A Novel Architecture, Topology, and Flow Control for Data Center Networks

Tingqiu Yuan

Thesis submitted to the University of Ottawa
in partial Fulfillment of the requirements for the
Doctorate in Philosophy degree in Electrical & Computer Engineering

School of Electrical Engineering and Computer Science
Faculty of Engineering
University of Ottawa

知之为知之，不知为不知，是知也。

"To know what you know and what you do not know, that's true knowledge."

–Confucius

Abstract

With the advent of new applications such as Cloud Computing, Blockchain, Big Data, and Machine Learning, modern data center network (DCN) architecture has been evolving to meet numerous challenging requirements such as scalability, agility, energy efficiency, and high performance. Among the new applications ones are expediting the convergence of high-performance computing and Data Centers. This convergence has prompted research into a single, converged data center architecture that unites computing, storage, and interconnect network in a synthetic system designed to reduce the total cost of ownership and result in greater efficiency and productivity. The interconnect network is a critical aspect of Data Centers, as it sets performance bounds and determines most of the total cost of ownership. The design of an interconnect network consists of three factors: topology, routing, and congestion control, and this thesis aims to satisfy the above challenging requirements.

To address the challenges noted above, the communication patterns for emerging applications are investigated, and it is shown that the dynamic and diverse traffic patterns (denoted as **∗-cast**), especially multi-cast, in-cast, broadcast (one-to-all), and all-to-all-cast, play a significant impact in the performance of emerging applications. Inspired by hypermesh topologies, this thesis presents a novel cost-efficient topology for large-scale Data Center Networks (DCNs), which is called HyperOXN. HyperOXN takes advantage of high-radix switch components leveraging state-of-the-art colorless wavelength division multiplexing technologies, effectively supports ∗-cast traffic, and at the same time meets the demands for high throughput, low latency, and lossless delivery. HyperOXN provides a non-blocking interconnect network with a relatively low overhead-cost. Through theoretical analysis, this thesis studies the topological properties of the proposed HyperOXN and compares it with other different types of interconnect networks such as Fat-Tree, Flattened Butterfly, and Hypercube-like topologies. Passive optical cross-connection networks are used in the HyperOXN topology, enabling economical, power-efficient,

and reliable communication within DCNs. It is shown that HyperOXN outperforms a comparable Fat-Tree topology in cost, throughput, power consumption and cabling under a variety of workload conditions.

A HyperOXN network provides multiple paths between the source and its destination to obtain high bandwidth and achieve fault tolerance. Inspired by a power-of-two-choices technique, a novel stochastic global congestion-aware load balancing algorithm, which can be used to achieve relatively optimal load balances amongst multiple shared paths is designed. It also guarantees low latency for short-lived mouse flows and high throughput for long-lasting elephant flows. Furthermore, the stability of the flow-scheduling algorithm is formally proven. Experimental results show that the algorithm successfully eliminated the interactions of the elephant and mouse DC flows, and ensured high network bandwidth utilization.

Acknowledgements

First and foremost, I wish to express my heartfelt gratitude to my supervisor, Prof. Dan Ionescu. He has been a great teacher who has prepared me to get to this place in my academic life. This Thesis would not be nearly as good without his help, support, patience and understanding.

Second, I would like to thank my friends, whose friendships have helped me a lot on some topics in the thesis. Especially I would like to honor the memory of my friend, Dongli Zhang, who had advised and encouraged me to study for the Ph.D. Program.

Finally, I would like to express my gratitude to my father, mother and other family members. My wife has been highly supportive of me throughout this entire process and has made countless sacrifices to help me get to this point. My son has made me stronger, better, and more fulfilled than I could have ever imagined. Without their tremendous understanding and encouragement in the past few years, it would be impossible for me to complete my studies. They always stay with me and support me no matter whether I succeed or fail. Thank you for your love!

