemma 4.7. *There exists a server reassignment when one or more active servers fail in a redundant faulty data center, if and only if a feasible network flow can be found in its auxiliary graph $G'(V, E)$.*

Proof. Based on the *integrality theorem* of network flow, as long as all the capacities in a flow network are integers, there is a feasible network flow, if any, consisting of only integers. Given such an integer feasible network flow, a server reassignment can be found as follows. If there are $f_{i,j}$ units of flow on edge $e_{i,j}$, we shift $f_{i,j}$ active servers in switch j to switch i. For example, as shown in Fig. 4.6, a feasible flow is found by sending a unit of flow through edges $e_{4,2}$ and $e_{2,3}$. Correspondingly, we shift one active server in edge switch 4 to its backup in edge switch 2, then shift one active server in edge switch 2 to its backup in edge switch 3.

Note that, edge switch i has a total of $u_{i,j}$ active servers whose backup servers are connected to edge switch j. Since the flow bound constraint states that $f_{i,j} \leq u_{i,j}$, we can always find $f_{i,j}$ active servers in edge switch j with backup connected to switch i. Therefore, the server reassignment is valid.

We now show that such a server reassignment is able to re-balance the uneven congestion among edge switches. Denote the total units of flow that enter node v_i as in_i and the total units of flow that leave node v_i as out_i. Clearly, out_i is the number of active servers in switch i whose operation is shifted to other switches by the server reassignment, and in_i is the number of backup servers in switch i that become active by the server reassignment. Therefore, if $in_i - out_i > 0$, edge switch i will have $in_i - out_i$ additional servers after the reassignment, otherwise, it will have $out_i - in_i$ fewer active servers. Since the flow balance constraint in (4.10) states that $out_i - in_i = b_i$, and the change in b_i is the change in the number of active servers connected to switch i caused by server failure, we can see that after the reassignment, each edge switch again has the same number of active servers as that when there is no server failure, in other words, the uneven congestion caused by server failure is re-balanced.

On the other hand, suppose we have a valid server reassignment that re-balances the uneven congestion among edge switches in a redundant faulty data center. For each transaction in the server reassignment, we can send one unit of flow in the corresponding directed edge. It is easy to check that the resulting flow is feasible. Thus the lemma is proved. □

From the proof of Lemma 4.7, we can see that each server reassignment in a redundant faulty fat-tree data center corresponds to a feasible network flow in the auxiliary graph $G'(V, E)$. Moreover, the number of transactions in a server reassignment is equal to the total number of flows over all the edges, i.e.,$\sum_{e_{i,j} \in E} f_{i,j}$. Therefore, finding the optimal server reassignment can be easily transformed to the *minimum cost network flow* problem, as shown in the following corollary.

Corollary 4.6. *Assign each edge in the auxiliary graph $G'(V, E)$ a uniform cost 1, then the optimal server reassignment problem is equivalent to finding the network flow with the minimum cost, i.e.,* $\min \sum_{e_{i,j} \in E} f_{i,j}$.

Minimum cost flow problem is one of the classic network flow problems that have received extensive attention in last few decades. Efficient algorithms with polynomial time complexity have been proposed to solve the problem. Given a graph $G(V, E)$ with unity cost on each edge, [110] gave the minimum cost flow algorithm with the best time complexity $O(|V||E| \log^2 |V|)$. As the auxiliary graph $G'(V, E)$ for a fat-tree DCN $ftree(m, n, r)$ consists of r nodes, each of which has n edges, the optimal server reassignment problem can be solved in $O(r^2 n \log^2 r)$ time. Readers may refer to [111] for a comprehensive review of minimum cost flow problem.

We have shown that the optimal server reassignment problem in a redundant faulty data center can be transformed to finding the minimum cost network flow in the auxiliary graph. However, it remains to be seen that whether a valid server reassignment can always be found for arbitrary redundant data centers in the event of server failure. For easy presentation, we begin the analysis with 2-redundant faulty data centers and then extend it to general redundant faulty data centers.

First, the auxiliary graphs of arbitrary 2-redundant faulty fat-tree DCNs share a common property that will be useful in our analysis, which is described below.

Property 4.1. *For each node v_i in the auxiliary graph $G'(V, E)$ of a 2-redundant faulty fat-tree DCN $ftree(m, n, r)$, we have*

$$\sum_{j:e_{i,j} \in E} u_{i,j} - \sum_{j:e_{j,i} \in E} u_{j,i} = b_i \tag{4.11}$$

Proof. For each node $v_i \in V$, let oc_i denote the total capacity of all the edges out of v_i, that is, $oc_i = \sum_{j:e_{i,j} \in E} u_{i,j}$. Similarly, let ic_i denote the total capacity of all the edges towards v_i, $ic_i = \sum_{j:e_{j,i} \in E} u_{j,i}$. We can see that oc_i and ic_i are equal to the number of active servers and backup servers in edge switch i, respectively. Suppose there is no server failure in the data center at the beginning, then each edge switch has $n/2$ active servers and backup servers by Lemma 4.1, that is, $oc_i = ic_i = n/2, \forall v_i \in V$. Whenever an active server in switch i fails over to switch j, $u_{i,j}$ and b_i is reduced by 1, and b_j is increased by 1. We can see that (4.11) holds for both nodes v_i and v_j. Thus, the property is proved. □

Lemma 4.8. *We can always find a server reassignment, when one or more active servers fail in a 2-redundant faulty fat-tree DCN.*

Proof. We prove this lemma by showing that a feasible network flow exists in the auxiliary graph $G'(V, E)$ for an arbitrary 2-redundant faulty fat-tree DCN.

As mentioned previously, the total demand of all the nodes $\sum_{i \in V} b_i$ in the auxiliary graph $G'(V, E)$ is zero, which is one of the necessary conditions for flow feasibility in $G'(V, E)$.

We define an *S-T cut* as a partition of V, in which S and T are two nonempty, disjoint subsets of V that satisfy $S \cup T = V$. The *capacity* of an *S-T* cut is defined by

$$C(S, T) = \sum_{i:v_i \in S} \sum_{j:v_j \in T} u_{i,j}$$

According to the *max flow min cut* theorem [111], a directed graph $G(V, E)$ has feasible flow(s), if and only if, for an arbitrary *S-T* cut, the total demand of the nodes in S is no larger than the capacity of the cut, i.e., $\sum_{v_i \in S} b_i \leq C(S, T)$. Next, we prove this is true for the auxiliary graph $G'(V, E)$ of any 2-redundant faulty fat-tree DCN.

Without loss of generality, we pick an arbitrary *S-T* cut. Based on Property 4.1 of auxiliary graph $G'(V, E)$, the total demand of nodes in the set S, $\sum_{i:v_i \in S} b_i$, can be written as

$$\begin{aligned}
&\sum_{i:v_i\in S}\Big(\sum_{j:v_j\in S} u_{i,j} + \sum_{j:v_j\in T} u_{i,j} - \sum_{j:v_j\in S} u_{j,i} - \sum_{j:v_j\in T} u_{j,i}\Big)\\
&= \sum_{i:v_i\in S}\sum_{j:v_j\in T} u_{i,j} - \sum_{i:v_i\in S}\sum_{j:v_j\in T} u_{j,i} \qquad (4.12)
\end{aligned}$$

As the capacity of the *S*-*T* cut is the first term in (4.12), $C(S,T) \geq \sum_{i:v_i\in S} b_i$ stands for an arbitrary *S*-*T* cut. The lemma is thus proved □

Next, we show that Lemma 4.8 can be extended to general redundant faulty data centers.

Corollary 4.7. *We can always find a server reassignment, when one or more active servers fail in a general redundant faulty fat-tree DCN.*

Proof. As shown in Theorem 4.2, the maximum number of active servers in an edge switch depends on the *least* redundant servers. Assume that every server is at least k-redundant in the data center. Then a server reassignment exists, if we can prove that a server reassignment can always be found in arbitrary 2^i-redundant $(i \geq 1)$ data centers, where $i = \arg\max_j\{j|2^j \leq k\}$.

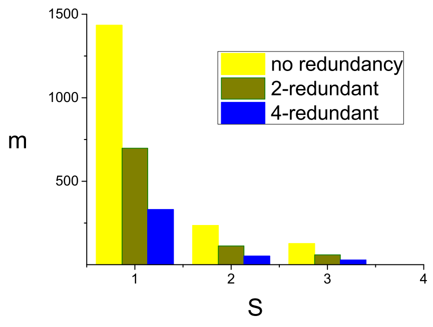
As shown in the proof of Lemma 4.5, the server assignment problem in the k-redundant $(k = 2^i)$ fat-tree DCN can be solved by recursively applying Eulerian Traversal algorithm i times. Each iteration only considers the edges chosen in the previous iteration, and reduces the number of edges by half. In the last iteration, all the servers corresponding to the edges recursively chosen from the last $i-1$ iterations would form a 2-redundant data center. Since a server reassignment always exists in arbitrary 2-redundant data centers, we can find a server reassignment for any k-redundant ($k = 2^i, i \geq 1$) data centers. The corollary is thus proved. □

Corollary 4.7 shows that with server reassignment, the number of active servers in each edge switch remains the same as in Lemma 4.5. Since other lemmas in the previous section still stand, Theorem 4.2 holds for arbitrary redundant faulty data centers with server reassignment.

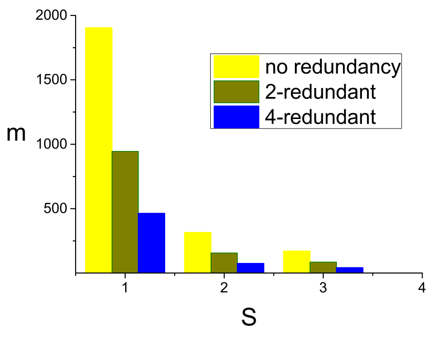
Theorem 4.3. *The sufficient multicast nonblocking condition in Theorem 4.2 still holds for redundant faulty fat-tree DCNs with server reassignment.*

Theorem 4.3 states that the uneven congestion caused by server failure in any faulty redundant data center can always be re-balanced through server reassignment, such that the DCN can remain nonblocking in case of server failure given that it satisfies the condition established in Theorem 4.2.

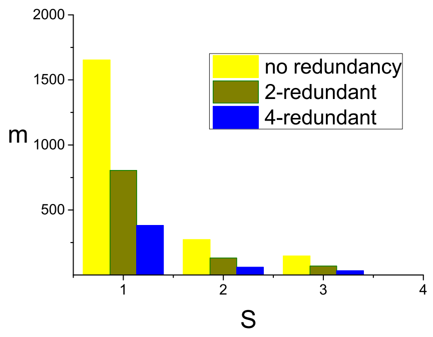
4.6 Comparison of Network Costs



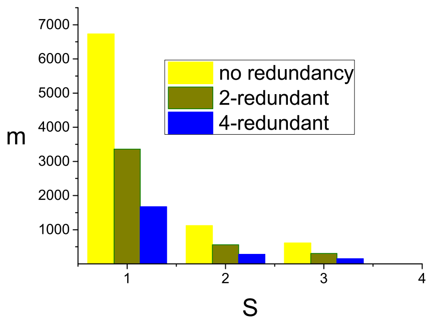
(a) $n = 40, r = 128$



(b) $n = 128, r = 40$



(c) $n = 40, r = 512$



(d) $n = 512, r = 40$

Figure 4.7: Sufficient multirate multicast nonblocking condition on the number of core switches m in $ftree(m, n, r)$ DCNs with different sizes, redundancy levels and normalized inter-level link bandwidths S.

In this section, we compare the sufficient multicast nonblocking condition on the number of core switches m in fat-tree DCNs $ftree(m, n, r)$ with different sizes and redundancy levels. We also evaluate the impact of the inter-level link bandwidth S on the multicast nonblocking condition.

The main advantage of fat-tree networks is to build large interconnects from smaller switches. Hence, we adopt commodity edge switches with 40 duplex ports for connecting servers, i.e., $n = 40$. Considering the number of servers inside a data center can vary from several hundreds to tens of thousands, we investigate a medium size fat-tree DCN with $r = 128$ and a large size fat-tree DCN with $r = 512$. We set the bandwidth of transmitting/receiving link of each server to 1, and denote the normalized inter-level link bandwidth as S. The smallest bandwidth fraction that a connection can carry, b, is set to 0.2.

Fig. 4.7 shows the sufficient multirate multicast nonblocking condition on the number of core switches m in $ftree(m, n, r)$ DCNs with different sizes, redundancy levels and normalized inter-level link bandwidths S. It can be observed from Fig. 4.7 that, for fat-tree DCNs of both sizes, the sufficient number of core switches m to support nonblocking multicast communication can be significantly reduced through exploring server redundancy. When the $ftree(m, n, r)$ DCN is 2-redundant, m can be reduced by around 50%, compared to the case without considering server redundancy. When the fat-tree DCN is at least 4-redundant, m can be further reduced by 75%. Considering the fact that core switches are usually the most expensive components in data center networks due to their large port count and high port speed, the cost saving towards building nonblocking multicast fat-tree DCNs through exploring server redundancy is very substantial.

It can be observed that the combination of n and r also has a significant impact on the multicast nonblocking condition for a fat-tree DCN $ftree(m, n, r)$. As can be seen in Fig. 4.7, for fat-tree DCNs of the same size, the nonblocking condition on m is much smaller, when we use more edge switches with fewer servers connected to each switch (larger r and smaller n) than the opposite case (smaller r and larger n). However, since a core switch is connected to all the edge switches, having a large number of edge switches r also means that we need large core switches, which are usually much more expensive than small commodity switches. Thus, it is essential that a cost-effective combination of n and r is chosen when building a fat-tree DCN.

To support a large number of ports in core switches, the method to build a level-2 nonblocking multicast fat-tree network can be recursively applied to build more levels of nonblocking fat-tree networks. In this case, server redundancy can reduce the cost of nonblocking multi-level fat-tree networks even further. For example, to obtain a level-3 nonblocking fat-tree network, a level-2 nonblocking fat-tree network block can be used to replace each core switch in the original level-2 fat-tree network. Since each core network block supports all types of multicast traffic without contention, it can be shown by induction that the recursively built larger network will also be nonblocking for multicast. The total cost saving by exploring server redundancy in this case would account for both requiring a fewer number of core network blocks and building each nonblocking core network block at lower cost, which is even more substantial than building a basic level-2 nonblocking fat-tree network.

Fig. 4.7 also shows that the normalized inter-level link bandwidth S has a significant impact on the multicast nonblocking condition. The sufficient number of core switches m for nonblocking multicast fat-tree DCNs drops drastically when S increases. The reason is that adopting higher inter-level link bandwidth allows more connections to be carried on each inter-level link simultaneously, thus reduces the number of core switches m required for nonblocking multicast communication. However, having high inter-level link bandwidth may require non-commodity, high speed core switches, which are generally orders of magnitude more expensive than commodity switches.

Overall, the factors that affect the cost of a nonblocking multicast fat-tree DC-N $ftree(m, n, r)$ include server redundancy level, the combination of n and r, the market price of switches, the hierarchy of the network and the inter-level link bandwidth. Though the price of switches changes constantly with the development of technology [5], based on the results obtained in this chapter, data center designers can always find the most economical way to build nonblocking multicast fat-tree DCNs based on the current market trend.

4.7 Conclusions

In this chapter, we have explored server redundancy in HA data centers to reduce the cost of nonblocking multicast fat-tree DCNs. We present a multirate network model that accurately describes the communication environment of fat-tree DCN-s. Initially, we consider fault-free data centers for tractability. We show that the sufficient condition on the number of core switches for nonblocking multicast communication can be significantly reduced when the fat-tree DCN is 2-redundant. We also extend the results to general redundant fat-tree DCNs where servers may have different numbers of redundant instances. Then, we consider redundant faulty data centers where one or more servers can fail at any time. We formulate the server reassignment problem as a network flow problem and prove that the obtained nonblocking condition under fault-free assumption still holds in the event of server failure with server reassignment. Based on the theorems, we provide an efficient multicast routing algorithm with linear time complexity to configure multicast connections in fat-tree DCNs. We also compare the sufficient multicast nonblocking condition on the number of core switches m in fat-tree DCNs with different sizes and redundancy levels. The comparison results demonstrate the substantial cost saving of exploring server redundancy.

Chapter 5

On-line Multicast Scheduling with Bounded Congestion in Fat-tree Data Center Networks

In the previous chapters, we mainly focus on reducing the cost of nonblocking multicast fat-tree DCNs. Nonblocking networks guarantee congestion-free communication, however, they also incur very high hardware cost. On the other hand, network oversubscription is a commonly adopted technique in practical data centers to avoid under-utilization of network resource and reduce cost. An oversubscribed fat-tree DCN requires much fewer core switches, thus incurs significantly lower cost than the nonblocking counterparts [18]. However, without an efficient flow scheduling algorithm that appropriately routes multicast flows to achieve traffic load balance, heavy congestion may occur throughout the network in an oversubscribed fat-tree DCN, which prevents full utilization of link bandwidth and causes unpredictable network performance.

In this chapter, we study multicast flow scheduling in oversubscribed fat-tree DCNs, where multicast flow requests arrive one by one without a priori knowledge of future traffic. To address the drastic traffic fluctuation in data centers, we consider a very general traffic model called *hose* traffic model, where the only assumption is that the total bandwidth demand of traffic that enters (leaves) an ingress (egress) link of each server at any time is bounded by the capacity of its network interface card. We present a low-complexity on-line multicast flow scheduling algorithm for fat-tree DCNs. The algorithm can achieve bounded congestion and efficient bandwidth utilization under any arbitrary sequence of multicast flow requests that satisfy the hose model. We also derive the bound on congestion that the algorithm can achieve in a fat-tree DCN.

The rest of the chapter is divided into six sections. Section 5.1 provides the introduction of research issue. Section 5.2 discusses the related work. Section 5.3 briefly introduces the background of fat-tree DCNs and discusses the requirements of multicast flow scheduling in fat-tree DCNs. Section 5.4 gives the network model and some useful notations. Section 5.5 presents the BCMS algorithm. Section 5.6 analyzes the bound on congestion and algorithm complexity. Section 5.7 provides the performance evaluation results. Finally, Section 5.8 concludes the chapter.

5.1 Introduction

Though Fat-tree DCNs have the potential to deliver huge aggregated bandwidth through providing rich path multiplicity between any pair of hosts, uneven traffic load distribution, a common phenomenon in fat-tree DCNs usually caused by inefficient traffic scheduling, prevents full utilization of network bandwidth [15]. Besides degraded network utilization, unbalanced traffic load distribution also causes variability in the bandwidth offered by the DCN to tenant applications, leading to unpredictable network performance.

To illustrate the detrimental effects of uneven traffic load distribution in fat-tree DCNs, we show a simple communication pattern in a small fat-tree DCN with every link of 1Gbps bandwidth capacity in Fig. 5.1. Suppose there are four flows, A through D, each having a source server and a destination server located at different edge switches and demanding 1Gbps bandwidth. We notice that flow A and flow B are routed through the same link towards core switch 2, and flow C and flow D are routed through the same link from core switch 3, which causes heavy congestion among certain core switch links. As a result, the bandwidth obtained by each flow is capped at 500Mbps. In this example, all flows could have reached their bandwidth demand of 1Gbps with the improved scheduling: flow A could have been forwarded through core switch 1 and flow D could have been switched through core switch 4. We can see that, due to unbalanced traffic distribution, the network suffers a 50% bisection bandwidth loss, and all flows suffer long delay and high packet drop

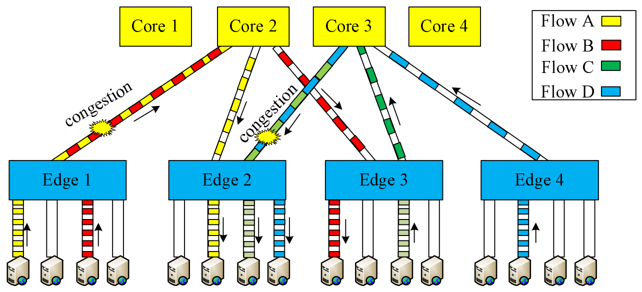


Figure 5.1: Unbalanced traffic load distribution leads to reduced bisection bandwidth and unpredictable network performance. Unused links omitted for clarity.

due to congestion. The bandwidth available to each flow could suffer from further reduction if the scheduler cannot identify heavy congestion in the network and keep assigning incoming flows on congested core switch links. Therefore, it is essential for a scheduling algorithm adopted by a fat-tree DCN to achieve good traffic load balance.

Due to the volatility and unpredictability of data center traffic [8–10], we focus on on-line multicast scheduling, where multicast flow set-up requests arrive over time and future demands are unknown. In order to address the drastic traffic volume fluctuation, which is quite common in data centers [8–10], we will consider multicast scheduling under a very general traffic model called *hose* traffic model, where the only assumption is that the total bandwidth demand of traffic that enters (leaves) an ingress (egress) link of each server at any time is bounded by the capacity of its network interface card.

First, we will present Bounded Congestion Multicast Scheduling (BCMS), an on-line multicast flow scheduling algorithm with very low-complexity, which is able to guarantee bounded congestion (the ratio of the aggregated bandwidth demands of flows routed through a link to the link capacity) as well as maintain good traffic load balance and efficient network utilization for fat-tree DCNs. Then, we will derive the bound on congestion on any link in a fat-tree DCN under arbitrary multicast traffic patterns satisfying hose model under the proposed BCMS algorithm.

Given that BCMS possesses the desirable property of guaranteeing the worst-case congestion across the network even under the most malicious traffic patterns, our results can also greatly enhance network performance predictability, which is desired by many tenant applications. Finally, we will evaluate the performance of the BCMS algorithm under various types of traffic patterns in an event-driven DCN simulator. The simulation results demonstrate that BCMS achieves superior performance in terms of aggregated network bandwidth delivered to hosts and evenness of traffic load distribution.