Contents

List of Figures

List of Tables

Abbreviations

A2A	**A**ll-to-**A**ll
ABT	**A**ggregate **B**ottleneck **T**hroughtput
ADN	**A**pplication-**D**riven **N**etwork
AI	**A**rtifical **I**ntelligence
AIMD	**A**dditive **I**ncrease-**M**ultiplicative **D**ecrease
ALW	**A**lizadeh **W**eb **S**earch
AQM	**A**ctive **Q**ueue **M**anagement
ARED	**A**daptive **R**andom **E**arly **D**etection
AVQ	**A**daptive **V**irtual **Q**ueue
AWGR	**A**rrayed **W**aveguide **G**rating **R**outer
BB	**B**isection **B**andwidth
BBR	**B**ottleneck **B**andwidth and **R**ound-trip time
B&S	**B**roadcast-and-**S**elect
CAPEX	**CAP**ital **EX**penditure
CCC	**C**redit-based **C**ongestion **C**ontrol
CDR	**C**lock-**D**ata **R**ecovery
CMOS	**C**omplementary **M**etal-**O**xide-**S**emiconductor
CWDM	**C**oarse **W**avelength **D**ivision **M**ultiplexing
DC	**D**ata **C**enter
DCN	**D**ata **C**enter **N**etwork
DCQCN	**D**ata **C**enter **Q**uantized **C**ongestion **N**otification
DCTCP	**D**ata **C**enter **T**ransmission **C**ontrol **P**rotocol
DCIM	**D**ata **C**enter **I**nfrastructure **M**anagement
DML	**D**irectly **M**odulated **L**aser

DRB	**D**igit-**R**eversal **B**ouncing
DWDM	**D**ense **W**avelength **D**ivision **M**ultiplexing
ECMP	**E**qual **C**ost **M**ulti**p**ath
ECMP-VLB	**E**qual **C**ost **M**ulti**p**ath - **V**aliant **L**oad **B**alancing
ECN	**E**xplicit **C**ongestion **N**otification
ET	**E**nd-**T**erminal
FBB	**F**ull **B**isection **B**andwidth
FBFLY	**F**lattened **B**tter**fly** network
FC	**F**olded-**C**los network
FCFS	**F**irst **C**ome **F**irst **S**erved
FPR	**F**ree **P**ropagation **R**egion
FT	**F**at-**T**ree network
GE	**G**igabit **E**thernet
HC	**H**yper**c**ube network
HOLMES	**HOL**istic **M**ice-**E**lephants **S**tochastic scheduling algorithm
HOXN	**H**yper **O**ptical Cross-Connection **N**etwork
HPC	**H**igh-**P**erformance **C**omputing
HyperOXN	**H**yper **O**ptical Cross-Connection **N**etwork
HYB	ECMP-VLB **HYB**rid
IB	**I**nfini**B**and
IoT	**I**nternet **o**f **T**hings
IP	**I**nternet **P**rotocol
IT	**I**nformation **T**echnology
LA	**L**imiting **A**mplifier
LAN	**L**ocal **A**rea **N**etwork
LD	**L**aser **D**river
LM	**L**ongest **M**atching
MA	**M**ulticast **A**gent
MC	**M**onitor **C**ontroller
MPTCP	**M**ulti**p**ath **T**ransmission **C**ontrol **P**rotocol
MSS	**M**aximum **S**egment **S**ize

MTU	**M**aximum **T**ransmission **U**nit
NUM	**N**etwork **U**tility **M**aximization
OPEX	**OP**erational **EX**penditure
OSI	**O**pen **S**ystem **I**nterconnect
OXN	**O**ptical **Cross**-Connection **N**etwork
P2C	**P**ower of Two Random **C**hoices algorithm
PA	**P**re-**A**mplier
PAR	**Par**eto
PD	**P**hoto-**D**iode
PDU	**P**rotocol **D**ata **U**nit
PLTS	**P**acket-**L**evel **T**affic **S**plitting
PON	**P**assive **O**ptical **N**etwork
POSC	**P**assive **O**ptical **S**tar **C**oupler
PCDSDE	**P**oisson **C**ounter **D**riven **SDE**
PSD	**P**arallel **S**ignal **D**etection
PUE	**P**ower **U**sage **E**ffectiveness
QCN	**Q**uantized **C**ongestion **N**otification
QIC	**Q**uantum **I**mpedance **C**onversion
QoS	**Q**uality **o**f **S**ervice
RDMA	**R**emote **D**irect **M**emory **A**ccess
RED	**R**andom **E**arly **D**etection
RFC	**R**equest **F**or **C**omments
RM	**R**andom **M**atching
RTT	**R**ound-**T**rip **T**ime
Rx	**R**eceiver
SAN	**S**torage **A**rea **N**etwork
SE	**S**witch **E**lement
SLA	**S**ervice **L**evel **A**greement
SDDC	**S**oftware-**D**efined **D**ata **C**entre
SDE	**S**tochastic **D**ifferential **E**quation
SS	**S**low-**S**tart