5.2 Related Work

Multicast in DCNs has drawn a considerable amount of attention. Several work have been proposed to address reliability and scalability issues in the transmission of multicast packets in data centers [33, 34]. As data centers usually adopt commercial switches that cannot provide high reliability, an efficient packet repairing scheme [33] was proposed, which relies on unicast to retransmit dropped multicast packets caused by switch buffer overload or switch failure. In the meantime, because the commercial switches in data centers can only hold a small number of multicast group states, bloom filter [34] is proposed to compress the multicast forwarding table in switches, which improves the scalability of the data center network to support more multicast groups.

Multicast scheduling has also been the subject of research in the context of traditional networking environments such as ATM and MPLS networks, which can be roughly divided into two categories [35–38]. The first category of the work targets efficient spanning tree construction for multicast groups and develops low-complexity heuristic algorithms for solving Steiner tree problem, which is known to be NP-Hard in general graphs. The Steiner tree problem in a general graph G requires a minimum-cost tree spanning a given set of vertices X in the graph. The TM algorithm [35] and the KMB algorithm [36] are two well-known polynomial time approximation algorithms for the Steiner tree problem in general graphs, both of which can achieve an approximation guarantee of factor 2 in an undirected graph and an approximation guarantee of $O(k)$ in a directed graph, where $k = |X|$.

The second category of the work focuses on *competitive* algorithm design, which compares the performance of on-line multicast routing algorithms to that of the off-line optimal algorithms that know the whole sequence of traffic requests in advance [37, 38]. A common goal in this line of research is to achieve low *competitive ratio*, which is the worst-case performance ratio between on-line and optimal off-line algorithms, in terms of certain performance metric. In [37], an online multicast routing algorithm was proposed that has a competitive ratio of $O(\log n \log d)$ with respective to the maximum link congestion, where n is the network size and d is the maximum size of a multicast group. In [38], an algorithm was given that achieves $O(\log n)$ competitive ratio with respect to the total number of accepted multicast group requests under the constraint that the congestion on any link is not allowed to exceed 100%.

Note that all the aforementioned algorithms were developed in the context of traditional networks, thus may not be suitable for implementation in fat-tree DCNs. In this chapter, we consider on-line multicast flow scheduling in fat-tree data center networking environment. We explore novel traffic control techniques available to data centers, such as OpenFlow framework, and the topological property of fat-tree networks to achieve efficient bandwidth utilization in designing the algorithm. Another significant advantage of our algorithm compared with previous algorithms is that it can guarantee bounded congestion on any link in a fat-tree DCN even under the most malicious traffic patterns. As network congestion is a determining factor in DCNs to deliver predictable performance and bandwidth guarantees to tenants, which are indispensable for many cloud applications, such bounded congestion property of our algorithm is particularly desirable for data center communications.

5.3 Preliminary

In this section, we introduce some notations describing fat-tree DCNs and hose traffic model. Then we identify the main requirements that a multicast scheduling algorithm must satisfy in data center networking environment.

5.3.1 Fat-Tree DCNs

We have given the description of fat-tree DCNs in the previous chapter. For clarity, we give a brief review of its notations. As shown in Fig. 5.2, a high-level fat-tree is recursively built from the basic level-2 fat-tree blocks, and any property of a level-2 fat-free is still pertaining to a high-level fat-tree, we focus on level-2 fat-tree DCNs in this chapter. In a level-2 fat-tree DCN, an edge switch connects to each core switch through two directed links. We denote the links from edge switches to core

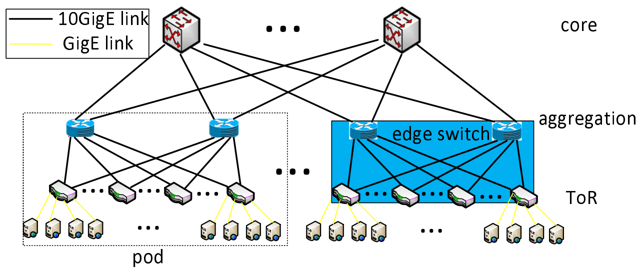


Figure 5.2: A level-3 fat-tree DCN, which can be reduced to a level-2 fat-tree if ToR switches and aggregation switches in each pod are considered as a nonblocking edge switch.

switches as *uplinks* and the links from core switches to edge switches as *downlinks*. Both uplinks and downlinks are indistinguishably referred to as *core switch links* in some occasions. Also, each server connected to an edge switch port through two directed links, denoted as *egress link* and *ingress link*, respectively.

5.3.2 Hose Traffic Model

It has been shown that DCNs exhibit highly variable traffic over time and space[8–10]. Over a long period, DCN traffic shows a clear diurnal pattern: traffic peaks during the day and falls off at night, there can be an order of magnitude difference between the peak and minimum load in the DCN during a 24-hour time period. DCN traffic is also volatile and unpredictable over short periods. The traffic matrix in a DCN shifts frequently and its overall volume changes dramatically in short instants.

Due to the highly volatile nature of data center traffic, we consider a very general traffic model referred to as *hose* traffic model, where each host server is assigned a maximum ingress bandwidth capacity and a maximum egress bandwidth capacity. Any traffic matrix that is consistent with the ingress/egress bandwidth capacity must be included in this model. Hose traffic model was originally proposed to specify the bandwidth requirements for point-to-point communication in a Virtual Private Network (VPN) [112], and has been used to describe unicast traffic in data centers [13, 14] due to its great flexibility in describing volatile traffic conditions. We can see that all the allowed traffic matrices in hose model constitute a polytope, and we will take into account the "worst" multicast traffic patterns allowed by hose model in our algorithm.

5.3.3 Multicast Flow Scheduling in DCNs

Data center networking environment presents numerous challenges in multicast traffic scheduling. On the other hand, novel traffic control techniques available to DCNs, like recently developed OpenFlow control framework [44, 45], offer many useful features that can greatly facilitate the design of multicast scheduling algorithms. Next, we identify some requirements and opportunities presented by data center networking environment in designing multicast scheduling algorithms for fat-tree DCNs.

- Scheduling in flow granularity: OpenFlow offers scalable, flow-level control of switching by abstracting each switch (data plane) as a flow-table. Each multicast flow can be managed by simply adding a flow entry in relevant switches along its route in an OpenFlow network. In this way, switches are spared from maintaining numerous routing states and group membership information associated with multicast traffic in the network as in IP multicast, which solves the scalability problem of managing multicast traffic.

- Feasibility of centralized, adaptive scheduler: OpenFlow is a centralized control framework, where all the flows set-up requests are handled by a centralized scheduler. OpenFlow also has an effective network-wide status monitoring mechanism, thus it can quickly responded to network congestion [**?**]. With centralized-decision making and global knowledge of network condition, OpenFlow allows adaptive, convergence-free flow route creation, which can greatly simplify multicast scheduling.

- Avoid traffic rerouting and splitting: Though traffic splitting and traffic rerouting can be used for better load balance purpose [15, 60], they cause heavy control overhead. In addition, splitting a flow into multiple paths may lead to out-of-order transmission, which is undesirable for many applications. Rerouting existing traffic flows risks disruption of important cloud services that may be time-sensitive (e.g., MapReduce). Therefore, a scheduling algorithm should be able to achieve efficient network utilization without counting on traffic splitting and rerouting.

Table 5.1: Notations used in the BCMS algorithm

Notation	Definition
m	Number of core switches
n	Number of hosts per edge switch
r	Number of edge switches
S	Normalized bandwidth capacity of core switch links
i	Source edge switch of the multicast flow
D	Set of destination edge switches of the multicast flow
ω	Reserved bandwidth of the multicast flow
$M_j^{C,\omega}$	Inaccessible set of core switch j
C	Bound on congestion of a network under the hose model
x	Maximum number of core switches a flow routes via
b	Granularity of flow bandwidth
$U_{C,\omega}$	Maximum number of (C, ω)-unavailable uplinks
$D_{C,\omega}$	Maximum number of (C, ω)-unavailable downlinks

- Computational requirements: Due to the large scale of today's data center, a myriad of flows are injected to the network per second, which could easily overwhelm the processing capability of the central flow scheduler. On the other hand, DCN traffic consists of a small number of long-lived, throughput-sensitive "elephant" flows and numerous delay-sensitive "ant" flows. As "ant" flows usually have very small bandwidth demand, they incur minimal impact on network condition. Hence, one way to lower the stringent computational requirement of the central scheduler is selective scheduling, that is, let the central scheduler handle only "elephant" flows, and apply OpenFlow flow-match wildcards or hash-based routing for "ant" flows at local edge switches [15, 17]. Through selective scheduling, we can have a reasonable time budget (several milliseconds) for computing the route of each flow [17] .

5.4 Network Model

We adopt the same multirate network model as the previous chapter. In this section, we introduce some notations that will be used in the remainder of the chapter, which are summarized in Table 5.1.

We consider a level-2 fat-tree DCN, denoted as $ftree(m, n, r)$, with m core switches at the top, and r edge switches, each of which connects n host servers, at the bottom. We assume all switches in such a fat-tree DCN have *multicast capability*. Similar to the previous chapter, we set the bandwidth capacity of ingress/egress links of each server to 1, and denote the normalized bandwidth capacity of core switch links as $S(\geq 1)$.

We consider a centralized scheduler that keeps track of available bandwidth capacity on every link of the network. The input to the scheduler is a sequence of multicast flow set-up requests, and the scheduler determines the route of each multicast flow. The flow set-up requests arrive one at a time and there is no knowledge of the characteristics of future demands. We assume flow arrivals satisfy hose traffic model, which dictates that the total bandwidth demand of all the flows entering/leaving a server must not exceed the bandwidth capacity of its ingress/egress link at any time.

A multicast flow can be abstracted as a triple (i, D, ω), in which $i \in \{1, 2, \ldots, r\}$ is the edge switch connecting to the source server and $D \subseteq \{1, 2, \ldots, r\}$ is the set of edge switches that the flow is destined for. $\omega, 0 \leq \omega \leq 1$ is the amount of bandwidth demanded by the multicast flow. The number of destination edge switches of a multicast flow (i, D, ω), $|D|$, is called its *fanout*. Note that since servers connected to the same edge switch can freely communicate with each other, we can ignore intra-edge switch traffic, hence, the maximum possible fanout of any multicast flow is $r - 1$. The reason why we do not specify the exact source and destination servers of a multicast flow is that any packet entering an edge switch can multicast within the switch to as many other ports as necessary given the multicast capability of the switch.

We denote the *congestion* of a link as the ratio of the total bandwidth demand of all the flows being routed on the link to its bandwidth capacity. Note that it is possible for the congestion of a link to exceed 100%, which means that the total bandwidth demand of traffic flows occupying a network link may surpass the link capacity. In this case, the network is oversubscribed and packet losses could occur. As oversubscription is widely adopted in DCNs to avoid network resource underutilization and reduce cost [18], links with congestion that is larger than 100% are common in DCNs and contending flows on these congested links receive only a fraction of their demanded bandwidth. For example, suppose two flows, each with

0.75 bandwidth demand are routed through a link with capacity 1, then each flow receives 0.5 bandwidth.

The goal of this chapter is to design an on-line multicast scheduling algorithm that can guarantee bounded congestion on any links in a fat-tree DCN. Such an algorithm has many desirable properties. First, by setting a limit on how much traffic load can be put on each link, the algorithm can achieve balanced traffic load distribution, thus improve the utilization of network bandwidth resource. Second, since the maximum congestion on any link is bounded, the algorithm can ensure that the network does not suffer from heavy link congestion even under worst-case traffic conditions, which greatly enhances network predictability. We will describe the algorithm in detail in the next section.

5.5 On-line Multicast Scheduling with Bounded Congestion

In this section, we present Bounded Congestion Multicast Scheduling (BCMS), an on-line multicast scheduling algorithm that is able to achieve bounded congestion as well as efficient bandwidth utilization even under worst-case traffic conditions in a fat-tree DCN.

5.5.1 Notations

Before explaining the details of the algorithm, we introduce some notations that will be used.

Suppose that there is an incoming multicast flow set-up request (i, D, ω) from edge switch i. We say a core switch link (C, ω)-*available* if its congestion is no larger than C *after* the new flow (i, D, ω) is added to the link. Otherwise, we say the link (C, ω)-*unavailable*. For example, suppose a link with bandwidth capacity 1 is carrying two flows, each demanding 0.6 bandwidth. The link is $(2, 0.7)$-available, because its congestion would remain below 2 after an incoming flow with 0.7 demanded bandwidth is added.

With respect to the new request (i, D, ω), we refer to the set of core switches connected by (C, ω)-*available* uplinks from edge switch i as (C, ω)-*available* core switches, denoted by T. Also, we define the term *inaccessible set*, $M_j^{C,\omega} \subseteq \{1, 2, \ldots, r\}$, as the subset of destination edge switches in D that core switch j is currently connecting to via (C, ω)-*unavailable* downlinks. In other words, the paths to the edge switches in $M_j^{C,\omega}$ *cannot* be established through core switch j with respect to the incoming flow (i, D, ω) without causing the congestion of corresponding downlinks to exceed C.

5.5.2 Bounded Congestion Multicast Scheduling Algorithm

In this subsection, we explain in detail the Bounded Congestion Multicast Scheduling (BCMS) algorithm.

BCMS takes a centralized approach by leveraging the OpenFlow control framework of data centers, in which a central flow scheduler collects the bandwidth demand of incoming flows, monitors the data center network condition (i.e., the current available bandwidth of network links), computes the routing paths for each flow, and configures the switches. Suppose there is an incoming multicast flow (i, D, ω), we explain how BCMS schedules the incoming flow step by step in the following. The details of BCMS are also summarized in Table 5.2.

One of the main features of BCMS is that it can guarantee bounded congestion on any links in a fat-tree DCN. We call the maximum congestion on any links in a fat-tree DCN under BCMS for an arbitrary sequence of flow set-up requests that satisfy hose model the *congestion bound* of the network, represented by C. As will be shown in the next section, the value of C in a fat-tree DCN $ftree(m, n, r)$ can be solely determined by the configuration of the fat-tree DCN, that is, the values of m, n, r and core switch link bandwidth S. Therefore, for a given fat-tree DCN $ftree(m, n, r)$, we can use the congestion bound C as a guideline for BCMS to find appropriate routes for multicast flows, such that the total traffic load put on each core switch link is limited.

Since every multicast flow must route towards a set of core switches through uplinks first, the first step of BCMS identifies the set of (C, ω)-available uplinks out of source edge switch i, and correspondingly, the set of (C, ω)-available core switches T. After being sent to core switches, the flow will be forwarded to all destination edge switches in D via downlinks. The second step of BCMS iteratively finds an appropriate subset of core switches from T, through which (i, D, ω) can be

Table 5.2: Description of Bounded Congestion Multicast Scheduling (BCMS)

```
Bounded Congestion Multicast Scheduling
Input: Incoming flow $(i, D, \omega)$, current network condition,
bound on congestion $C$
Output: Connection paths that deliver $(i, D, \omega)$.

Step 1: //check the condition of uplinks out of source edge switch $i$
Identify the set of $(C, \omega)$-available core switches with respect to
$(i, D, \omega)$, denoted as $T$;

Step 2: //select appropriate core switches
do {
   Find the core switch(es) in $T$ with minimum cardinality
   inaccessible set;
   if there are multiple core switches found
      then select the one that would deliver the maximum bandwidth
      to corresponding destination edge switches;
   End if
   Remove destination edge switches that can be reached by the
   selected core switch from $D$;
   Update the inaccessible set of remaining core switches in $T$;
}
while There are still destination edge switches remaining in $D$;
Establish connection path between source edge switch $i$ to
selected core switches, denoted as $T'$;

Step 3: //further balance traffic load
for every destination switch of flow $(i, D, \omega)$, do{
   Find the core switch in $T'$ that delivers maximum bandwidth;
   Connect the edge switch and the corresponding core switch;
}
Deliver $(i, D, \omega)$ through the established path;
Update congestion of all involved links;
End
```

delivered via (C, ω)-available downlinks. The goal is to find the set of core switches to deliver (i, D, ω), such that the schedule does not cause the congestion of any link in the network to exceed C as well as achieves efficient bandwidth utilization and load balance.

For the second step, we adopt a greedy strategy called *minimum cardinality s-trategy* proposed in [29], which chooses the core switch with the inaccessible set of minimum cardinality in each iteration. In other words, the core switches chosen in each iteration can send the flow to the most remaining destination edge switches without causing the congestion of any link to exceed C. The underlying rationale of such a strategy is that we should utilize the multicast capability of core switches to duplicate the multicast flow as much as possible, which reduces the total bandwidth consumption of multicast flows and allows more network resources to better accommodate future traffic. At the end of the iteration, we update the set of remaining destination edge switches D by removing the destination edge switches that can be reached by the core switch selected in this iteration via (C, ω)-available downlinks. The iteration continues till D is empty. We denote the set of core switches chosen as T'. As will be shown later, T' can always be found for any incoming multicast flow request, provided it satisfies the hose model.

Notice that the congestion bound C corresponds to the “worst” traffic patterns in the hose model. Hence, it is highly likely that we find multiple (C, ω)-available core switches with minimum cardinality inaccessible set in each iteration, especially under non-congested traffic condition, because an idle link and a fairly congested link could both be considered as (C, ω)-available especially when C is large. Therefore, we need to take extra measures in order to maintain good traffic load balance. To achieve this objective, if multiple core switches are found with minimum cardinality inaccessible set in one iteration, BCMS evaluates the congestion in each core switch found, and breaks the tie by choosing the one that would deliver the maximum total bandwidth to the remaining destination edge switches via (C, ω)-available downlinks. For simplicity, we assume *proportional fairness* when calculating the actual bandwidth a flow receives in a congested link, which means that the bandwidth is distributed to each contending flow proportionally to its bandwidth demand.

After step 2, we obtain a set of core switches T' such that every edge switch in D can find at least one (C, ω)-available downlinks from some core switches in T'. The final step of BCMS further improves traffic load balance by letting each edge switch in D connect to the core switch in T' that would deliver maximum bandwidth to that edge switch. After the connection paths for (i, D, ω) are established, BCMS updates the congestion of all involved links in the network, and sends configuration signals to corresponding switches.

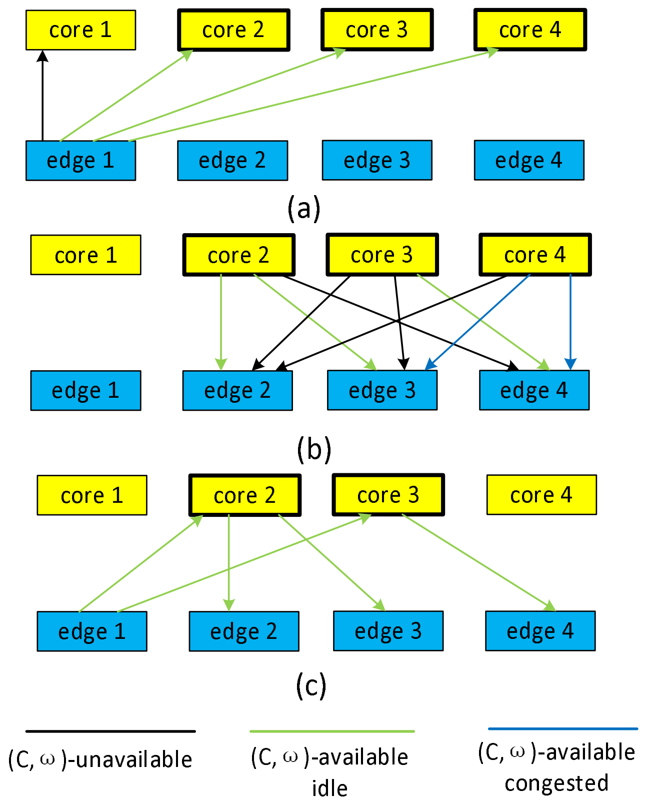


Figure 5.3: Scheduling process of an incoming flow $(1, \{2, 3, 4\}, \omega)$. (a) Check uplinks out of source edge switch 1; (b) Iteratively find a set of core switches with minimum cardinality inaccessible set; (c) Scheduling result.

5.5.3 Scheduling Example

For illustration purpose, we give a small scheduling example in a fat-tree DCN $ftree(4, 4, 4)$. Suppose the congestion bound of the fat-tree DCN is C, and there is an incoming flow $(1, \{2, 3, 4\}, \omega)$. Next, we show how BCMS schedules this flow step by step, as shown in Fig. 5.3. For clarity, only relevant links are drawn in each step.

As shown in Fig. 5.3(a), BCMS first checks the condition of uplinks out of source edge switch 1, and finds the set of (C, ω)-available core switches, marked by bold outlines. In the second step, BCMS iteratively finds the (C, ω)-available core switch with the minimum cardinality inaccessible set. As shown in Fig. 5.3(b), the inaccessible sets of core switches 2, 3 and 4 are $\{4\}$, $\{2, 3\}$ and $\{2\}$, respectively.

In the first iteration, BCMS finds both core switches 2 and 4 with the minimum cardinality inaccessible set, and breaks the tie by choosing core switch 2, because the downlinks from core switch 2 to destination edge switches 2 and 3 are relatively less congested than that of core switch 4. After choosing core switch 2, only destination edge switch 4 remains and the algorithm updates the inaccessible sets of core switches 3 and 4, which are both empty. Then BCMS breaks the tie by choosing core switch 3, as it has less congested downlink towards edge switch 3. Hence, core switches 2 and 3 are chosen in the second step to route the flow. In the final step, every destination edge switch selects one of the two core switches with less congested links to establish the connection path, and the schedule of the flow is shown in Fig. 5.3(c).

5.6 Theoretical Analysis

In this section, we first prove that BCMS guarantees bounded congestion in a fat-tree DCN for an arbitrary sequence of multicast flow requests, provided that they satisfy hose traffic model. We then derive the congestion bound C in a fat-tree $ftree(m, n, r)$. We also apply the results to a special case where only unicast traffic exists. Finally, we analyze the time complexity of BCMS.

Before deriving the congestion bound, we make two reasonable assumptions. First, to simplify notations, we assume the bandwidth demand ω of each flow belongs to a finite set of discrete values $B = \{b, 2b, \ldots, Db\}$. Also, we assume that $1/b$ is an integer and $Db = 1$ in our analysis. The analysis below can be easily extended to the case where flow bandwidth demand is continuous and $1/b$ is not an integer. Moreover, when b is sufficiently small, the discrete bandwidth model is an accurate approximation for flow bandwidth demand in practical DCNs, where each flow consists of fix-sized packets.

Second, we assume that every multicast flow request will be routed through *at most* x $(1 \leq x \leq r)$ core switches under the BCMS algorithm. In other words, the number of core switches chosen in the second step of the BCMS algorithm $|T'| \leq x$. As will be proved later, this assumption always holds with the value of x being determined by the configuration of the fat-tree DCN $ftree(m, n, r)$.

Now, we move on to deriving the congestion bound C under BCMS in a given fat-tree DCN $ftree(m, n, r)$. Recall that the inaccessible set of core switch j, $M_j^{C,\omega}$, is the set of edge switches in D that cannot be reached through the core switch with respect to an incoming multicast flow (i, D, ω), the sufficient and necessary network condition for BCMS to deliver (i, D, ω) while keeping the congestion of all network links bounded by a certain number C is given in the following lemma.

Lemma 5.1. *A fat-tree DCN $ftree(m, n, r)$ can deliver an incoming multicast flow (i, D, ω) without causing the congestion of any links to exceed C, if and only if some x $(x \geq 1)$ (C, ω)-*available *core switches with respect to the flow, say, $j_1, j_2, \ldots, j_x$, can be found, whose inaccessible sets satisfy the following condition.*

$$\bigcap_{k=1}^{x} M_{j_k}^{C,\omega} = \emptyset \tag{5.1}$$

Proof. Clearly, to route an incoming multicast flow (i, D, ω) while keeping congestion of all links bounded by C, there must exist connection paths that consist of (C, ω)-available core switch links from source edge switch i to all destination edge switches in D. In other words, the core switch links chosen to route (i, D, ω) must be (C, ω)-available. Suppose we have some $x(x \geq 1)$ (C, ω)-available core switches. The flow can be successfully established *if and only if* an (C, ω)-available downlink to every destination edge switch in D can be found from *at least* one of the x core switches. Therefore, Equation (6.1) gives the necessary and sufficient condition for BCMS to deliver a multicast flow (i, D, ω) while keeping the congestion of every link bounded by C in a fat-tree DCN. Hence, the lemma is proved. □

With respect to the incoming multicast flow (i, D, ω), we use $U_{C,\omega}$ to denote the maximum number of (C, ω)-unavailable uplinks associated with source edge switch i, or equivalently, the number of core switches that are *not* available to (i, D, ω) under constraint that the congestion of any links cannot exceed C. We also use $D_{C,\omega}$ to denote the maximum number of (C, ω)-unavailable downlinks associated with each destination edge switch in set D. Then, we have the following lemma regarding $U_{C,\omega}$ and $D_{C,\omega}$.

Lemma 5.2. *Suppose there is an incoming multicast flow (i, D, ω) in a $ftree(m, n, r)$ DCN. We have the maximum number of (C, ω)-unavailable uplinks out of source edge switch i*

$$U_{C,\omega} = \left\lfloor \frac{(n-\omega)x}{C \cdot S + b - \omega} \right\rfloor$$

Also, the maximum number of (C, ω)*-unavailable downlinks* $D_{C,\omega}$ *associated with each destination edge switch* k *in set* D *is* $\left\lfloor \frac{n-\omega}{C \cdot S+b-\omega} \right\rfloor$.

Proof. According to hose traffic model, the aggregated bandwidth demand of flows being transmitted/received by each server cannot surpass the egress/ingress link capacity. Since there are n servers connected to each edge switch, the maximum total bandwidth demand of existing flows being transmitted from edge switch i is $n - \omega$. Also, according to our assumption, each flow will be routed through at most x core switches, hence, the maximum total bandwidth demand imposed to all the uplinks out of edge switch i is $(n - \omega)x$. Recall that each core switch link has bandwidth capacity S. Thus, if a link is (C, ω)-unavailable, then the total bandwidth demand of existing flows carried by the link must be at least $C \cdot S + b - \omega$. Consequently, the maximum possible number of (C, ω)-unavailable uplinks connected to the corresponding edge switch of (i, D, ω) is $\left\lfloor \frac{(n-\omega)x}{C \cdot S-\omega+b} \right\rfloor$.

By the same token, the total bandwidth demand of existing flows being received by edge switch k in set D is no more than $n - \omega$. The difference here is that a flow can only pass from *one* downlink to the ingress link of a server. Thus, the maximum number of (C, ω)-unavailable downlinks associated with edge switch k $D_{C,\omega}$ is equivalent to the condition for $U_{C,\omega}$ with $x \equiv 1$, which can be denoted by $\left\lfloor \frac{n-\omega}{C \cdot S+b-\omega} \right\rfloor$. The lemma is thus proved. $\square$

Next, we analyze the minimum number of (C, ω)-available core switches required to successfully deliver an incoming multicast flow (i, D, ω) while keeping the congestion of every link in the fat-tree DCN bounded by C, that is, satisfy the condition in Lemma 5.1.

Lemma 5.3. *Given an incoming multicast flow* (i, D, ω) *with fanout* r', $1 \leq r' \leq r$, *if there are at least* $m' = \left\lfloor \frac{n-\omega}{C \cdot S+b-\omega} \right\rfloor r'^{1/x}, 1 \leq x \leq r'$, (C, ω)*-available core switches, then BCMS can always find no more than* x (C, ω)*-available core switches through which the flow can be delivered while keeping the congestion of every link bounded by* C *in the fat-tree DCN.*

Proof. Suppose there are m' (C, ω)-available core switches for an incoming multicast flow request (i, D, ω) with fanout r'. Without loss of generality, we assume that the flow is destined for edge switches $1, 2, \ldots, r'$. Next, we show that BCMS can always find x (C, ω)-available core switches, whose inaccessible sets satisfy the condition in Lemma 5.1.

From Lemma 5.2, we know that there are at most $D_{C,\omega}$ downlinks from core switches to each destination edge switch that cannot be used by the flow request, which means that there are at most $D_{C,\omega}$ 1s, 2s, $\ldots$, r's in all inaccessible sets $M_j^{C,\omega}$s. Hence, initially there are at most $D_{C,\omega}r'$ elements in total among all $M_j^{C,\omega}$s.

Recall that the second step of BCMS adopts the minimum cardinality strategy when choosing the set of core switches to route the flow. In the first iteration, we find the (C, ω)-available core switch with the inaccessible set of minimum cardinality, denoted as $M_{j_1}^{C,\omega}$. The cardinality of the chosen inaccessible set cannot be more than the average cardinality of all the inaccessible sets, thus $|M_{j_1}^{C,\omega}| \leq \frac{D_{C,\omega}r'}{m'}$. Without loss of generality, we assume $M_{j_1}^{C,\omega}$ contains elements $1, 2, \ldots, |M_{j_1}^{C,\omega}|$.

In the next iteration, we focus on the destination switches left in D that cannot be reached through the selected core switch. To do this, we intersect $M_{j_1,\omega}$ with each of the inaccessible sets and obtain another m' sets. As the intersection of two sets is the set that contains common elements belonging to both sets, there are at most $D_{C,\omega}$ 1s, 2s, $\ldots$, $|M_{j_1}^{C,\omega}|$s among the m' inaccessible sets after the intersection. Hence, there are at most $D_{C,\omega}|M_{j_1}^{C,\omega}|$ elements in all these sets. We again choose the set with minimum cardinality $M_{j_2}^{C,\omega}$. By the same token, we have

$$|M_{j_2}^{C,\omega}| \leq \frac{D_{C,\omega}|M_{j_1}^{C,\omega}|}{m'} \leq \left(\frac{D_{C,\omega}}{m'}\right)^2 r'$$

In general, in the k^{th} iteration ($1 \leq k \leq x$), we have

$$|M_{j_k}^{C,\omega}| \leq \left(\frac{D_{C,\omega}}{m'}\right)^k r'$$

In order to deliver the flow to all its destination edge switches using no more than x core switches, Lemma 5.1 must hold. Equivalently, we must have

$$|M_{j_x}^{C,\omega}| \leq \left(\frac{D_{C,\omega}}{m'}\right)^x r' < 1$$

By solving the inequality, we obtain

$$m' > D_{C,\omega} r'^{1/x}$$

By Lemma 5.2, $D_{C,\omega} = \left\lfloor \frac{n-\omega}{C \cdot S + b - \omega} \right\rfloor$. The lemma is thus proved. $\square$

The property of the fat-tree network topology must be taken into account in order to find the congestion bound. Through examining the structure of the fat-tree network, we have the following important observation regarding the relationship between the number of core switches m and the congestion bound C in a fat-tree DCN $ftree(m, n, r)$.

Property 5.1. *Given the number of edge switches r, the number of servers connected to each edge switch n and a certain flow scheduling strategy, the congestion bound* monotonically decreases *with the increase of core switches m in a $ftree(m, n, r)$ DCN.*

To see why this property stands, suppose there are two fat-tree DCNs $ftree_1$ and $ftree_2$, both have exactly the same configuration except for that $ftree_1$ has more core switches provisioned. Suppose the servers in both DCNs generate an identical sequence of flow set-up requests. Given the same scheduling strategy, each flow in $ftree_1$ has the option to choose from a larger number of parallel paths between any pair of edge switches than that in $ftree_2$, hence, less flows would be routed through each link, which makes the congestion bound of $ftree_1$ strictly smaller than that of $ftree_2$.

The above property allows us to convert the problem of finding the congestion bound C in a given $ftree(m, n, r)$ DCN to a more tractable, equivalent problem, which is to find the minimum number of core switches m required to guarantee a given congestion bound C in a fat-tree DCN with r edge switches, each connecting n servers. By Lemma 5.2 and Lemma 5.3, we can find the minimum number of core switches required to guarantee the congestion bound of C in a $ftree(m, n, r)$ under the BCMS algorithm for any sequence of multicast flow requests that satisfy the hose model, as shown in the following theorem.

Theorem 5.1. *The BCMS algorithm can ensure the congestion bound of C in a $ftree(m, n, r)$ DCN for an arbitrary sequence of multicast flow requests satisfying* *hose model, if the number of core switches*

$$m > \min_{1 \le x < r} \left\{ \left\lfloor \frac{(n-1)x}{C \cdot S + b - 1} \right\rfloor + \left\lfloor \frac{(n-1)}{C \cdot S + b - 1} \right\rfloor (r-1)^{1/x} \right\} \tag{5.2}$$

Proof. For an incoming multicast flow (i, D, ω), Lemma 5.2 gives the maximum possible number of (C, ω)-unavailable core switches under the condition that every flow is routed through at most x core switches. Then, Lemma 5.3 reveals that the flow can always be delivered through at most x core switches, provided that there are more than $m' = D_{C,\omega} r'^{1/x}$ (C, ω)-available core switches. Therefore, combining these two lemmas, we can obtain the sufficient number of core switches needed to deliver (i, D, ω) while keeping the congestion of all network links bounded by C.

Since the fanout of a multicast flow can range from 1 to $r-1$, and the bandwidth demand ω is a discrete value in B, we need to ensure that the DCN is capable of accommodating any incoming flow types. Clearly, m' is maximum when the flow fanout is $r-1$. Also, according to the network model, n is much larger than S in fat-tree DCNs, and congestion bound C should be reasonably small to maintain good network performance, hence, $U_{C,\omega}$ is maximized when $\omega = 1$. Finally, we should find the optimum value for x, the maximum number of core switches that each flow will route through, such that the minimum number of core switches is used.

To sum it up, Inequality (5.2) gives the minimum sufficient number of core switches to ensure the congestion bound of C in a fat-tree DCN $ftree(m, n, r)$ under the hose traffic model when the BCMS algorithm is adopted. □

Note that a circuit switching fat-tree network can be viewed as a special instance of the network model adopted in this chapter, that is, if we set the values of b, S and C to 1, i.e., each link can carry at most one flow, then the fat-tree network model in this chapter would reduce to a simple circuit switching network. In this case, the condition in Theorem 5.1 degenerates to the condition for the nonblocking multicast Clos networks in [29].

We can also simplify the condition in Inequality (5.2), as shown in the following corollary.

Corollary 5.1. *The BCMS algorithm can ensure the congestion bound of C in a $ftree(m, n, r)$ DCN, if the number of core switches*

$$m > 3 \left\lceil \frac{(n-1)}{C \cdot S + b - 1} \right\rceil \frac{\log r}{\log \log r} \tag{5.3}$$

Proof. The condition in Inequality (5.2) can be relaxed to

$$m > \min_{1 \le x < r} \left\{ \left\lceil \frac{(n-1)}{C \cdot S + b - 1} \right\rceil (x + r^{1/x}) \right\} \tag{5.4}$$

For any constant $u > 0$, we let $x = u\frac{\log r}{\log \log r}$ in (5.4). Then,

$$r^{1/x} = r^{\frac{u \log \log r}{\log r}} = (\log r)^{\frac{1}{u}}$$

Letting $u = 2$, Inequality (5.4) can be written as

$$m > \left\lceil \frac{(n-1)}{C \cdot S + b - 1} \right\rceil \left\lceil 2\frac{\log r}{\log \log r} + (\log r)^{\frac{1}{2}} \right\rceil$$

Since $\frac{\log r}{\log \log r}$ is of higher order than $(\log r)^{\frac{1}{2}}$, we have that

$$m > 3 \left\lceil \frac{(n-1)}{C \cdot S + b - 1} \right\rceil \frac{\log r}{\log \log r}$$

core switches are sufficient to ensure the congestion bound of C in $ftree(m, n, r)$ under BCMS. □

Next, we show the above results can be easily applied back to fat-tree DCNs with only unicast traffic by considering a unicast flow as a special multicast flow with only one destination server. In this case, the BCMS algorithm would become a greedy scheduling strategy, in which each flow chooses the path that consists of a (C, ω)-available uplink and a (C, ω)-available downlink and has the maximum available bandwidth. We will state the corollary that gives the minimum number of core switches required to achieve the congestion bound of C in a unicast fat-tree DCN under the BCMS algorithm without a proof, as it can be similarly derived.

Corollary 5.2. *The BCMS algorithm can ensure the congestion bound of C in a*

$ftree(m, n, r)$ DCN with only unicast traffic, if the number of core switches

$$m > 2 \left\lfloor \frac{n-1}{C \cdot S + b - 1} \right\rfloor \tag{5.5}$$

Again, it is easy to see that the condition in Corollary 5.2 coincides with the strict-sense nonblocking condition for unicast Clos circuit switching networks [113], if the values of b, S and C are set to 1.

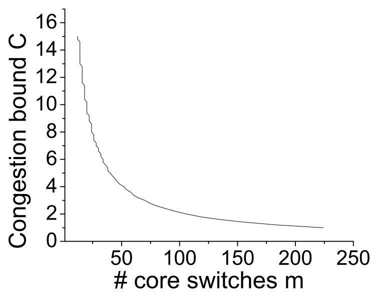
Theorem 5.1 and Corollary 5.2 give the minimum number of core switches m required to guarantee a certain congestion bound C in a $ftree(m, n, r)$ DCN under multicast and unicast traffic, respectively. Now, considering that C strictly decreases with the increase of number of core switches m, we can use the above results in the opposite way and determine the minimum congestion bound C in any given fat-tree DCN $ftree(m, n, r)$ under the BCMS algorithm. From Theorem 5.1, we can also see that the value of congestion bound C and that of x, the maximum number of core switches each flow routes though under the BCMS algorithm, are solely determined by the configuration of the fat-tree DCN $ftree(m, n, r)$, that is, the values of m, n and r, and the normalized core switch link bandwidth capacity S.

Finally, we analyze the time complexity of the BCMS algorithm, which is given by the following theorem.

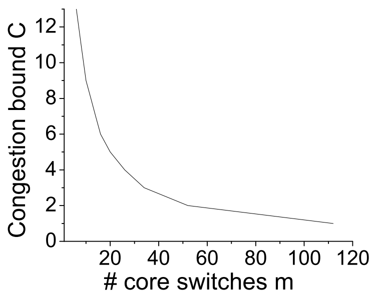
Theorem 5.2. *The BCMS algorithm finishes scheduling each multicast flow in $O(m \log r)$ time, where m is the number of core switches and r is the number of edge switches in a fat-tree DCN $ftree(m, n, r)$.*

Proof. In step 1 of BCMS, the algorithm checks the condition of every uplink out of the source edge switch of the incoming flow, which takes linear time to the total number of core switches m in a $ftree(m, n, r)$ DCN. The most involved part of BCMS is step 2, which determines the time complexity of the algorithm. In each iteration, the algorithm first finds the core switch(es) with the minimum cardinality inaccessible set, then breaks the tie by selecting the core switch that would deliver the maximum bandwidth to corresponding edge switches. Clearly, the procedure in each iteration also costs $O(m)$ time.

To determine how many iterations are needed to schedule a flow, recall that Lemma 5.3 proves that under the minimum cardinality strategy, any incoming flow



(a) Multicast



(b) Unicast

Figure 5.4: Relationship between congestion bound C and the number of core switches m in a $ftree(m, 512, 40)$ DCN with core switch link bandwidth $S = 10$ under: (a) multicast traffic; (b) unicast traffic.

(i, D, ω) will be routed through at most x core switches, with x being the optimum value that minimizes the condition in Theorem 5.1. To find the minimum value of the right hand side of Inequality (5.2), x should be chosen such that the two addends are equal. Hence, we can see that x is roughly logarithmically proportional to the number of edge switches r, i.e., $x = O(\log r)$. Since step 2 of BCMS will terminate in no more than x iterations, the worst-case time complexity for this step is $O(m \log r)$. In step 3, every destination edge switch in D chooses the core switch that would deliver maximum bandwidth from at most x core switches. As each edge switch in D can perform this procedure simultaneously in parallel, this step only costs $O(\log r)$ time. Overall, the time complexity of BCMS is $O(m \log r)$. □

Note that the complexity of the algorithm is polynomial to the number of core switches and edge switches (pods), which usually ranges from tens to hundreds even in a large fat-tree DCN. Therefore, BCMS is very time-efficient in computing flow schedules.

5.6.1 Discussions

For demonstration purpose, in Fig. 5.4, we show the relationship between congestion bound C under the BCMS algorithm and the number of core switches m in a medium size fat-tree DCN with $r = 40$ edge switches (pods), each of which connecting to $n = 512$ servers, obtained from Theorem 5.1.

We can draw several interesting observations from the above example. First, when the congestion bound $C = 1$, the total traffic load on each network link must be within the link capacity. In other words, the DCN is *nonblocking*, meaning that the DCN can be considered as a huge crossbar switch that allows all connected servers to communicate with each other without congestion. Not surprisingly, such ideal fat-tree DCNs incur prohibitively high cost, for example, 224 40×40 core switches with 10Gbps ports are required to build a nonblocking multicast fat-tree DCN of $r = 40$ edge switches with $n = 512$ servers in each edge switch. Even under an optimistic assumption that each core switch only costs $\$5k$, the expense for core switches alone would be millions of dollars.

Second, we can observe that the number of core switches required decreases drastically in an oversubscribed fat-tree DCN even when the congestion bound C is small, but such drop in the number of core switches slows down when C reaches a certain level. This observation suggests that data center providers should find the perfect balance between DCN cost and performance by introducing an appropriate amount of oversubscription.

Third, the above results, for the first time, give an accurate description of worst-case network congestion down to the link level in multicast fat-tree DCNs, under the condition that all flows are routed by a practical scheduling algorithm, which reveals important performance-cost trade-off of fat-tree DCNs under multicast traffic. These results can be used to facilitate the design of fat-tree DCNs. Building a DCN in practice is a complex optimization problem with numerous variables. Data center providers must decide the performance requirement of a DCN according to the Service Level Agreement (SLA) of various cloud services. Many implementation details such as wiring, cooling and power consumption, etc, also need to be considered. There have been several optimization models for DCN construction [13, 14]. Our results above can provide a theoretical guideline in estimating potential network congestion, thus can be integrated in these models when building cost-effective, high-performance fat-tree DCNs.

5.7 Performance Evaluations

We have developed an event-driven flow simulator to evaluate the performance of BCMS in fat-tree DCNs under different traffic patterns. In our simulation, we target a medium-size fat-tree DCN built by 32-port GigE commercial switches that

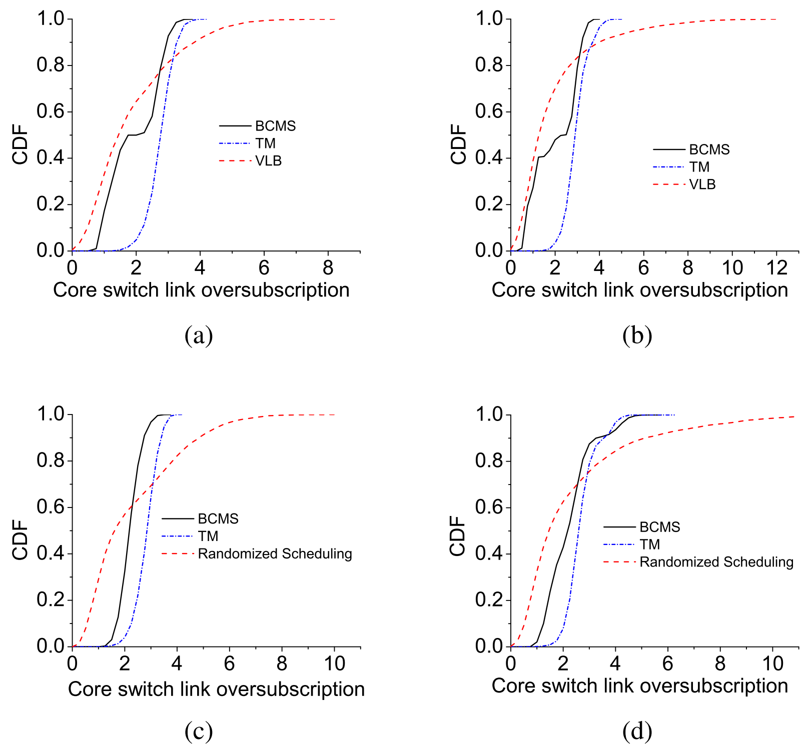


Figure 5.5: Cumulative density function (CDF) of core switch link congestion in a $ftree(64, 256, 32)$ DCN under: (a) Uniform unicast traffic; (b) Staggered unicast traffic; (c) Uniform mixed traffic; (d) Staggered mixed traffic.

can accommodate 8,912 host servers as in [18]. In the fat-tree DCN, there are 32 pods, each containing 16 Top-of-Rack (ToR) switches and 16 aggregation switches. Each ToR switch has half of the ports connecting servers and the remaining half of the ports connected to aggregation switches. There are 64 32-port core switches, each of which has one port connecting to each of the 32 pods. All the links in the DCN have 1Gb/s bandwidth. Clearly, this configuration corresponds to $ftree(64, 256, 32)$ DCN according to our network model.

Due to the fact that a myriad of packets are injected into the network every second in even a small cluster with hundreds of hosts [15], packet-level simulation is impractical for large scale data centers. Therefore, we have developed an event-driven flow simulator as in [15, 60]. Flow-level simulation, though cannot fully capture inter-flow dynamics or buffer behavior, offers a good approximation of packet-level performance of DCNs [60].

The simulator accepts as input a communication pattern among hosts and uses it, along with a specification of flow bandwidth demand and arrival rates, to generate simulated traffic. New flows arrive with an exponentially distributed length, and start times follow a Poisson arrival process with a given mean. The bandwidth demand of each incoming flow follows a uniform distribution between $(0, 1]$ Gb/s. The simulation proceeds in an event-driven manner, in which a priority queue is used to store different types of events, such as flow arrival and termination, in their temporal order. When triggered by the head event of the queue, the simulator will add or remove flows from involved links accordingly, then update the network condition and current time stamp. Currently, since no data center traffic traces are publicly available due to privacy and security concerns, we generate synthetic traffic to evaluate our algorithm as in [15, 60]. In order to create traffic patterns that can stress and saturate the core network, we assume there is no local traffic, that is, all flows are between different edge switches (pods). The description of each traffic pattern is elaborated as follows.

We first consider the case when only unicast traffic exists in the DCN. For unicast traffic, we create two traffic patterns similar to [15] according to the following style:

- Uniform unicast traffic: Each unicast flow (i, D, ω) is originated from some server connected to a randomly chosen edge switch i and destined for another edge switch chosen with uniform probability.

- Staggered unicast traffic: We first randomly pick H hotspot edge switches amongst all the edge switches. Each flow has α probability to be destined for one of these hotspot switches, and $1-\alpha$ probability to be destined for the rest of the network. In our simulation, we set $H = 5$ and $\alpha = 0.4$.

We also investigate the case when there is a mixture of unicast traffic and multicast traffic in the network. We notice that multicast flows with small fanout, such as file chunk replication, are much more than multicast flows with large fanout, such as map-reduce binary distribution in data centers. Therefore, we assume the fanout of multicast flows follows a power law distribution, more formally, the probability that a flow (i, D, ω) has a fanout of $|D|$ is $a \cdot b^{|D|}$. In our simulation, a is set to 1 and b is set to 0.5. Under this setting, half of the traffic consists of unicast flows, as each unicast flow can be seen as a special multicast flow with fanout 1. We also create two traffic patterns for such mixed traffic, shown as follows:

- Uniform mixed traffic: A flow (i, D, ω) orientates from a source edge switch i chosen uniformly at random amongst all edge switches $\{1, 2, \ldots, r\}$. The fanout $|D|$ is decided according to the power law probability distribution as shown above. Once the fanout $|D|$ is decided, the set of destination edge switches $D \subseteq \{1, 2, \ldots, r\}/i$ is chosen uniformly at random amongst all subsets of $\{1, 2, \ldots, r\}/i$ having cardinality of $|D|$.

- Staggered mixed traffic: We randomly pick H hotspot edge switches amongst all the edge switches. To generate a flow (i, D, ω), we first randomly choose the source edge switch i and decide the fanout $|D|$ according to the power law probability distribution. Each destination edge switch in D has α probability to be one of the hotspot switches, and $1 - \alpha$ probability to be in the rest of the network. In our simulation, we set $H = 5$ and $\alpha = 0.4$.

Initially, the mean flow duration is set to be much longer than the interval between flow arrivals in order to accumulate traffic load in the network. When the traffic load reaches a preset level, then the flow arrival rate and duration are adjusted such that the network is stably loaded at the preset level. In order to ensure the generated traffic patterns satisfy the hose traffic model, a flow arrival is placed in the event queue only if the total flow bandwidth demand at the corresponding source server and destination servers would not surpass the link capacity after the flow is added. Since we would like to evaluate the performance of BCMS under congested traffic conditions, the network is stably loaded at the level that 90% of the ingress link capacity of each server is occupied by existing flows on average in this simulation.

For comparison purpose, we implement Valiant Load Balancing (VLB) [17] for unicast flow scheduling, in which each flow is routed towards its destination server via a randomly chosen core switch. Due to the lack of previous multicast scheduling algorithms for fat-tree DCNs, we also adopt a randomized scheduling algorithm for multicast flows as benchmark, described as follows. Suppose there is an incoming multicast flow (i, D, ω), the randomized scheduling algorithm randomly chooses x core switches, with x being an integer uniformly chosen from $(1, \min(r, |D|)]$.

Then we divide the destination edge switches in D into x non-empty and non-overlapping subsets, and each of the x core switches forwards the flow to one of the subsets. We can see that the randomized scheduling is identical to VLB when scheduling unicast flows. VLB and randomized scheduling algorithms are non-adaptive algorithms, because they are oblivious to the current network condition when scheduling incoming flows.

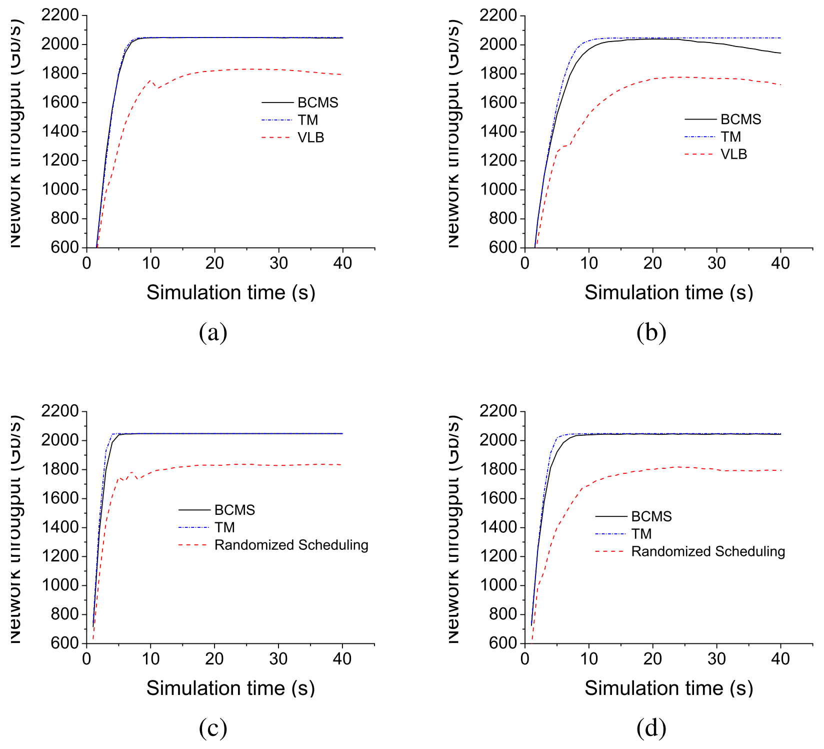


Figure 5.6: Total network throughput vs. time for a $ftree(64, 256, 32)$ DCN under. (a) Uniform unicast traffic; (b) Staggered unicast traffic; (c) Uniform mixed traffic; (d) Staggered mixed traffic.

As mentioned earlier, the multicast scheduling problem can be modeled as the problem of finding a Steiner tree in the network, which is NP-hard. In this simulation, we also compare the performance of BCMS with the TM algorithm [35], which is a well-known polynomial time heuristic algorithm for solving the Steiner tree problem in general graphs. However, it should be noted that the multicast scheduling problem studied in this chapter is different from the Steiner tree problem in that the destination edge switches of a multicast flow can only be the leaves of the spanning tree, because they are not allowed to relay data to other edge switches in fat-tree DCNs. We impose the above constraint in the implementation of the TM algorithm, which proceeds as follows. First, it finds the destination edge switch with the minimum-cost path to the source, in which the cost of a link l is set to e^c, where c is the current congestion of link l as in [37, 38]. It connects the source and the destination and sets them as the selected tree, then extends a branch to the destination edge switch with the minimum-cost path to the selected tree. The T-M algorithm repeats these extension steps until all destinations are covered, then outputs the selected tree. Since it takes $O(m)$ time to find the minimum-cost path from the selected tree to a destination edge switch in a $ftree(m, n, r)$ DCN, the TM algorithm has $O(mr)$ time complexity, higher than the complexity of BCMS, which is $O(m \log r)$. Clearly, the TM algorithm is also adaptive, as it chooses the route of each flow according to the current link congestion.

First, we measure the cumulative density function (CDF) of the congestion of all core switch links. In order to obtain accurate results, all statistics are collected when the network is stable and we run the simulation for 10 times to obtain the average CDF of core switch link congestion, as shown in Fig. 5.5. From the figure, we can see that the CDF of core switch link congestion spreads across a wide range under VLB and randomized scheduling, meaning that these scheduling algorithms cannot prevent heavy congested links even though there are still idle core switch links in the network. Therefore, the network suffers from unbalanced traffic load distribution under non-adaptive flow scheduling algorithms like VLB. In comparison, the CDF under BCMS rises sharply from 0 to 1 over a very narrow range of core switch link congestion for all tested traffic patterns, which confirms that the traffic load distribution is much more balanced amongst core switch links, as every core switch link is similarly oversubscribed. We also find that under VLB and randomized scheduling, some core switch links suffer from more severe congestion under staggered traffic patterns where the destination edge switches of flows are non-uniformly distributed. The reason is that since there are more flows concentrating on hotspot edge switches than other switches, it is more likely that multiple "elephant" flows are placed on core switch links towards these hotspot edge switches under non-adaptive scheduling algorithms like VLB and randomized scheduling, causing heavy congestion on these links.

BCMS also achieves better performance than the TM algorithm under all tested traffic patterns, as shown in Fig. 5.5. The advantage of BCMS over TM is more evident when multicast traffic exists in the network, because the minimum cardinality strategy adopted in BCMS can significantly reduce duplicated traffic of a multicast flow through effectively utilizing the multicast capability of core switches. When there is only unicast traffic, the congestion of core switch links under BCMS is slightly lower in general than that under the TM algorithm, because BCMS finds a path that maximizes received bandwidth for each unicast flow, which leads to better traffic load balance than the minimum-cost path approach adopted in TM. Moreover, there is no upper bound on the link congestion in TM, which means that it is possible for some flows to go through heavily congested links under malicious traffic patterns. In comparison, the link congestion under BCMS is bounded under arbitrary traffic patterns as shown earlier. This feature gives BCMS the capability to always maintain a benevolent network environment. Since heavily congested links cause many detrimental consequences such as unpredictable network performance and unfair bandwidth sharing as mentioned earlier, we can see that BCMS holds significant advantages over all compared algorithms in terms of network predictability and fairness.

Next, we consider *network throughput*, defined as the total bandwidth delivered to destination servers by all the existing flows in the network. As mentioned earlier, we assume proportional fairness when calculating flow bandwidth, i.e., each contending flow on an oversubscribed link gets the bandwidth proportional to its demand. Fig. 5.6 shows the variation over time of the total network throughput for $ftree(64, 256, 32)$ DCN. We can see that at the beginning of the simulation, the network is populated with an increasing number of flows till traffic load reaches the preset level. Adaptive algorithms like TM and BCMS achieves higher network throughput than non-adaptive algorithms such as VLB during this period, because BCMS and TM can make discrete flow placements to increase network throughput when the traffic is not very congested.

When the network reaches a stable state, both TM and BCMS achieve consistent throughput close to the network bisection bandwidth (i.e., 2048Gb/s), which is much higher than non-adaptive algorithms, under all tested traffic patterns. The reason is that under congested traffic patterns as used in our simulation, any residue bandwidth capacity on idle core switch links is wasted, since it could have been used to transmit flows on other heavily congested links if given an efficient flow scheduling algorithm. As can be seen in Fig. 5.5, a considerable number of core switch links are under-subscribed (i.e., the congestion is smaller than one) under VLB and randomized scheduling, while almost all core switch links are reasonably oversubscribed under BCMS and TM.

Even though both BCMS and TM can achieve throughput close to network bisection bandwidth under very congested traffic patterns, it is worth pointing out that BCMS allocates bandwidth to individual flows more fairly than TM, because as shown earlier, it achieves better load balance than TM. Also, each multicast flow gains more bandwidth under BCMS than TM on average, given that the minimum cardinality strategy adopted in BCMS significantly reduces bandwidth consumption caused by duplicated traffic in multicast flows.

Since the centralized scheduler has to accommodate a large number of flow set-up requests per second in a data center, the runtime of the scheduling algorithm must be reasonably short. Here, we also study the runtime of BCMS. We use a modest quad-core 2.2 GHZ machine in our simulation. For a medium-size fat-tree DCNs with 8192 hosts, the maximum runtime of scheduling a multicast flow over a sufficiently long simulation period is only 9 ms, and the average runtime is approximately 3 ms, which is very short. We expect the scheduler in practical data centers to be a fairly high performance machine, thereby keeping the average runtime well under 1 ms.

One problem with centralized scheduling is that it may be difficult to scale in a large DCN, even though BCMS is very time-efficient. For example, it has been found that the number of flows arriving every second on a small 1500-server cluster is about 100K, which is considerably larger than the volume that a central scheduler can handle [17]. In addition, the sub-10ms flow install overhead may be unacceptable for delay-sensitive traffic.

On the other hand, as mentioned earlier, DCN traffic can be roughly considered as a mixture of a small number of throughput-sensitive “elephant” flows and numerous delay-sensitive “ant” flows. To solve the scalability problem, we can let the central BCMS flow scheduler handle only “elephant” flows, and apply OpenFlow flow-match wildcards or hash-based routing for “ant” flows at local edge switches, as in Hedera [15] and DevoFlow [17]. Since these “ant” flows usually have very small bandwidth demand, thus minimum impact on the congestion of core switch links, the BCMS algorithm can avoid heavy link congestion though it is not directly involved in the scheduling of delay-sensitive “ant” flows. Furthermore, BCMS can greatly benefit “ant” flows by creating a benevolent network environment that guarantees low-latency communication.

5.8 Conclusions

In this chapter, we present a low-complexity on-line multicast scheduling algorithm called BCMS for fat-tree DCNs, which leverages centralized control and global knowledge of network condition of OpenFlow framework for data centers to achieve efficient bandwidth utilization. We show that BCMS can guarantee bounded congestion in a fat-tree DCN for an arbitrary sequence of multicast flow requests under hose traffic model. We also derive the congestion bound for a given fat-tree DCN under BCMS. Our results, for the first time, give an accurate description on the worst-case network congestion down to link level in fat-tree DCNs under multicast traffic. Finally, we compare the performance of BCMS with several non-adaptive scheduling algorithms and a well-known adaptive multicast scheduling algorithm called the TM algorithm through simulations. The simulation results demonstrate that BCMS achieves significantly better traffic load balance and higher network throughput than non-adaptive algorithms. It also outperforms the TM algorithm in terms of traffic load balance and time complexity.

Chapter 6

Collaborative Network Configuration in Hybrid Electrical/optical Data Center Networks

Recently, there has been much effort on introducing optical fiber communication to data center networks (DCNs) because of its significant advantage in bandwidth capacity and power efficiency. However, due to limitations of optical switching technologies, optical networking alone has not yet been able to accommodate the volatile data center traffic. As a result, hybrid packet/circuit (Hypac) switched D-CNs, which argument the electrical packet switched (EPS) network with an optical circuit switched (OCS) network, have been proposed to combine the strengths of both types of networks. However, one problem with current Hypac DCNs is that the EPS network is shared in a best-effort fashion and is largely oblivious to the accompanying OCS network, which results in severe drawbacks, such as degraded network predictability and deficiency in handling correlated traffic. Since the OCS/EPS networks have unique strengths and weaknesses, and are best suited for different traffic patterns, coordinating and collaborating the configuration of both networks is critical to reach the full potential of Hypac DCNs, which motivates the study in this chapter. First, we present a network model that accurately abstracts the essential characteristics of the EPS/OCS networks. Second, considering the recent advances in network control technology, we propose a time-efficient algorithm called Collaborative Bandwidth Allocation (CBA) that configures both networks in a complementary manner. Third, we show, for the first time, that given sufficient bandwidth from both networks, a Hypac DCN can *guarantee* 100% throughput with a bounded delay. Finally, we conduct comprehensive simulations, which demonstrate that CBA significantly improves the performance of Hypac DCNs in many aspects.

The rest of the chapter is divided into six sections. Section 6.1 gives the introduction of our work. Section 6.2 introduces some related work. Section 6.3 discusses the challenges faced by existing Hypac DCNs. Section 6.4 gives the network model and the formulation of the collaborative network configuration problem. Section 6.5 presents the proposed CBA algorithm. Section 6.6 gives the sufficient network condition for a Hypac DCN to guarantee 100% throughput with a bounded delay. Section 6.7 presents the performance evaluation results, and finally Section 6.8 concludes the chapter.

6.1 Introduction

It is very challenging for electrical DCNs to accommodate the fast increasing bandwidth demand in today's data centers without imposing exorbitant hardware cost and power consumption. Meanwhile, the significant advantage of optical fiber communication in power efficiency and huge bandwidth capacity has led to several proposals of all-optical DCNs recently [39]. However, due to the limitations of optical switching technologies, optical networking alone cannot effectively handle the volatile traffic in data centers. For example, optical circuit switching (OCS) and optical burst switching require connections to be established before communication, which are not suitable for latency-sensitive traffic. Optical packet switching enables packet granularity transmission as electrical networks. However, its implementation is hindered by the lack of practical optical buffers at current stage.

These factors have motivated the emergence of hybrid electrical/optical (referred to as Hypac for short) DCN architectures [40, 41]. As shown in Fig. 6.1, a Hypac DCN augments the electrical packet switched (EPS) network with a rack-to-rack optical circuit switched (OCS) network. The goal is to combine the best features of both networks: the EPS network can provide flexible and fine-grained

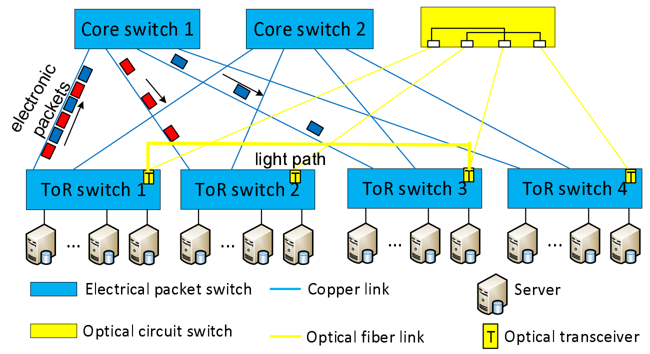


Figure 6.1: A general Hypac DCN architecture, in which the EPS network is augmented with a rack-to-rack OCS network.

transmission, which enables each server to communicate with multiple other servers simultaneously; The OCS network, on the other hand, can deliver high bandwidth transmission with lower power consumption between any pair of server racks with established connections. Hence, Hypac DCNs are able to maintain the flexibility of packet switching, while allocating *on-demand* large bandwidth to the places where it is most needed by dynamically establishing OCS connections. It has been demonstrated [40, 41] that the introduction of optics results in orders of magnitude less power consumption and hardware cost than pure electrical DCN architectures with comparable performance.

However, one big problem with current Hypac DCNs is that the EPS network is still shared in a best-effort way, and is completely oblivious to the accompanying OCS network. As will be commented in detail later, the lack of coordination between the two networks causes severe performance deficiencies in current Hypac DCNs, including incompetence in handling correlated traffic and degraded network predictability [42]. Since the EPS network and the OCS network have unique strengths and weaknesses, they are best suited for transmitting different traffic patterns. The coordination and collaboration between the two types of networks is essential in order to exploit the full potential of Hypac DCNs. In the meantime, emerging network control technologies, such as Software Defined Networking (SDN) [43], provide many useful features, such as guaranteed bandwidth provisioning between host servers. These features can be utilized to achieve more effective allocation of the EPS network bandwidth than best-effort sharing. Leveraging the new opportunities enabled by novel traffic control technologies, we study the collaborative network configuration problem in Hypac DCNs in this chapter.

The challenge of this problem lies in that the two networks have fundamentally different characteristics. To tackle the challenge, we first develop a network model that accurately captures the essential characteristics of the EPS/OCS networks. Then we investigate the key factors that lead to bandwidth loss in the OCS network, and demonstrate that the EPS network can be utilized to significantly improve the optical bandwidth utilization. Based on these findings, we develop a time-efficient Collaborative Bandwidth Allocation (CBA) algorithm that effectively combines the strengths of both networks.

Ideally speaking, a DCN should provide similar performance to a congestion-free (or nonblocking) network that guarantees 100% throughput and no queueing delay. Though previous work has demonstrated through experiments and simulations that a Hypac DCN can deliver near 100% throughput and low latency under tested traffic patterns [40], no conclusive analysis is given on whether such performance is guaranteed under *all* traffic patterns. In this chapter, we prove that it is indeed feasible for a Hypac DCN to emulate a *constrained* nonblocking network, which guarantees 100% throughput under all possible traffic patterns. By constrained, we mean a bounded delay is imposed to compensate for the scheduling overhead of the OCS network. We also give the sufficient network condition to achieve such emulation under the proposed CBA algorithm. Finally, we evaluate CBA through simulations and show that it significantly improves network performance in many aspects compared to current Hypac DCNs.

6.2 Related Work

Due to the limitation of optical switching technology, Hybrid packet/circuit (Hypac) switched DCNs have been proposed to combine the best features of optics and electronics. C-through [40] arguments the electrical packet switched DCN with a rack-to-rack MEMS optical switch. Helios [41] shares a similar architecture to c-through, but uses WDM links to deliver higher transmission bandwidth. In both Hypac DCNs, the EPS network is used in a best-effort way and is unaware of the added OCS network.

Meanwhile, novel network control frameworks have been developed for DCNs, which enable more effective control of the EPS network than the traditional best-effort bandwidth sharing. For example, software defined networking (SDN), represented by the OpenFlow framework [43], provides many useful features, such as centralized control, global knowledge of network condition, bandwidth isolation and so on. Seawall [52] is a bandwidth allocation framework that can distribute guaranteed bandwidth to categorized traffic based on administrator-specified policies.

To effectively utilize the OCS network bandwidth, our proposed CBA algorithm finds multiple configurations per scheduling period using a matrix decomposition method, which has been adopted in high-speed switches [114–117]. However, the problem we study is fundamentally different from the previous work, because the key questions to be considered, such as how to find a proper traffic partition for the EPS/OCS networks and how to coordinate the configuration of two fundamentally different networks, are unique to Hypac DCNs and have not been studied before.

6.3 The Hypac DCN

In this section, we describe the general architecture and discuss the challenges in current Hypac DCNs.

6.3.1 A General Hypac DCN Architecture

Fig. 6.1 shows a general Hypac DCN architecture, in which an electrical packet switched (EPS) network is augmented with an optical circuit switched (OCS) network. The EPS network in existing Hypac DCNs adopts a tier-2 fat-tree network due to its good scalability and simple structure [40, 41]. As shown in Fig. 6.1, a fat-tree network places multiple core switches at the root, which provides several parallel paths between any pair of ToR switches. Sometimes a multi-tier fat-tree network is used if the data center size is large, which can be recursively built from basic 2-tier fat-trees [18]. In this chapter, we focus on a tier-2 fat-tree EPS network as in c-through [40] and Helios [41] .

In current Hypac DCN design, the OCS network is implemented using a large MEMS optical switch. Commercial MEMS switches with hundreds of ports and research prototypes with more than 1000 ports are currently available, which are adequate for most data centers today. In such switches, each switch port connects to a ToR switch to transmit the aggregate traffic of all the associated servers, instead of directly connecting to each individual server. MEMS optical switches need to mechanically adjust micro mirrors to reconfigure connections, which incurs a non-negligible *switching overhead* (several milliseconds). The OCS network is configured in a periodical manner through a central controller, which needs to acquire up-to-date traffic demand information from all the server racks for calculating the new configuration. This also introduces a considerable *control overhead*. Therefore, the scheduling period must be sufficiently long to properly amortize the control overhead (e.g., more than 1 seconds in c-through[40]).

One important motivating factor of Hypac DCNs is that data center traffic commonly consists of a small number of long-lived, throughput-intensive "elephant" flows and numerous latency-sensitive "ant" flows [8, 9]. Hypac DCNs are particularly effective in handling such traffic: the EPS network can be used to transmit the volatile portion of traffic at packet-granularity, while the slow switching OCS network allocates high-bandwidth light paths to transmit the stable portion of traffic. In this way, Hypac DCNs maintains the flexibility that allows servers to communicate freely at any moment, while deliver cost-efficient large bandwidth via dynamically established light paths. Moreover, since the EPS network is alleviated from the duty of providing high bandwidth, it can be oversubscribed to achieve significant cost reduction.

6.3.2 Challenges in Current Hypac DCNs

In current Hypac DCNs [40, 41], the network controller configures the OCS network by finding a *maximum-weighted matching* (MWM) on a bipartite graph of ToR switches in each scheduling period, with edge weights denoted by the corresponding inter-rack traffic demand. The EPS network, on the other hand, is unmodified from traditional electrical DCNs and completely oblivious to the OCS network. Such an approach causes severe performance deficiencies [42], as discussed below.

First, the MWM-based configuration generates only one OCS configuration per scheduling cycle, which remains unchanged throughout the entire scheduling period. However, Since each scheduling period is usually quite long given the considerable reconfiguration overhead, there has to be a large amount of traffic accumulation between connected ToRs to maintain high optical bandwidth utilization. For example, the typical scheduling period in c-through [40] is 1s. Suppose the optical link bandwidth is 10Gb/s, each connected pair of ToRs needs to accumulate 10Gb traffic in order to fully utilize the optical link, which is difficult to achieve in practice. As a result, optical bandwidth is significantly under-utilized under light traffic load.

Additionally, as it would generally take a long time for two ToR switches to accumulate enough traffic to be connected again after their traffic is depleted from the previous connection, servers will experience high delay variability and unpredictable network performance due to the sporadic optical bandwidth, which is a major obstacle to the cloud adoption for many reliability-driven applications.

More importantly, it has been shown that current Hypac DCNs are inefficient in handling correlated traffic [42], which is generated by many common distributed services in data centers, such as MapReduce, where the completion time depends on the slowest flow. The reason is that the OCS network prioritizes transmission between hosts with high traffic demand at the cost of stranding the rest [42]. In this chapter, we show that these challenges can be effectively addressed by adopting more effective OCS network configuration algorithm and collaborating the configuration of both networks.

6.4 Network Model and Problem Formulation

In this section, we present a network model for Hypac DCNs that accurately describes the essential characteristics of the EPS/OCS networks. We also describe the network control plane and formulate the collaborative network configuration problem.

Before introducing the model, we make a few reasonable assumptions. Since each ToR switch can forward packets to any of the associated servers, we consider a ToR switch as a single entity rather than individual servers connected to the ToR. We consider a Hypac DCN with N ToR switches, and assume all ToR switches share homogeneous setting, that is, each ToR switch connects the same number of servers and has the same port speed. We consider a single-wavelength OCS network as in c-through [40] and leave the multi-wavelength case for future study.

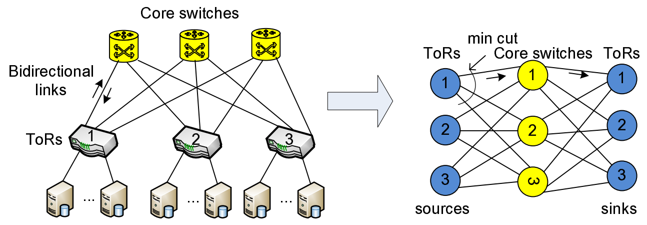


Figure 6.2: A fat-tree EPS network can be modeled as a flow graph, in which each ToR adds a source node and a sink node and each bidirectional link is unfolded into two unidirectional links.

6.4.1 Modeling the OCS network

One key characteristic of the OCS network, such as the MEMS switch, is that it is connection-oriented, which means that each switch port can be connected to at most one other port at the same time. Hence, we can represent a feasible OCS network configuration using a binary *configuration matrix* $\boldsymbol{O} = \{o_{ij}\}$, in which the entries in each line (row or column) sum to at most 1 and $o_{ij} = 1$ means that there is a connection from ToR i to ToR j. Another important feature of the OCS network is that it imposes a certain amount of switching overhead, which is a period of network "downtime" during switching state changes, denoted by σ. Finally, for simple illustration, we set the total bandwidth capacity of all the servers associated with a ToR switch to 1, and denote the normalized optical link bandwidth capacity by S. For example, if there are 40 servers connected to a ToR switch via 1Gb ports, and the OCS network connects to each ToR via a 100Gb port, we say the normalized optical link bandwidth is 2.5.

6.4.2 Modeling the EPS network

Different from the OCS network, the EPS network enables a ToR to communicate with multiple other ToRs simultaneously via different paths, and each link can be shared by packets from different sources through multiplexing. The traffic from ToR i to ToR j is said to belong to the same *flow* f_{ij}. One characteristic of the EPS network is that the bandwidth each flow receives can vary over time depending on the overall network condition. The instantaneous bandwidth received by all the flows in the EPS network can be represented as a matrix $\boldsymbol{B(t)}$, in which element $b_{ij}(t)$ represents the amount of bandwidth received by flow f_ij at time t. For simplicity, bandwidth here is also normalized as in the OCS network.

Apparently, the total bandwidth allocated to all the flows must be constrained by the network capacity. We can construct an auxiliary flow graph G with each vertex representing a switch and each edge representing a link in the EPS network. For each ToR, we add a source vertex and a sink vertex to G. The bandwidth received by flow f_{ij} is the value of a flow between the corresponding source and sink vertex. For example, the flow graph of a fat-tree DCN is shown in Fig. 6.2, in which each bidirectional link is unfolded into two unidirectional links.

According to the *max-flow min-cut theorem*, the flow bandwidth matrix must satisfy the following constraint

$$\sum_{i \in X} \sum_{j \in Y} b_{ij}(t) \leq CUT_{X,Y}, \forall X \subset Z, Y = Z - X,$$

where Z is the set of all ToRs and $CUT_{X,Y}$ is the cut with minimum bandwidth capacity in the network between X and Y. This constraint indicates that the maximum bandwidth of flows sent from any set of ToRs to the rest of the network is equal to the the minimum cut bandwidth capacity allowed by the network.

The constraint can be used to determine the feasible region of flow bandwidth matrices for general EPS networks. In this chapter, we focus on a fat-tree EPS network, whose simple structure makes it relatively easy to define the feasible region of $\boldsymbol{B(t)}$. It is not difficult to see that all the core switch links connecting a ToR switch constitute the minimum cut for flows entering and leaving the ToR, as shown in Fig. 6.2. According to the constraint above, the maximum bandwidth of flows entering/leaving a ToR must be no larger than C at any moment, where C is the *normalized* total bandwidth of all the core switch links connecting each ToR. It is worth mentioning that the value of C is the reciprocal of the *oversubscription ratio* of the fat-tree network, an important metric for evaluating the network performance, which is defined as the ratio of worst-case aggregate bandwidth demand of servers to the network bisection bandwidth [18]. Since it is very costly to provision large bandwidth in the EPS network, fat-tree DCNs are often heavily oversubscribed (e.g., higher than 8:1 oversubscription ratio is common) [18], hence, C is usually much smaller than 1.

In the network model, we assume any flow bandwidth matrix $\boldsymbol{B(t)} = \{b_{ij}(t)\}$ that has the maximum line sum, i.e., the sum of a row or column, no larger than C

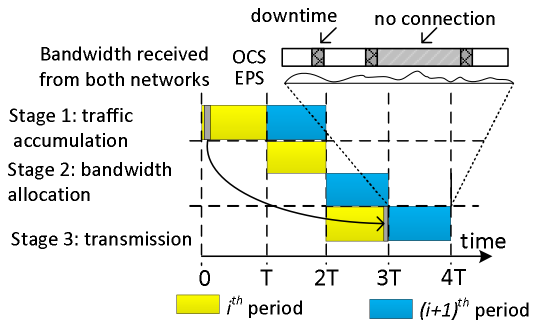

downtime
no connection
Bandwidth received from both networks
OCS
EPS
Stage 1: traffic accumulation
Stage 2: bandwidth allocation
Stage 3: transmission
time
0
T
2T
3T
4T
i^th period
(i+1)^th period

Figure 6.3: Each scheduling period is divided into three stages: traffic accumulation, configuration (assume $H = T$) and transmission, which can be pipelined between consecutive scheduling periods. A packet is marked in grey with an arc connecting its arrival and transmission time.

can be realized by the EPS network. This assumption is a close approximation to that in the real world, because near 100% link bandwidth utilization in fat-tree D-CNs [15] and flow-level bandwidth allocation can be achieved using novel network control frameworks [52].

6.4.3 Network Control

Finally, we describe the network control plane of Hypac DCNs, in which a centralized controller periodically collects traffic information and calculates the OCS network configuration. In general, a scheduling period can be divided into three sequential stages: traffic accumulation, configuration and transmission.

The first stage is traffic accumulation. Due to the volatility of data center traffic, we consider all traffic patterns that satisfy the *hose constraint* are possible. The hose constraint sets a very general "non-overbooking" rule that the traffic volume entering (leaving) a network node within unit time is bounded by its access bandwidth capacity, e.g., a server connected by 1Gb/s link can inject/receive at most 1Gb traffic into/from the network in unit time, which has been commonly used for describing data center traffic in DCN design and optimization [13]. The duration of the traffic accumulation stage is set to T, which is largely determined by the control overhead incurred by each configuration. As will be discussed later, T has significant impact on the network performance.

In the second stage, the network controller gathers information on the current

Table 6.1: Important Notations Used in the CBA Algorithm

N	Number of ToR switches
T	Duration of accumulation stage and transmission stage
$\boldsymbol{D}$	Traffic demand matrix at present, d_{ij} is traffic volume from ToR i to ToR j
K	Number of OCS network configurations in a schedule
$\boldsymbol{O_k}$	k^{th} OCS network configuration, $o_{ij}^k = 1$ denotes a connection from ToR i to ToR j
$\boldsymbol{X_k}$	Transmission matrix for $\boldsymbol{O_k}$ denoting actual traffic volume scheduled for transmission
S	Normalized bandwidth capacity of an optical link
ϕ_k	Weight of $\boldsymbol{O_k}$
σ	Switching overhead
$\boldsymbol{B(t)}$	Flow bandwidth distribution in the EPS network at time t
$\boldsymbol{E}$	Traffic allocation matrix of EPS network
C	Total normalized bandwidth capacity of core switch links connecting each ToR

inter-rack traffic demand, which can be done by sending queries to each ToR [41]. The inter-rack traffic demand is denoted by a *traffic demand matrix* $\boldsymbol{D} = \{d_{ij}\}$, whose entry d_{ij} is the accumulated traffic volume from ToR i to ToR j. For simplicity, entries are represented in terms of the time needed to transmit the traffic under 1 unit of normalized bandwidth. For example, suppose ToR i has $1Gb$ data to send to ToR j, and 1 unit of normalized bandwidth is $10Gb/s$, then $d_{ij} = 0.1$s. Using the gathered traffic information, the controller then calculates the proper configuration for the OCS/EPS networks. The duration of this stage is denoted by H, as shown in Fig. 6.3. Depending on the data center scale, algorithm complexity and controller hardware, H could be longer than T. In this case, $\lceil H/T \rceil$ controllers are required for pipelined processing of multiple batches.

In the third stage, traffic is transmitted according to results calculated in stage 2, whose duration is also T. Any remaining traffic that cannot be transmitted rolls over and adds to the traffic demand matrix of a future scheduling period. Also, stages in consecutive periods are pipelined, as shown in Fig. 6.3. Additionally, Fig. 6.3 shows a graphic representation of the bandwidth received by a flow from both networks: the bandwidth from the OCS network follows an on-off pattern with a certain period of downtime associated with each reconfiguration, whereas bandwidth from the EPS network is continuous and varying over time.

6.4.4 Problem Formulation

Next, we formulate the the collaborative network configuration problem. For clarity, a summary of useful notations is presented in Table 6.1.

Different from previous work, collaborative network configuration requires the controller to not only calculate the OCS network configuration(s), but also determine the traffic volume to be transmitted via the EPS network. We use a *traffic allocation matrix* $\boldsymbol{E} = \{e_{ij}\}$ to denote the traffic volume scheduled to be sent via the EPS network in the transmission stage, whose entry $e_{ij} = \int_t^{t+T} b_{ij}(t)dt$ is the traffic volume sent from ToR i to ToR j assuming the transmission stage starts at time t. Since the line sum of $\boldsymbol{B(t)}$ is no larger than C as specified in the network model, $\boldsymbol{E}$ is feasible if and only if its line sum is bounded by CT. Once $\boldsymbol{E}$ is determined, it can be fed to the aforementioned control frameworks like Seawall [52] to allocate bandwidth to flows during the transmission stage.

To avoid the optical bandwidth under-utilization caused by the maximum weighted matching based configuration adopted in current Hypac DCNs [40, 41], we use a matrix decomposition method to find a set of K weighted configurations $\boldsymbol{O_k}$ ($k \in \{1, 2, \ldots, K\}$) for the OCS network during each scheduling period. Let the weight ϕ_k of $\boldsymbol{O_k}$ denote the traffic volume a connection is able to transmit over the duration $\frac{\phi_k}{S}$, then the total traffic volume to be sent via the OCS network through these configurations can be written as $\sum_{k=1}^{K} \phi_k O_k$.

In order to transmit the accumulated traffic completely, the sum of traffic scheduled to be transmitted via the EPS network and the OCS network must be no less the corresponding entries in the traffic demand matrix $\boldsymbol{D}$, that is,

$$\sum_{k=1}^{K} \phi_k o_{ij}^k + e_{ij} \geq d_{ij}, \forall i, j \in \{1, 2, \ldots, N\} \tag{6.1}$$

The overall transmission time required for the K weighted OCS network configurations $\boldsymbol{O_k}$ ($k \in \{1, 2, \ldots, K\}$ is the total holding time of these configurations plus the total switching overhead, that is, $\frac{1}{S}\sum_{k=1}^{K} \phi_k + \sigma K$. The objective of the collaborative network configuration problem is to find $\boldsymbol{E}$ and a set of weighted configurations $\boldsymbol{O_k}$, such that the overall transmission time required is minimized. There are several reasons for choosing such an objective. On one hand, minimizing transmission time is important to ensure traffic is transmitted with low latency. On the other hand, if the transmission time is larger than T, a portion of traffic will roll over to the next scheduling period. Minimizing the total transmission time means that the amount of traffic left is also minimized, which reduces the possibility of future network congestion.

6.5 The Collaborative Bandwidth Allocation (CBA) Algorithm

In this section, we investigate the key factors that cause bandwidth loss in the OC-S network and how the EPS network can be best used to reduce optical bandwidth loss. Based on the findings, we then present an efficient algorithm called Collaborative Bandwidth Allocation(CBA), which leverages the flexible bandwidth allocation of the EPS network to significantly improve optical bandwidth utilization.

6.5.1 Overview

We first give a general description of the proposed CBA algorithm.

Given a traffic demand matrix $\boldsymbol{D}$, CBA operates in two phases: the OCS network configuration phase and the EPS network bandwidth allocation phase. In the OCS network configuration phase, CBA uses a matrix decomposition method to find a set of weighted OCS network configurations to accommodate the traffic demand matrix $\boldsymbol{D}$, with the objective that the total transmission time required is as short as possible. There are two important aspects that need to be considered. On one hand, it is important to make sure all connected ToR pairs have enough traffic to maintain high bandwidth utilization during each configuration. As will be explained later, having a larger number of configurations makes it easier to achieve such a goal, which leads to higher optical bandwidth utilization. On the other hand, reconfiguring the OCS network imposes switching overhead, which could considerably increase the total transmission time if the number of configurations is large. Hence, the trade-off between these two aspects must be carefully considered when configuring the OCS network.

The CBA algorithm manages to find the optimal balance between switching overhead and bandwidth utilization by adaptively setting the maximum number of configurations according to the parameters of the specific OCS network (e.g., network size, switch overhead, scheduling period, etc). It then uses an efficient "filtering" mechanism to ensure every connected ToR pair has sufficient amount of accumulated traffic in each configuration.

The second phase utilizes the flexible bandwidth allocation of the EPS network to minimize the total transmission time of the configurations obtained in the previous phase. First, CBA identifies the part of traffic that cause bandwidth loss in each OCS network configuration, then *shifts* it to the EPS network. In order to reduce switching overhead, which is a non-negligible part of the overall transmission time, CBA also tries to remove configurations with little traffic by shifting their traffic completely to the EPS network. We show that this problem can be modeled as a Multidimensional, Multiple-choice Knapsack Problem (MMKP) [118], which is NP-hard, and propose an efficient heuristic algorithm to find a good solution. Next, we describe the CBA algorithm in detail.

6.5.2 The OCS Network Configuration

In this phase, CBA decomposes the traffic demand matrix $\boldsymbol{D}$ to obtain a set of weighted OCS network configurations. Before explaining the detail of the algorithm, we use a simple example to illustrate some important issues in the algorithm design. As mentioned before, it is likely that the accumulated traffic between a pair of connected ToRs is insufficient to maintain full bandwidth utilization during the holding time of a configuration. We use a *transmission matrix* $\boldsymbol{X_k}$ to denote the *actual traffic volume* to be sent by an OCS network configuration $\boldsymbol{O_k}$. As shown in Fig. 6.4 , the traffic demand matrix $\boldsymbol{D}$ is decomposed into several *transmission matrices*[1]. The accompanying diagram shows the holding time of configurations and the bandwidth utilization of connections from the source ToRs (shown vertically) in each configuration, where shaded regions between configurations denote switching overhead and white area denote unutilized bandwidth.

In Fig. 6.4 (a), $\boldsymbol{D}$ is decomposed into four configurations with no bandwidth loss. However, frequent reconfiguration causes long switching overhead. In Fig. 6.4 (b), an alternative decomposition of $\boldsymbol{D}$ gives three configurations with shorter overall switching overhead, but bandwidth under-utilization exists in every configuration, given that some connections become idle earlier than others after their

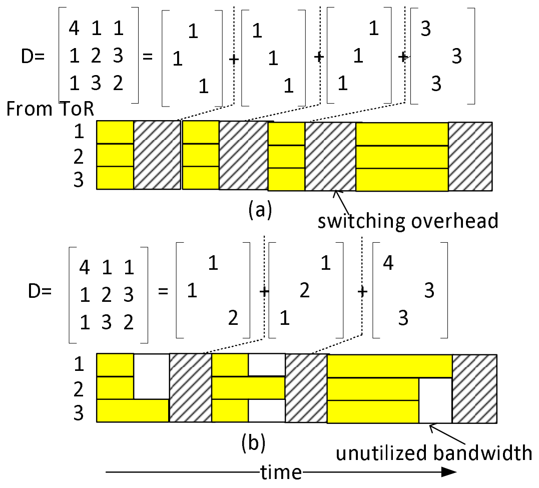


Figure 6.4: Two sets of configurations with the same total transmission time: (a) Four configurations with large overhead and no bandwidth loss; (b) Three configurations with bandwidth loss.

scheduled traffic is depleted. Both decompositions lead to the same total transmission time.

We make several important observations from the example. First, having a larger number of configurations results in finer transmission granularity, which makes it easier to maintain high bandwidth utilization in each configuration. However, it also introduces longer switching overhead, whose impact can be quite substantial if slow-switching optical networks, such as MEMS switches [40, 41] , are adopted. Hence, it is important to find a good balance between bandwidth utilization and switching overhead. Second, the weight (and thus holding time) of a configuration is determined by the largest entries (LEs) in the corresponding transmission matrix, that is, the connection with the most scheduled traffic, and bandwidth loss is caused by the gap between LEs and other entries in the transmission matrix. Therefore, in order to reduce optical bandwidth loss, we should find a decomposition such that every connection in each configuration has a similar amount of traffic to transmit.

The design of CBA is guided by the above observations. To keep the overall switching overhead at a reasonable level, the CBA algorithm first determines the maximum number of configurations allowed for each decomposition according to the parameters of the OCS network, such as the switching overhead σ and the net-

[1]In practice, diagonal entries d_{ii} should be 0, because local traffic does not go through the network. They are set to be nonzero in the example for easy illustration.

work size N. It also adopts an efficient "filtering" mechanism to group entries with similar value in each traffic matrix. The detail is shown as follows.

Given a traffic demand matrix $\boldsymbol{D}$, we first calculate its maximum line sum, denoted by M, that is, $\sum_{i=1}^{N} d_{ij} \leq M$, $\sum_{j=1}^{N} d_{ij} \leq M, \forall i, j \in \{1, 2, \ldots, N\}$. Then, we divide $\boldsymbol{D}$ into a *coarse* part and a *fine* part by representing $\boldsymbol{D}$ as the sum of a weighted *quotient* matrix with only integer entries $\boldsymbol{Q} = \{q_{ij}\}$ (coarse part) and a *residue* matrix $\boldsymbol{R} = \{r_{ij}\}$ (fine part), that is,

$$\boldsymbol{D} = \frac{M}{\mu N} \times \boldsymbol{Q} + \boldsymbol{R} \tag{6.2}$$

where $q_{ij} = \lfloor \frac{d_{ij}}{M/\mu N} \rfloor$, and $r_{ij} = d_{ij} - \frac{M}{\mu N} \times q_{ij}$. In the above equation, μ is a factor whose value is determined by the network parameters. How to find the optimal value of μ will be discussed later.

It is easy to see that only entries larger than $\lfloor \frac{M}{\mu N} \rfloor$ in $\boldsymbol{D}$ would appear in $\boldsymbol{Q}$. Meanwhile, any entry in $\boldsymbol{R}$ must be less than $\lfloor \frac{M}{\mu N} \rfloor$. Next, we schedule $\boldsymbol{Q}$ and $\boldsymbol{R}$ separately. To decompose $\boldsymbol{Q}$, we construct a *bipartite multigraph* G_Q from $\boldsymbol{Q}$ according to the following procedure. Rows and columns of $\boldsymbol{Q}$ are translated to two sets of vertices A and B in G_Q, and each entry q_{ij} is translated to q_{ij} edges connecting vertices $i \in A$ and $j \in B$, as shown in the example in Fig. 6.5.

Then, we find a *minimum edge coloring* of G_Q, which assigns the minimum number of colors to each edge such that no two adjacent edges have the same color, using existing algorithms [119]. Clearly, edges with the same color constitute a matching in the bipartite graph, thus can be mapped back to a valid configuration, where an edge connecting $i \in A$ and $j \in B$ is translated to 1 at the corresponding entry and all other entries are 0s. As a result, the coarse part can be covered by the configurations obtained, with a weight of $\frac{M}{\mu N}$ for each configuration. Meanwhile, we also obtain the transmission matrix $\boldsymbol{X}'_{\boldsymbol{k}}$ corresponding to each configuration $\boldsymbol{O}'_{\boldsymbol{k}}$, which is equal to $\frac{M}{\mu N} \times \boldsymbol{O}'_{\boldsymbol{k}}$. The apostrophe in symbols indicates that they are intermediate results that may be modified in the next phase.

To decompose the residue matrix $\boldsymbol{R}$, we choose any N *non-overlapping* permutation matrices as configurations, each of which has exactly one entry in each row or column equal to 1. These N matrices adds up to an all-1 matrix, which can be predefined arbitrarily without computation. The transmission matrix $\boldsymbol{X}'_{\boldsymbol{k}}$ of a configuration $\boldsymbol{O}'_{\boldsymbol{k}}$ is the *entrywise* product of $\boldsymbol{R}$ and $\boldsymbol{O}'_{\boldsymbol{k}}$, whose entry $x'^{k}_{ij} = r_{ij} \times o'^{k}_{ij}$, and the weight ϕ'_k equal to the *largest entry* in $\boldsymbol{X}'_{\boldsymbol{k}}$. Combining the configurations

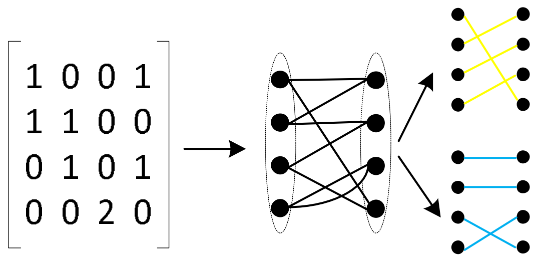


Figure 6.5: The bipartite multigraph of the given matrix can be covered with two colors.

generated for $\boldsymbol{Q}$ and $\boldsymbol{R}$, we have a set of configurations and corresponding transmission matrices that covers the traffic demand matrix $\boldsymbol{D}$.

Now, we analyze some important properties of CBA and discuss how to determine the value of μ. First, we have the following lemma on the decomposition of $\boldsymbol{Q}$.

Lemma 6.1. *For arbitrary traffic demand matrix $\boldsymbol{D}$, the coarse part (i.e.,$\frac{M}{\mu N} \times \boldsymbol{Q}$) can be covered by μN configurations with the same weight $\frac{M}{\mu N}$.*

Proof. Given the maximum line sum M of $\boldsymbol{D}$, the maximum row sum of $\boldsymbol{Q}$ satisfies

$$\sum_{i=1}^{N} q_{ij} = \sum_{i=1}^{N} \left\lfloor \frac{d_{ij}}{M/\mu N} \right\rfloor \leq \left\lfloor \frac{\sum_{i=1}^{N} d_{ij}}{M/\mu N} \right\rfloor \leq \left\lfloor \frac{M}{M/\mu N} \right\rfloor \leq \mu N$$

and similarly, the maximum column sum $\sum_{j=1}^{N} q_{ij} \leq \mu N$.

Since all entries in $\boldsymbol{Q}$ are integers, we know the maximum vertex degree in the bipartite multigraph G_Q is at most μN. According to König's theorem [120], every bipartite graph has a minimum edge coloring with the same number of colors as its maximum degree. Therefore, the coarse part can be covered by at most μN configurations with the same weight $\frac{M}{\mu N}$. □

Regarding the total transmission time of all the configurations obtained by CBA, we have the following lemma.

Lemma 6.2. *Give a traffic demand matrix $\boldsymbol{D}$, the maximum total transmission time of the obtained configurations can be represented as a function of μ:*

$$f(\mu) = \frac{M}{S}(1 + \frac{1}{\mu}) + (\mu + 1)N\sigma \tag{6.3}$$

Proof. As shown previously, the coarse part can be covered by at most μN configurations with the same weight $\frac{M}{\mu N}$. Given the normalized optical link bandwidth S, we know the total holding time of these μN configurations is at most $\frac{M}{S}$. Meanwhile, the residue matrix $\boldsymbol{R}$ is decomposed into N non-overlapping configurations, each of which has the weight equal to the largest entry in the corresponding transmission matrix. Since any entry in $\boldsymbol{R}$ is smaller than $\frac{M}{\mu N}$, the total holding time of these N configurations is also at most $\frac{M}{\mu S}$. Overall, the traffic matrix can be covered by a maximum of $(\mu + 1)N$ configurations, which introduces $(\mu + 1)\sigma N$ switching overhead. Add all these terms together, and we can get the maximum possible transmission time shown in the lemma. $\square$

By solving the equation $\frac{df(\mu)}{d\mu} = 0$, we have the following corollary:

Corollary 6.1. *The maximum transmission time is minimized with* $\mu = \sqrt{\frac{M}{\sigma NS}}$.

Corollary 6.1 gives several interesting findings on the properties of the OCS network. First, μ increases with the maximum line sum of the traffic demand matrix M, which is generally proportional to the length of traffic accumulation stage T. Therefore, we can see that a longer scheduling period allows more configurations, which leads to higher optical link bandwidth utilization. Second, besides switching overhead σ, μ also decreases with the optical link capacity S, because the bandwidth loss equivalent to having one extra switching overhead would be more substantial with higher S. By setting μ to $\sqrt{\frac{M}{\sigma NS}}$ as in Corollary 6.1, CBA can find an optimal balance between bandwidth utilization and switching overhead, which leads to short overall transmission time.

6.5.3 The EPS Network Bandwidth Allocation

By shifting the part of traffic that causes optical bandwidth loss to the EPS network, CBA uses the flexible EPS network to improve bandwidth utilization of the configurations obtained from Phase 1. It also seeks to reduce the switching overhead by removing the configurations with little scheduled traffic and sending their traffic via the EPS network. Overall, the goal of this phase is to minimize the total transmission time required.

Problem Formulation: We show that this problem can be formulated as a Multiple-class Multidimensional Knapsack Problem (MMKP) [118], in which there

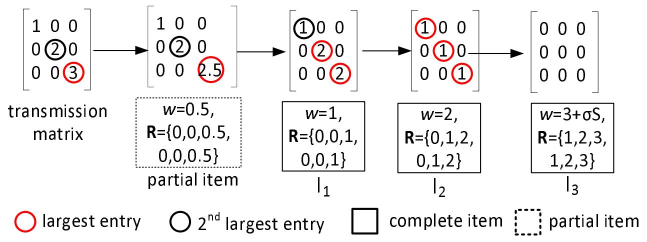


Figure 6.6: An example of transforming a transmission matrix $\boldsymbol{X}$ into a group of items.

are a multidimensional knapsack and multiple groups of items with different values and volumes. The objective is to maximize the total value of items included in the knapsack under the constraints that *at most one* item from each group can be put in the knapsack and the total volume of items in the knapsack does not exceed the knapsack's capacity. We explain the problem formulation in detail next.

As mentioned in the network model, the traffic to be sent through the EPS network during the transmission stage T can be represented by an $N \times N$ matrix $\boldsymbol{E}$. The maximum line sum of $\boldsymbol{E}$ is bounded by CT, which is the maximum traffic volume that can enter/leave a ToR during the transmission stage. Therefore, the EPS network can be considered as a "knapsack" with $2N$ dimensions, where each dimension represents a row or column in $\boldsymbol{E}$ and has CT capacity.

From Fig. 6.4, we can see that the weight of a configuration is dominated by the largest entries (LEs) in its transmission matrix. Therefore, if some traffic can be shifted from the LEs of a transmission matrix to the EPS network, then the weight of the corresponding configuration can be reduced. Following this observation, we can transform a transmission matrix into a group of "items" to be put into the knapsack (i.e., the EPS network). Each item is represented as $(w, \boldsymbol{P})$, in which w is the item "value" representing the reduction in the configuration weight, and the item "volume" $\boldsymbol{P}$ is a vector of $2N$ length recording the traffic volume required to be shifted to the EPS network in order to achieve such reduction. For clarity, we illustrate the transformation procedure using a small example shown in Fig. 6.6.

Suppose there is a 3×3 transmission matrix $\boldsymbol{X}$. First, we calculate the difference between the LE and the *second largest* entry in $\boldsymbol{X}$, denoted by Δ. We know that if $\delta(\delta \leq \Delta)$ amount of traffic is shifted from the LE (red circle) of $\boldsymbol{X}$ to the EPS network, the configuration weight can be reduced by δ. The above relationship can be represented using a series of items as shown in Fig. 6.6. We call the items with

$\delta < \Delta$ *partial items* and the item with $\delta = \Delta$ a *complete item*, and denote the first complete item by I_1. Note that, because shifting traffic from the LE to the EPS network would reduce the remaining capacity of two lines (the corresponding row and column) in $\boldsymbol{E}$, there are two non-zero entries in $\boldsymbol{P}$ of each item.

After shifting Δ traffic volume from the LE to the EPS network, $\boldsymbol{X}$ now has two equal-valued LEs. We repeat the above procedure to find the second complete item I_2 and the partial items in between, except that now we must shift traffic from both LEs in order to further reduce the configuration weight, thus $\boldsymbol{P}$ of each item has four nonzero entries, as shown in Fig. 6.6. Finally, we use the same procedure to generate I_3 and the partial items between I_2 and I_3. Since the traffic in $\boldsymbol{X}$ would be completely shifted to the EPS network if I_3 is put in the "knapsack," we can also remove the switching overhead it causes. Therefore, an extra σS is added to the value w in I_3, which is the weight reduction equivalent to eliminating one switching overhead given the optical link bandwidth S.

Given K transmission matrices $X'_k, k \in \{1, 2 \ldots, K\}$ from Phase 1, we can use the above procedure to transform them into K groups of items. We can see that the objective of minimizing the total transmission time is equivalent to picking at most one item from each group to put into the knapsack, such that the total value of items in the knapsack is maximized. Therefore, the problem can be formulated as a Multiple-class, Multidimensional Knapsack Problem (MMKP), which is NP-hard.

Heuristic Algorithm: Given that existing heuristic algorithms for solving MMKP are very complex, they cannot scale with the large data center size [118], hence, we design a time-efficient heuristic algorithm, shown as follows.

We again use a simple example to illustrate some important design concerns. As shown in Fig. 6.7, suppose there are four transmission matrices and the maximum line sum of $\boldsymbol{E}$ is 1. Fig. 6.7 presents three possible solutions for allocating EPS network bandwidth. Solution (a) reduces the total configuration weight by 1 through shifting 1 unit of traffic from two entries of $\boldsymbol{X_1}$ to $\boldsymbol{E}$. Further reduction is impossible even though there is unallocated bandwidth in the EPS network, because all the lines in $\boldsymbol{E}$ required to reduce the weight of other configurations are full. Solution (b) fully utilizes the EPS network and reduces the total holding time by 2. By shifting all the entries in $\boldsymbol{X_4}$ completely to $\boldsymbol{E}$, the optimal solution (c) allows the corresponding configuration to be removed and achieves a total weight reduction of $2 + \sigma S$.

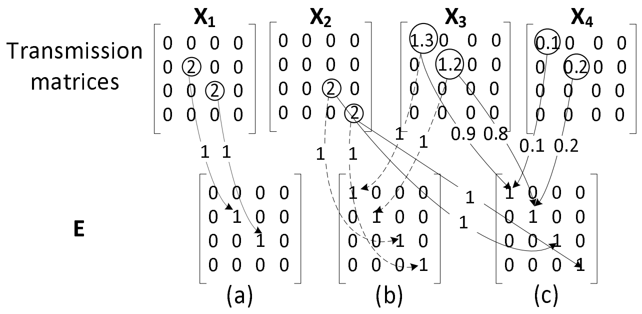


Figure 6.7: Three possible EPS network bandwidth allocations.

There are two important observations from the above example. First, some lines in $\boldsymbol{E}$ are more "valuable" than others if a larger amount of traffic is contending for them. If reducing the weight of a configuration requires shifting traffic to many valuable lines, then it should have a low priority when considered for transmission via the EPS network (e.g., $\boldsymbol{X_1}$ in Fig. 6.7). Second, if a transmission matrix is completely shifted to the EPS network, we can reduce the overall switching overhead as well. Therefore, configurations with little traffic should be given a high priority for transmission through the EPS network (e.g., $\boldsymbol{X_4}$ in Fig. 6.7).

Phase 2 of CBA is guided by the above observations. Suppose there are K transmission matrices $\boldsymbol{X'_k}, k \in \{1, 2, \ldots, K\}$ from phase 1. CBA uses the aforementioned procedure to transform each transmission $\boldsymbol{X'_k}$ to a group of *complete* items $I_{k,j}, 0 \leq j \leq N$. Then, a priority is assigned to each item, which decides the order the item is considered during the EPS network bandwidth allocation.

To calculate the item priority, CBA evaluates the "value" of each line in $\boldsymbol{E}$ first. This is done by calculating the line sum of traffic demand matrix $\boldsymbol{D}$ first. The results can be represented by a vector $\boldsymbol{V'}$ with $2N$ entries, each of which represents the sum of a line. Then, each entry in $\boldsymbol{V'}$ is divided by the sum of all the entries of $\boldsymbol{V'}$, the obtained *value vector* $\boldsymbol{V}$ is also a vector of $2N$ length, in which the sum of all the elements is equal to 1. In the second step, CBA calculates the total traffic volume in a traffic matrix $\boldsymbol{X'_k}$, that is, the sum of all its entries, denoted as $|X'_k|$. The priority of an item $I_{k,j} = \{w, \boldsymbol{P}\}$ is then set to $w/(\boldsymbol{V} \cdot \boldsymbol{P}) + \frac{N\sigma S}{|X'_k|}$, where the dot product of two vectors $\boldsymbol{V} \cdot \boldsymbol{P} = \sum_{i=1}^{2N} v_i p_i$.

Next, CBA iteratively finds the proper item in each group to put in $\boldsymbol{E}$. It uses a priority queue pq to efficiently locate the item with the highest priority for consideration. Initially, CBA inserts the *first* complete items of the K groups into pq. In each iteration, the item with the largest priority, say, $I_{k,j}$, is dequeued from pq for consideration. CBA maintains the invariant that at most one item from each group is in pq at any time, and items from the same group are considered in a strictly ascending order, which means that when $I_{k,j}$ is dequeued, all the previous complete items in the group would have been dequeued.

After $I_{k,j}$ is dequeued, CBA tries to reach $I_{k,j}$ from the previous item $I_{k,j-1}$ by shifting traffic from $\boldsymbol{X_k'}$ to $\boldsymbol{E}$ accordingly ($I_{k,0}$ is assumed to be a special item representing the original transmission matrix).

There are two possible cases. If the EPS network is able to accommodate all the traffic to be shifted, CBA updates $\boldsymbol{X_k'}$ and $\boldsymbol{E}$, then *enqueues* the next complete item $I_{k,j+1}$ if any. When $I_{k,j}$ is the final complete item in the group, CBA removes $\boldsymbol{X_k'}$ as its traffic has been completely shifted to the EPS network.

Otherwise, CBA identifies the minimum remaining capacity of the lines of $\boldsymbol{E}$ required by the traffic to be shifted, and shifts traffic from $\boldsymbol{X_k'}$ by that amount. Since it is impossible to further reduce the weight of $\boldsymbol{X_k'}$ in this case, CBA outputs the finalized transmission matrix represented as $\boldsymbol{X_k}$, and updates the remaining capacity of $\boldsymbol{E}$. When pq is empty, CBA outputs the finalized $\boldsymbol{E}$ and terminates. For a summary of the above algorithm, please refer to Table 6.2.

6.5.4 Complexity Analysis

Next, we analyze the time complexity of CBA. The time complexity of Phase 1 is determined by the edge coloring procedure, which takes $O(N^2 \log N)$ time to find a coloring consisting of at most $O(N)$ colors for the bipartite multigraph G_Q [119]. Adding the N configurations required to schedule the residue matrix $\boldsymbol{R}$, and there are at most $(\mu + 1)N$ transmission matrices generated in this phase.

In the EPS network bandwidth allocation phase, CBA decomposes each transmission matrix into at most N complete items, which requires iteratively finding the difference between the LEs and the second largest entries in each transmission matrix. This procedure takes $O(N)$ time provided that we sort all the entries of each transmission matrix in advance, which incurs a one-time-only $O(N \log N)$ time. As there are $O(N)$ transmission matrices, the time of this procedure is also $O(N^2 \log N)$.

Table 6.2: The EPS Network Bandwidth Allocation

```
Input: Transmission matrices $\boldsymbol{X'_k}, k \in \{1, 2, \dots, K\}$ from Phase 1
Output: Finalized transmission matrices $\boldsymbol{X_k}$, $\boldsymbol{E}$
Step 1: Transform each $\boldsymbol{X'_k}$ into a group of complete items $I_{k,j}$ and assign
priority to each item;
Step 2: Insert the first complete item of each group $I_{k,1}$ into $pq$
Step 3:
while($pq$ is not empty) {
  Dequeue the item with maximum priority, say, $I_{k,j}$;
  Move towards $I_{k,j}$ from $I_{k,j-1}$;
  if (EPS network has enough capacity) {
     Shift traffic from $\boldsymbol{X'_k}$ to $\boldsymbol{E}$ accordingly;
     If $I_{k,j}$ is the final item in that group, remove $\boldsymbol{X'_k}$;
     If not, enqueue the next item $I_{k,j+1}$; }
  else {
     Check minimum remaining capacity of lines in $E$ required;
     Shift traffic from $\boldsymbol{X'_k}$ by that amount;
     Output finalized transmission matrix $\boldsymbol{X_k}$; }
}
Step 4: Output finalized traffic allocation matrix $\boldsymbol{E}$
```

Next, calculating the priority of $O(N^2)$ items takes $O(N^2)$ time, as the vector product can be obtained in constant time with distributed computation. Also, each item will be enqueued and dequeued at most once. Provided that a heap structure is adopted for the priority queue, each enqueue operation takes $O(\log N)$ time and dequeue takes $O(1)$ time. After each dequeue operation, CBA checks the remaining capacity of lines in $\boldsymbol{E}$, which can be done in constant time if a series of parallel comparators is used.

Overall, with proper data structures and hardware implementation, the overall time complexity of CBA is $O(N^2 \log N)$. It is asymptotically faster than the Edmonds' algorithm for finding a maximum weighted matching adopted in current Hypac DCNs [40, 41], which has $O(N^3)$ time complexity.

6.6 Network Condition for Guaranteed Performance

Though extensive experiments and simulations have demonstrate that Hypac DCNs can achieve close to 100% throughput and low latency under tested traffic patterns [40, 41], an interesting and important question to ask is that whether such a performance can be *guaranteed* under all traffic patterns. In this section, we prove that it is indeed feasible for Hypac DCNs to guarantee 100% throughput with a bounded delay. We give the sufficient condition on system parameters, such as the length of a scheduling period, bandwidth capacity of the EPS/OCS networks and switching overhead, for a Hypac DCN to achieve guaranteed performance under the CBA algorithm.

Recall that each scheduling period of a Hypac DCN has three stages, among which both the traffic accumulation stage and transmission stage have the same length T, and the configuration stage takes H time. The traffic arriving during an accumulation stage forms a *batch*, which can be similarly represented by a traffic demand matrix $\boldsymbol{D(T)} = \{d_{ij}(T)\}$. Under the hose traffic constraint, a ToR can send or receive at most T units of traffic during T time, that is, the maximum line sum of $\boldsymbol{D}(T)$ is T. Then we have the following property regarding a Hypac DCN.

Property 6.1. *The sufficient and necessary condition for a Hypac DCN to achieve 100% throughput and a bounded delay of $2T + H$ is that any arbitrary traffic demand matrix $\boldsymbol{D(T)}$ can be transmitted within T time.*

Proof. Without loss of generality, we assume the system has no backlog traffic initially. In the first scheduling period, any packet in the first batch will be transmitted by the end of stage 3 with at most $2T + H$ delay, given that the above condition is satisfied, as shown in Fig. 6.3. Also, no backlog traffic will be left over to the next batch. Hence, any traffic arriving in the next batch will also be transmitted with $2T + H$ delay. By induction, the above condition is sufficient to guarantee 100% throughput and a delay bound of $2T + H$.

We prove necessity by contradiction. Suppose there exists one adversarial traffic demand matrix that requires longer than T transmission time, which means a portion of traffic will be rolled over to the next batch. Theoretically, it is possible that the adversarial traffic demand matrix is found for every batch, then the network would be unstable and bounded packet delay cannot be achieved. □

Next, we analyze the network condition for a Hypac DCN to achieve such performance guarantee under the CBA algorithm. Recall that Phase 2 of the CBA algorithm reduces the total transmission time of the OCS network configurations by shifting a part of traffic from the transmission matrices to the EPS network, which subjects to the constraint that the maximum line sum of $\boldsymbol{E}$ can be no larger than CT. We have the following lemma on the minimum amount of reduction in total transmission time the EPS network can achieve.

Lemma 6.3. *Phase 2 of CBA can reduce the total transmission time of all the configurations from Phase 1 by* at least $\frac{CT}{S}$.

Proof. One observation we make is that there must be at least one line (i.e., row or column) in $\boldsymbol{E}$ that reaches the maximum line sum of CT when Phase 2 terminates, if not all the transmission matrices have been completely shifted to the EPS network. This can be proved by contradiction. Assume the maximum line sum in $\boldsymbol{E}$ is smaller than CT after Phase 2 terminates. According to the CBA algorithm, a transmission matrix $\boldsymbol{X_k}$ will be considered as final only if one or more lines in $\boldsymbol{E}$ reach the maximum line sum CT when shifting traffic from the transmission matrix to the EPS network. This contradicts with the assumption, hence, we know at least one line in $\boldsymbol{E}$ has the maximum sum of CT.

Without loss of generality, suppose line j in $\boldsymbol{E}$ has the line sum of $C \cdot T$ after CBA terminates. Since there can be at most one non-zero entry in each line of a transmission matrix, at most one entry in line j of $\boldsymbol{E}$ will be increased when shifting traffic from a transmission matrix to the EPS network. In other words, any traffic shifted from a transmission matrix to line j reduces the weight of the configuration by the same amount. Therefore, the total configuration weight is reduced by at least CT, which accounts for a reduction of $\frac{CT}{S}$ in transmission time given the optical link bandwidth S. The lemma is thus proven. □

Combining the above lemmas and the results in the previous section, we have the following theorem.

Theorem 6.1. *Under the CBA algorithm, a Hypac DCN satisfying the following condition*

$$S \geq \frac{(1-C)^2}{(\sqrt{1-C+\lambda^2}-\lambda)^2} \tag{6.4}$$

where $\lambda = \sqrt{\frac{N\sigma}{T}}$*, is sufficient to schedule any* $\boldsymbol{D(T)}$ *within time* T*, thus can guarantee 100% throughput and a bounded delay of* $2T + H$*.*

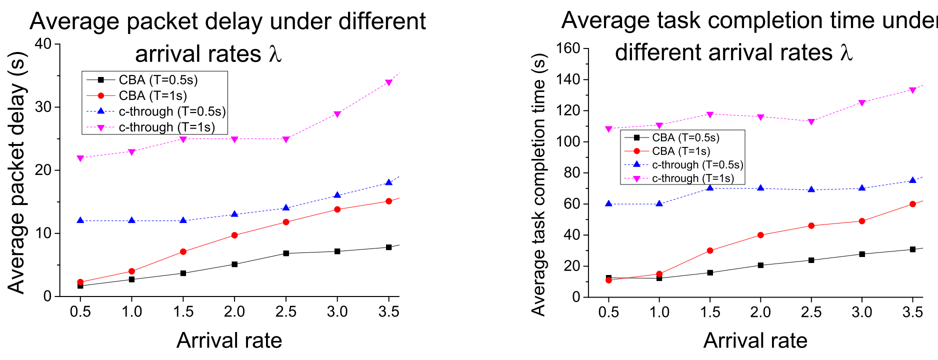
Proof. We give a sketchy proof of this theorem. Details are omitted due to limited space. From Lemma 6.3, we know any arbitrary traffic batch $\boldsymbol{D(T)}$ can be transmitted within T time if the total transmission time of the OCS network configurations from Phase 1 is *no larger* than $T + \frac{CT}{S}$. Given the fact that the maximum line sum of $\boldsymbol{D(T)}$ is T, we can obtain the maximum total transmission time of the configurations from Phase 1 by substituting the value of μ as in Corollary 6.1 in Lemma 6.2. The theorem can be proven by solving the inequality $\frac{T}{S}(1 + \sqrt{\frac{\sigma NS}{T}}) + (\sqrt{\frac{T}{\sigma NS}} + 1)N\sigma \leq T + \frac{CT}{S}$. □

From the theorem, we can see that the data center size N, the length of T and switching overhead σ are all important factors in the sufficient condition for a Hypac DCN to guarantee 100% throughput. A longer scheduling period leads to a smaller optical link bandwidth S required for the performance guarantee, because as indicated in Corollary 6.1, we can have more configurations with longer T, which in turn results in higher optical bandwidth utilization. When T is large enough such that λ is negligible, $S = 1 - C$ is sufficient to guarantee 100% throughput. This is consistent with the finding in the BirkhoffCvon Neumann switches [114], which states that any traffic demand matrix can be decomposed into a sufficient number of configuration matrices without bandwidth loss.

On the other hand, it is important to note that a long scheduling period also leads to longer packet delay from the traffic accumulation stage. Meanwhile, we notice that reducing the switching overhead has similar effects to increasing T without introducing extra packet delay. Therefore, finding optical interconnects faster than the MEMS switches, such as optical multistage networks [121], is critical in further improving the performance of Hypac DCNs.

6.7 Performance Evaluation

We build a packet-level network simulator to evaluate the performance of CBA, which, though cannot capture every detail of the practical DCN environment, provides valuable insights on network properties of interests. As packet-level simulation is computational intensive, we consider a small-sized data center with 32 ToRs



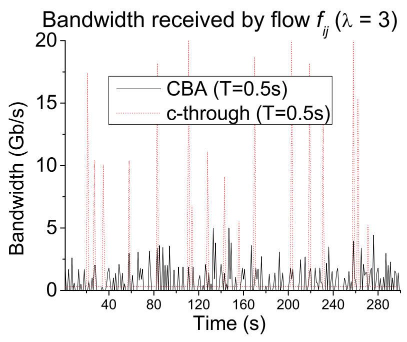
(a) (b)

Figure 6.8: Delay performance under different arrival rates λ. (a) Average packet delay. (b) Average task completion time.

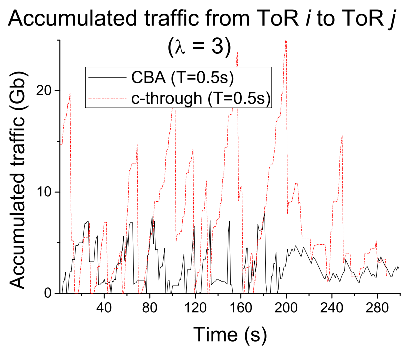
for tractability, each of which is connected to 40 servers via 1Gb/s ports, as in [40]. The EPS network is a tier-2 fat-tree, with the aggregate core switch link bandwidth for each ToR set to 10Gb/s. This corresponds to an oversubscription ratio of 4:1, or, $C = 0.25$. We also set the optical link bandwidth to 40 Gb/s, that is, $S = 1$, and the reconfiguration overhead for the OCS network to 1 ms.

In the transmission stage, CBA configures the OCS network according to the set of weighted configurations obtained and configures the EPS network by allocating bandwidth proportional to the entries in $\boldsymbol{E}$ to corresponding flows. For comparison, we implement the popular Hypac DCN c-through [40]. As mentioned earlier, c-through configures the OCS network using a Maximum Weighted Matching algorithm that connects ToR pairs with the most accumulated traffic in each scheduling period. Here, we assume the EPS network in c-through allocates equal bandwidth to all *active* flows from each ToR (a flow f_{ij} is active if there is traffic from ToR i to ToR j). If flows received by a ToR have a total bandwidth more than C, the bandwidth of these flows is downsized proportionally, and any bandwidth vacancy at a ToR due to downsizing will be redistributed to other active flows. This process repeats till the bandwidth received by all the flows is stabilized.

We have tested CBA under a comprehensive set of traffic patterns. Due to limited space, only representative results for all-to-all data shuffle are shown here. Data shuffle is a necessary procedure in many communication-intensive MapReduce/Hadoop operations [7]. A data shuffle task involves multiple hosts, in which every host sends a flow consisting of a certain amount of data to every other host



(a)



(b)

Figure 6.9: (a) Bandwidth received by flow f_{ij} over time; (b) Accumulated traffic from ToR i to ToR j.

participating in the shuffle. It generates strongly correlated traffic, as its completion depends on the slowest flow [42]. To make the simulation tractable, we assume the arrival of data shuffle tasks follows a Poisson distribution with arrival rate λ, in which the probability density function of the interval between arrivals t is $f(t) = \lambda e^{-\lambda t}$ and the average number of task arrivals per second is λ. Here, we make a simplifying assumption that each task involves m random chosen ToRs, where m is an integer uniformly chosen from 2 to 32. We also assume the amount of data to be transferred from one ToR to every other ToR in a data shuffle task follows a uniform distribution from 10Mb to 2Gb. All packets have the same size of 1024 Bytes. Each simulation is run for a sufficiently long period ($1000s$ in simulation time) to ensure accurate results.

First, we measure the average packet delay under different arrival rates λ. As shown in Fig. 6.8(a), c-through suffers long packet delay when λ is small, because that one OCS network configuration would hold for an entire transmission stage, which results in severe bandwidth under-utilization if there is not enough traffic accumulation between connected ToR pairs. The bandwidth wasting can be somewhat mitigated by adopting a shorter scheduling period, as there is a sharp decrease in packet delay when changing T from 1s to 0.5s. However, given the considerable control and switch overhead imposed by reconfiguring the OCS network, the bandwidth under-utilization problem cannot be fundamentally solved by reducing the scheduling period. On the other hand, CBA achieves much lower packet delay under all tested arrival rates, particularly when λ is small, due to that CBA generates multiple configurations in one scheduling period, leading to higher optical bandwidth utilization.

We then measure the average task completion time, defined as the period between the arrival of a task and the completion of its last traffic flow. As shown in Fig. 6.8(b), the simulation results confirm that c-through is inadequate in handling strongly correlated traffic, as pointed out in [42]. Meanwhile, the average task completion time under CBA is shown to be only a small fraction of that in c-through. One major reason is that the EPS network is utilized to effectively complement the OCS network in CBA, which prevents small flows from being stranded.

To have a closer look at the performance of CBA and c-through, we examine a 300s trace recording the bandwidth, i.e., the traffic volume transmitted per second, of a flow f_{ij} between a pair of randomly chosen ToRs in Fig. 6.9(a). We can see that the flow bandwidth is characterized by sparse "spikes" in c-through, due to sporadic available optical bandwidth. In comparison, the flow receives bandwidth from the OCS network much more frequently under CBA. As a result, as shown in Fig 6.9(b), CBA manages to keep the accumulated traffic from ToR i to ToR j at a much lower level than c-through. This observation indicates that CBA enables a much "smoother" transmission that has lower delay variability (jitter). Therefore, CBA leads to more predictable network performance than c-through.

6.8 Conclusion

In this chapter, we present a thorough study on the collaborative network configuration problem, which holds the key to addressing the various challenges faced by Hypac DCNs currently. First, we develop a network model that accurately abstracts the key characteristics of the OCS/EPS networks. Then, we propose a low-complexity scheduling algorithm called CBA, which effectively leverages the strengths of optical and electrical transmission to configure the two networks collaboratively. Additionally, we give the sufficient condition for a Hypac DCN to achieve 100% throughput and bounded delay under CBA. Finally, we evaluate C-BA through simulation, which demonstrates that CBA not only drastically reduces average packet delay, but also significantly improves network predictability and the performance under correlated traffic.

Chapter 7

Conclusions

Cloud computing has transformed the IT industry in a profound way. Large data centers have emerged to power the world's financial market, perform analytic and data mining tasks of unprecedent scale, and host services that billions of people use in their daily life. As the tasks and services hosted on cloud rely on the collaboration and communication between numerous servers, the network is a determining factor in their performance.

This dissertation studies several critical and closely coupled areas in designing high-performance, low-cost and multicast-capable DCNs. First, we study optical packet switching (OPS) [122–127], a key component in achieving high bandwidth and energy-efficiency in future DCNs. In [122], we propose an aggregation model that can accurately predict the performance of OpCut, a hybrid optical/electrical packet switch, in terms of average latency and packet drop under different traffic patterns. In [123, 124, 126], we study how to effectively transmit multicast traffic in all-optical packet switches. We designed a novel FDL buffer structure called M-FDLs, based on which we proposed a time-efficient, low-latency multicast scheduling algorithm. We also designed a parallel and pipelined scheduler architecture that achieves line-rate scheduling, which fundamentally solves the scalability problem of the electrical scheduler in high-speed OPS. Incorporating multicast capability into switches is the first step towards multicast-capable DCNs, which we study next.

The second part of our research focuses on how to deploy multicast communication with guaranteed bandwidth in fat-tree DCNs in a cost-effective manner [128–134]. In [128–131], we have explored server redundancy in HA data centers to reduce the cost of nonblocking multicast fat-tree DCNs. We give theory and algorithms to help operators achieve significant cost reduction in building nonblocking multicast fat-tree DCNs, even under the condition of possible server failures. In [132–134], we present a low-complexity on-line multicast scheduling algorithm called BCMS for oversubscribed fat-tree DCNs, which leverages centralized control and global knowledge of network condition of the SDN framework for data centers to achieve traffic load balance. We show that BCMS can guarantee bounded congestion in an oversubscribed fat-tree DCN for an arbitrary sequence of multicast flow requests under hose traffic model.

Third, we study how to achieve predictably high network performance in Hypac DCNs [135]. We find that collaborative network configuration holds the key to addressing the various challenges currently faced by Hypac DCNs. We design a low-complexity scheduling algorithm called CBA, which not only drastically reduces average packet delay, but also significantly improves network predictability by utilizing the optical and electrical networks in a complementary manner. Moreover, we show that using CBA, a Hypac DCN can achieve 100% throughput and bounded delay.

In summary, this dissertation combines theoretical analysis, algorithm design, network optimization, and simulation techniques to provide a systematic and thorough study on the design of high-performance, cost-effective and multicast-capable DCNs. Our results reveal interesting and important findings on the fundamental design principles of DCNs, and have the potential to greatly boost the performance of numerous cloud computing applications, as well as facilitate the cloud adoption for many future applications that rely on group communication with predictable high bandwidth.

Bibliography

[1] R. Miller, “Ballmer: Microsoft has 1 Million Servers,” *http://www.datacenterknowledge.com/archives/2013/07/15/ballmer-microsoft-has-1-million-servers/*

[2] “Amazon Web Services,” *http://aws.amazon.com/ec2*

[3] M. Armbrust, et al. “Above The Clouds: A Berkeley View of Cloud Computing.” *Technical Report UCB/EECS-2009-28,* EECS Department, U.C. Berkeley, Feb. 2009.

[4] Q. Zhang, L. Cheng, R. Boutaba, “Cloud Computing: State-of-the-art and Research Challenges”, Journal of Internet Services and Applications, 2010

[5] A. Greenberg, et al, “The Cost of a Cloud: Research Problems in Data Center Networks,” *ACM SIGCOMM CCR* Jan. 2009.

[6] M. Alizadeh, et al, “Dctcp: Efficient Packet Transport for The Commoditized Data Center”. In SIGCOMM, 2010.

[7] J. Dean and S. Ghemawat, “MapReduce: Simplified Data Processing on Large Clusters.” *6th Symposium on Operating System Design and Implementation (OSDI),* Dec. 2004.

[8] Y. Chen, et al, “A First Look at Inter-Data Center Traffic Characteristics via Yahoo! Datasets,” *IEEE INFOCOM ’11,* Mar. 2011.

[9] T. Benson, A. Anand, A. Akella and M. Zhang. “Understanding Data Center Traffic Characteristics.” *1st ACM Workshop on Research on Enterprise Networking (WREN),* 2009.

[10] S. Kandula, S. Sengupta, A. Greenberg and P. Patel. “The Nature of Datacenter Traffic: Measurements & Analysis.” *IMC’09,* 2009.

[11] S. Ghemawat, H. Gobioff and S. Leung, “The Google File System,” *ACM SOSP’03,* Oct. 2003.

[12] L. Popa, S. Ratnasamy, G. Iannaccone, et al., “A Cost Comparison of Datacenter Network Architectures,” *Proc. ACM Co-NEXT 2010,* pp. 16-28, 2010.

[13] A.R. Curtis, S. Keshav and A. Lopez-Ortiz,“LEGUP: Using Heterogeneity to Reduce The Cost of Data Center Network Upgrades.” *ACM CoNext,* 2010.

[14] A. R. Curtis, et al. “REWIRE: An Optimization-based Framework for Unstructured Data Center Network Design,” *IEEE INFOCOM,* 2012.

[15] M. Al-Fares, et al. “Hedera: Dynamic Flow Scheduling for Data Center Networks.” *NSDI,* 2010.

[16] C. HOPPS, “Analysis of an Equal-Cost Multi-Path Algorithm,” *RFC 2992, IETF,* 2000.

[17] Andrew R. Curtis and Alejandro Lopez-Ortiz. “Capacity provisioning a Valiantload-balanced network.” IEEE INFOCOM’ 09, 2009.

[18] M. Al-Fares, A. Loukissas and A. Vahdat, “A Scalable, Commodity Data Center Network Architecture,” *Proc. ACM SIGCOMM’08,* Aug. 2008.

[19] C. Guo, et al. “BCube: A High Performance, Server-centric Network Architecture for Modular Data Centers,” *ACM SIGCOMM’09,* Aug. 2009.

[20] D. Li, et al. ”FiConn: Using Backup Port for Server Interconnection in Data Centers,” *Proc. IEEE INFOCOM,* 2009.

[21] D. Guo, et al. “Expandable and Cost-Effective Network Structures for Data Centers Using Dual-Port Servers,” *IEEE Trans. Computers,* vol. 62, No. 7, pp. 1303-1317, Jul. 2013

[22] C. Guo, et al. “DCell: A Scalable and Fault-Tolerant Network Structure for Data Centers,” *ACM SIGCOMM’08,* Aug. 2008.

[23] A. Greenberg et al. “VL2: A Scalable and Flexible Data Center Network.” *Proc. ACM SIGCOMM’08*, Aug. 2009.

[24] R.N. Mysore, et al., “PortLand: A Scalable Fault-Tolerant Layer 2 Data Center Network Fabric,” *Proc. ACM SIGCOMM’09,* pp. 39-50, Aug. 2009.

[25] K. Bilal, et al., “A Taxonomy and Survey on Green Data Center Networks,” *Future Generation Computer Systems,* 2013

[26] K. Chen. “Generic and Automatic Address Configuration for Data Center Networks.” ACM SIGCOMM, 2010

[27] J.S. Turner and R.Melen, "Multirate Clos networks," *IEEE Communications Magazine,* vol. 41, no. 10, pp. 38-44, Oct. 2003.

[28] R. Melen and J.S. Turner, "Nonblocking Multirate Networks," *SIAM Journal of Computing,* vol. 18, no. 2, pp. 301-13, Apr. 1989.

[29] Y. Yang and G.M. Masson, "Nonblocking Broadcast Switching Networks," *IEEE Transactions on Computers,* vol. 40, no. 9, pp. 1005-1015, Sept. 1991.

[30] Y. Yang and G. M. Masson, "The Necessary Conditions for Clos-type Nonblocking Multicast Networks," *IEEE Transactions on Computers,* vol. 48, pp. 1214-1227, 1999.

[31] D. Li, J. Yu, J. Yu and J. Wu, "Exploring Efficient and Scalable Multicast Routing in Future Data Center Networks," *IEEE INFOCOM '11,* Mar. 2011.

[32] D. Li, Y. Li, J. Wu, S. Su and J. Yu. "ESM: Efficient and Scalable Data Center Multicast Routing," *IEEE Trans. NETWORKING,* vol. 20, No. 3, pp. 944-955, Jun. 2012.

[33] D. Li, et al, "RDCM: Reliable Data Center Multicast," *IEEE INFOCOM,* 2011.

[34] D. Li, et al., "Scalable Data Center Multicast Using Multi-class Bloom Filter," *IEEE ICNP*, 2011.

[35] H. Takahashi and A. Matsuyama, "An Approximate Solution for The Steiner Problem in Graphs," *Mathematica Japonica*, vol. 24, pp. 573-577, 1980.

[36] L. Kou, G. Markowsky and L. Berman, "A Fast Algorithm for Steiner Trees," *Acta Informatica,* vol. 15, pp. 141-145, 1981.

[37] B. Awerbuch, Y, Azar, "Competitive Multicast Routing," *Wireless Networks* vol. 1, no. 1, pp. 107-114, 1995.

[38] B. Awerbuch, Y. Azar, and S. Plotkin. "Throughput competitive online routing." *In 34th IEEE Symposium on Foundations of Computer Science,* 1993.

[39] C. Kachris and I. Tomkos, "A Survey on Optical Interconnects for Data Centers," *IEEE Communications Surveys & Tutorials,* vol. 14, no. 4, pp. 1021-1036, 2012.

[40] G. Wang, et al, "c-Through: Part-time Optics in Data Centers." *ACM SIGCOMM'10,* Aug. 2010.

[41] N. Farrington, et al., “Helios: A Hybrid Electrical/Optical Switch Architecture for Modular Data Centers.” *ACM SIGCOMM CCR* no. 4, pp. 339-350, 2011.

[42] H. H. Bazzaz, et al., “Switching The Optical Divide: Fundamental Challenges for Hybrid Electrical/Optical Datacenter Networks.” *ACM Symposium on Cloud Computing.* 2011.

[43] “Open Networking Foundation”, https://www.opennetworking.org/

[44] N. Gude, et al. “NOX:Towards an Operating System for Networks.” In SIGCOMM CCR, July 2008.

[45] Andrew R. Curtis, et al. “DevoFlow: Scaling Flow Management for High-performance Networks.” ACM SIGCOMM, 2011.

[46] Sushant Jain, et, al. “B4: Experience with a Globally-Deployed Software Defined WAN.” ACM SIGCOMM, 2013.

[47] Y. Cui, ”Wireless Data Center Networking,” IEEE Wireless Communications, vol.18, no.6, pp.46,53, Dec. 2011

[48] H. Huang, “The Architecture and Traffic Management of Wireless Collaborated Hybrid Data Center Network” ACM SIGCOMM, 2013

[49] D. Halperin, et al. “Augmenting Data Center Networks with Multi-gigabit Wireless Links” ACM SIGCOMM CCR, 2011.

[50] M. Handley. “Why the Internet Only Just Works.” BT Technology Journal, 2006.

[51] N. Mckeown, et al. “OpenFlow: Enabling Innovation in Campus Networks,” *ACM SIGCOMM CCR,* 2008.

[52] A. Shieh, S. Kandula, A. Greenberg and C. Kim. “Sharing the Data Center Network.” *USENIX NSDI*, 2011.

[53] C. Kachris, et al. “ A Survey on Optical Interconnects for Data Centers.” IEEE Communications Surveys & Tutorials, 2011

[54] X. Ye, et al., “DOS: A Scalable Optical Switch for Datacenters,” ACM/IEEE ANCS, 2010

[55] R. Proietti, et al., “40 Gb/s 8x8 Low-latency Optical Switch for Data Centers,” OSA OFC/NFOEC, 2011.

[56] H. J. Chao, Z. Jing, and K. Deng, “PetaStar: A Petabit Photonic Packet Switch,” IEEE JSAC, vol. 21, pp. 1096C1112, 2003

[57] R. Luijten, W. E. Denzel, R. R. Grzybowski, and R. Hemenway, “Optical interconnection networks: The OSMOSIS project,” in The 17th Annual Meeting of the IEEE Lasers and Electro-Optics Society, 2004.

[58] R. Hemenway, R. Grzybowski, C. Minkenberg, and R. Luijten, “Optical Packet-switched Interconnect for Supercomputer Applications,” J. Opt. Netw., vol. 3, no. 12, pp. 900C913, Dec. 2004

[59] H. Ballani, P. Costa, T. Karagiannis and A. Rowstron, “Towards Predictable Datacenter Networks,” *ACM SIGCOMM,* 2011.

[60] C. Raiciu, et al, “Improving Datacenter Performance and Robustness with Multipath TCP” *ACM SIGCOMM,* 2011.

[61] A.G.P. Rahbar and O.W.W. Yang, “Contention Avoidance and Resolution Schemes in Bufferless All-optical Packet-Switched networks: a survey,” *IEEE Communications Surveys & Tutorials* vol. 10, no. 4, pp. 94-107,

[62] Z. Zhang and Y. Yang, “Performance Analysis of Optical Packet Switches Enhanced with Electronic Buffering,” *IEEE International Symposium on Parallel & Distributed Processing*, pp. 1-9, May 2009.

[63] L. Liu, Z. Zhang and Y. Yang, “Packet Scheduling in A Low Latency Optical Packet Switch,” *Proc. of the 11th International Conference on High Performance Switching and Routing (HPSR 2010)*, Dallas, TX, June 2010.

[64] S.-T. Chuang, A. Goel, N. McKeown and B. Prabhakar, “Matching Output Queueing with A Combined Input Output Queued Switch,”in *Proc. IEEE INFOCOM 1999,* pp. 1169-1178, vol. 3, Mar. 1999.

[65] G. Bianchi and J.S.Turner, “Improved Queueing Analysis of Shared Buffer Switching Networks,” *IEEE/ACM Trans. Netw.,* vol. 1, no. 4, pp. 482-490, Aug. 1993.

[66] J.A. Schormans and J.M. Pitts, “Overflow Probability in Shared Cell Switched Buffers,” *IEEE Communi. Lett.,* vol. 4, no. 5, pp. 167-169, May 2000.

[67] Z. Zhang and Y. Yang, “A Novel Analytical Model for Switches with Shared Buffer,” *IEEE/ACM Trans. Netw.,* vol. 15, no. 5, pp. 1191-1203, Oct. 2007.

[68] S. Fong and S. Singh, “Modeling Cell Departure for Shared Buffer ATM Switch,” *Proc. IEEE Int. Conf. Communications (ICC ’98),* vol. 3, pp. 1824-1828, Jun. 1998.

[69] L. Xu, H.-G. Perros and G.-N. Rouskas, “A Queueing Network Model of An Edge Optical Burst Switching Node,” in *Proc. IEEE INFOCOM 2003,* pp. 2019- 2029, vol.3, Apr. 2003

[70] L. Xu, H.G. Perros and G. Rouskas, “Techniques for Optical Packet Switching and Optical Burst Switching,” *IEEE Commun. Mag.,* vol. 39, no. 1, pp. 136-142, Jan. 2001.

[71] M. Karol, M. Hluchyj and S. Morgan, “Input Versus Output Queueing on A Space-division Packet Switch,” *IEEE Trans. Commun.,* vol. 35, no. 12, pp. 1347- 1356, Dec 1987.

[72] T. Zhang, K. Lu and J.R. Jue, “Shared Fiber Delay Line Buffers in Asynchronous Optical Packet Switches,” *IEEE JSAC*, vol. 24, no. 4, pp. 118-127, 2006

[73] Z. Zhang and Y. Yang, “Performance Modeling of Bufferless WDM Packet Switching Networks with Limited-range Wavelength Conversion,” *IEEE Trans. Commun.,* vol. 54, no. 8, pp. 1473-1480, Aug. 2006.

[74] S. Sharma and Y. Viniotis, “Optimal Buffer Management Policies for Shared-buffer ATM Switches,” *IEEE/ACM Trans. Netw.,* vol. 7, no. 4, pp. 575-587, Aug 1999.

[75] Z. Guo, X. Luo, Y. Jin, et al, “Improving Resource Utilization in Hybrid Packet/circuit Multicasting for IPTV Delivery,” *OFC 2008,* 2008.

[76] Q. Huang and W. Zhong, “A Wavelength-routed Multicast Packet Switch with a Shared-FDL Buffer,” *Journal of Lightwave Technology,* vol. 28, pp. 2822-2829, Oct. 2010.

[77] Q. Huang and W.D. Zhong, “An Optical Wavelength-routed Multicast Packet Switch Based on Multi-timeslot Multi-wavelength Conversion,” *IEEE Photonic Technology Letter*, vol. 20, no. 18, pp. 1518-1520, 2008.

[78] H. Yang and S.J.B. Yoo, “All-optical Variable Buffering Strategies and Switch Fabric Architectures for Future All-optical Data Routers,” *IEEE Journal of Lightwave Technology,* vol. 23, pp. 3321-3330, Oct. 2005.

[79] P. Gambini, et al., “Transparent Optical Packet Switching: Network Architecture and Demonstrators in the KEOPS Project,” *IEEE Journal on Selected Areas Communications*, vol. 16, no. 7, pp. 1245-1259, Sep. 1998.

[80] C. Guillemot, et al., “Transparent Optical Packet Switching: The European ACTS KEOPS Project Approach,”*IEEE Journal of Lightwave Technology,*, vol. 16, no. 12, pp. 2117-2134, Dec. 1998.

[81] X. Qin and Y. Yang, “Multicast Connection Capacity of WDM Switching Networks with Limited Wavelength Conversion,” *IEEE/ACM Trans. Networking*, vol. 12, no. 3, pp. 526-538, June 2004.

[82] C. Zhou and Y. Yang, “Wide-sense Nonblocking Multicast in a Class of Regular Optical WDM Networks,” *IEEE Trans. Communications*, vol. 50, no. 1, pp. 126-134, Jan. 2002.

[83] Y. Wang and Y. Yang, “Multicasting in A Class of Multicast-capable WDM Networks,” *Jounral of Lightwave Technology*, vol. 20, no. 3, pp. 350-359, March 2002.

[84] D. Pan and Y. Yang, “FIFO-based Multicast Scheduling Algorithm for Virtual Output Queued Packet Switches,” *IEEE Trans. Computers,* vol. 54, no. 10, pp. 1283-1297, Oct. 2005.

[85] B. Prabhakar, N. McKeown and R. Ahuja, “Multicast Scheduling for Input-queued Switches,” *IEEE Journal on Selected Areas in Communications* vol. 15, no. 5, pp. 855-866, June 1997.

[86] M.A. Marsan, A. Bianco, P. Giaccone, E. Leonardi and F. Neri, “Multicast Traffic in Input-queued Switches: Optimal Scheduling and Maximum Throughput,” *IEEE/ACM Trans. Networking*, vol. 11, no. 3, pp. 465-477, June 2003.

[87] A. Wonfor, H. Wang, R.V. Penty, and I.H. White, “Large Port Count High-speed Optical Switch Fabric for Use Within Datacenters [Invited],” *IEEE/OSA Journal of Optical Communications and Networking,* vol.3, no.8, pp.A32-A39, Aug. 2011.

[88] H. Harai and M. Murata, “Optical Fiber-delay-line Buffer Management in Output-buffered Photonic Packet Switch to Support Service Differentiation,” *IEEE Journal on Selected Areas in Communications,* vol. 24, no. 8, pp. 108-116, Aug. 2006.

[89] W.T. Chen, C.F. Huang, Y.L. Chang and W.Y. Hwang, "An Efficient Cell-scheduling Algorithm for Multicast ATM Switching Systems," *IEEE/ACM Trans. Networking,* vol.8, no.4, pp. 517-525, 2000.

[90] Y. Hao, S. Ruepp, M.S. Berger, and L. Dittmann, "Integration of Look-ahead Multicast and Unicast Scheduling for Input-queued Cell Switches," *IEEE H-PSR 2012* pp.24-27, June 2012

[91] H. Harai and M. Murata, "High-speed Buffer Management for 40 Gb/s-based Photonic Packet Switches," *IEEE/ACM Trans. Networking,* vol. 14, pp. 191-204, Feb. 2006.

[92] N. McKeown, "The iSLIP Scheduling Algorithm for Input-queued Switches," *IEEE/ACM Trans. Networking,*, vol.7, no.4, pp.188C201, Apr.1999

[93] A. Prakash, S. Sharif, and A. Aziz, "An $O(log^2N)$ Parallel Algorithm for Output Queueing," IEEE INFOCOM 2002, pp.1623C1629, Jun. 2002.

[94] H.Furukawa, H. Harai, M. Ohta and N. Wada, "Implementation of High-speed Buffer Management for Asynchronous Variable-length Optical Packet Switch,"*OFC 2010,* Mar. 2010

[95] S. Yu, S.-C. Lee, O. Ansell and R. Varrazza, "Lossless Optical Packet Multicast Using Active Vertical Coupler Based Optical Crosspoint Switch Matrix," *IEEE Journal of Lightwave Technology,* vol. 23, no. 10, pp. 2984-2992, Oct. 2005.

[96] A. Bianco,et al., "On the Number of Input Queues to Efficiently Support Multicast Traffic in Input Queued Switches," *IEEE High Performance Switching and Routing 2003* Jun. 2003.

[97] K. Claffy, D. Andersen and P. Hick, "The CAIDA Anonymized 2010 Internet Traces,"
http://www.caida.org/data/passive/passive_2010_dataset.xml

[98] K. Bilal, et al., "Quantitative Comparisons of the State of the Art Data Center Architectures," *Concurrency and Computation: Practice and Experience,* vol. 25, no. 12, pp. 1771-1783, 2013.

[99] K. Bilal, et al., "On the Characterization of the Structural Robustness of Data Center Networks," *IEEE Transactions on Cloud Computing* vol.1, no.1, Jan. 2013

[100] M. Manzano, et al., "On the Connectivity of Data Center Networks,"*IEEE Communications Letters,* vol 17, no. 11, Nov. 2013

[101] "Data Center High Availability Clusters Design Guide." *http://www.cisco.com/en/US/docs/solutions/Enterprise/Data_Center/HA_Clusters/HA_Clusters.html*

[102] "A Technology and Networking Guide for High Availability Clusters Extended Across Multiple Data Centers," *www.cisco.com/global/EMEA/cds/corporate_marketing/HA_Clusters_White_Paper.pdf*

[103] S. Distefano, F. Longo and M. Scarpa, "Availability Assessment of HA S-tandby Redundant Clusters," *29th IEEE Symposium on Reliable Distributed Systems,* pp. 265-274, Oct. 2010.

[104] G.M. Masson and B.W. Jordan, "Generalized Multi-stage Connection Networks," *Networks,* vol. 2, pp. 191-209, 1972.

[105] F.K. Hwang, "Rearrangeability of Multiconnection Three-stage Networks," *Networks,* vol. 2, pp. 301-306, 1972.

[106] S.C. Liew, Ming-Hung Ng and C.W. Chan, "Blocking and Nonblocking Multirate Clos Switching Networks," *IEEE/ACM Transactions on Networking,* vol. 6, no. 3, pp. 307-318, Jun. 1998.

[107] Y. Yang, "An Analysis Model on Nonblocking Multirate Broadcast Networks," *ACM International Conference on Supercomputing,* pp. 256-263, 1994.

[108] N. Bonvin, T.G. Papaioannou and K. Aberer, "Cost-efficient and Differentiated Data Availability Guarantees in Data Clouds," *IEEE ICDE' 10*, pp. 980-983, Mar. 2010.

[109] K. Vishwanath and N. Nagappan, "Characterizing Cloud Computing Hardware Reliability" *Proceedings of the 1st ACM symposium on Cloud computing (SoCC '10),* 2010.

[110] R.E. Tarjan, "Dynamic Trees As Search Trees Via Euler Tours Applied to The Network Simplex Algorithm." *Mathematical Programming* 1997.

[111] R.K. Ahuja, T.L. Magnanti and J. B. Orlin, *Network Flows: Theory, Algorithms and Applications*, Prentice Hall, 1993.

[112] N. G. Duffield, et al, "A Flexible Model for Resource Management in Virtual Private Networks," *ACM SIGCOMM,* 1999.

[113] C. Clos, “A study of non-blocking switching networks”. *Bell System Technical Journal,* Mar. 1953.

[114] C. Chang, W. Chen and H. Huang, “Birkhoff-von Neumann Input-buffered Crossbar Switches for Guaranteed-rate Services,” *IEEE Transactions on Communications,* vol. 49, no. 7, pp.1145-1147, Jul. 2001.

[115] I. Keslassy, M. Kodialam, T. V. Lakshman and D. Stiliadis, “On Guaranteed Smooth Scheduling for Input-Queued Switches,” *IEEE/ACM Transactions on Networking* vol. 13, no. 6, pp. 1364-1375, Dec. 2005.

[116] B. Towles and W. J. Dally, “Guaranteed Scheduling for Switches with Configuration Overhead.” *IEEE/ACM Transactions on Networking* vol. 11, no. 5, pp. 835-847, Oct. 2003.

[117] B. Wu, K.L. Yeung, M. Hamdi and L. Xin, “Minimizing Internal Speedup for Performance Guaranteed Switches With Optical Fabrics,” *IEEE/ACM Transactions on Networking*, vol. 17, no. 2, pp. 632-645, Apr. 2009.

[118] M. Akbar, and et al., “Solving The Multidimensional Multiple-choice Knapsack Problem by Constructing Convex Hulls.” *Computers and Operations Research* vol. 33, no. 5, May 2006.

[119] R.Cole and J. Hopcroft, “On Edge Coloring Bipartite Graphs” *SIAM J. Comput.,* vol. 11, pp.540-546, 1982.

[120] Jonathan L. Gross, J. Yellen, “Graph Theory and Its Applications”, *CRC Press,* pp. 568, 2005.

[121] H. Wang and K. Bergman, “A Bidirectional 2×2 Photonic Network Building-Block for High-performance Data Centers,” *Optical Fiber Communication Conference and Exposition (OFC/NFOEC)* Mar. 2011.

[122] Z. Guo, Z. Zhang and Y. Yang, “Performance Modeling of Hybrid Optical Packet Switches with Shared Buffer, IEEE INFOCOM, 2011.

[123] Z. Guo and Y. Yang, “Pipelining Multicast Scheduling in All-Optical Packet Switches with Delay Guarantee, 23rd International Teletraffic Congress (ITC), Sept. 2011.

[124] Z. Guo and Y. Yang, “High-Speed Multicast Scheduling for All-Optical Packet Switches, IEEE NAS 13, July 2013.

[125] Z. Zhang, Z. Guo and Y. Yang, “Bounded-Reorder Packet Scheduling in Optical Cut-through Switch, IEEE INFOCOM, 2013.

[126] Z. Guo and Y. Yang, Low-Latency Multicast Scheduling Algorithm for All-Optical Interconnects, to appear in IEEE Transactions on Communications, 2014.

[127] Z. Guo and Y. Yang, "High Speed Multicast Scheduling in Hybrid Optical Packet Switches with Guaranteed Latency, IEEE Transactions on Computers. vol. 62, no.10, pp.1972-1987, Oct. 2013

[128] Z. Guo, Z. Zhang and Y. Yang, "Exploring Server Redundancy in Nonblocking Multicast Data Center Networks, IEEE INFOCOM, 2012.

[129] Z. Guo and Y. Yang, "On Nonblocking Multirate Multicast Fat-tree Data Center Networks with Server Redundancy, IEEE IPDPS, 2012.

[130] Z. Guo and Y. Yang, "Exploring Server Redundancy in Nonblocking Multicast Data Center Networks, to appear in IEEE Transactions on Computers.

[131] Z. Guo and Y. Yang, "On Nonblocking Multirate Multicast Fat-tree Data Center Networks with Server Redundancy, to appear in IEEE Transactions on Computers

[132] Z. Guo, J. Duan and Y. Yang, "Oversubscription Bounded Multicast Scheduling in Fat-tree Data Center Networks, IEEE IPDPS, 2013.

[133] Z. Guo and Y. Yang, "Multicast Fat-tree Data Center Networks with Bounded Link Oversubscription, IEEE INFOCOM Mini, 2013.

[134] Z. Guo, J. Duan and Y. Yang, "On-line Multicast Scheduling with Bounded Congestion in Fat-tree Data Center Networks, IEEE Journal on Selected Areas in Communications (J-SAC), vol. 32, no. 1, Jan. 2014.

[135] Z. Guo and Y. Yang, " Collaborative Network Configuration in Hybrid Electrical/optical Data Center Networks, IEEE IPDPS, 2014