SWAN	**S**oftware-driven **W**ide **A**rea **N**etwork
TCO	**T**otal **C**ost of **O**wnership
TOC	**TO**tal **C**ost
TCP	**T**ransmission **C**ontrol **P**rotocol
TIA	**T**rans-**I**mpedance **A**mplifier
TM	**T**raffic **M**atrix
ToR	**T**op **o**f **R**ack
TSO	**T**CP **S**egment **O**ffload
Tx	**T**ransmitter
UDP	**U**ser **D**atagram **P**rotocol
VCSEL	**V**ertical-**C**avity **S**urface-**E**mitting **L**aser
VLB	**V**aliant **L**oad **B**alancing
VM	**V**irtual **M**achine
WDM	**W**avelength **D**ivision **M**ultiplexing
WPCDSDE	**W**iener **PCDSDE**
WSDM	**W**avelength-**S**pace **D**ivision **M**ultiplexing
XOR	e**X**clusive **OR**

Dedicated to my family and friends...

Chapter 1

Introduction

1.1 Motivation

The Internet of Things [21] is creating new use cases and possibilities that were inconceivable just a few years ago. "Things" in the IoT can refer to a wide variety of devices. According to Gartner [22], there will be nearly 26 billion devices on the Internet of Things by 2020. The IoT enables billions of new connections between people, processes and things, and each of these connections can produce data [23]. Although the amount of annual global Data Center IP traffic will reach 17.1 Zbytes per year by 2020 as shown in Figure 1.1 based on data from Cisco [1] and IBM [2], total DCN energy efficiency will have to drop from 63 milliwatts per gigabit per second in 2012 to 1 milliwatt per gigabit per second in 2020. Thus, it is clear that much more energy-efficient Data Center interconnect networks have to be considered in order to sustain much higher network traffic but with a limited amount of power consumption.

New applications such as Big Data, Machine Learning, and Cloud Computing are recently emerging technologies that pose tremendous challenges on current Data Centers. Over the past few years, numerous DCN designs have been put forward to meet the ever-growing demand. However, the topology, routing, and flow control of a DCN should be fully addressed in all proposals. Topology of a

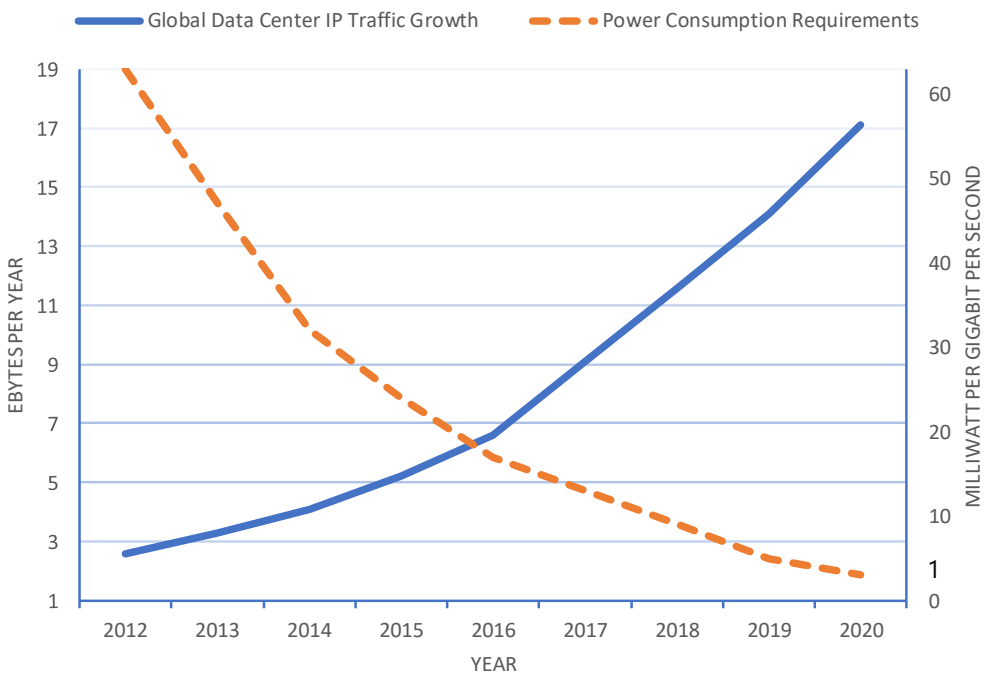


FIGURE 1.1: Global Data Center IP Traffic Growth and Power Consumption Requirements [1][2]

network refers to the interconnect of a shared set of nodes and channels. Once an interconnect network has been chosen, the routing scheme collects all the network states and determines a path to deliver messages. As the topology sets limits on the performance of the network such as throughput and delay, congestion control and load balancing algorithms strive to maximize the network utility including QoS, throughput, latency, etc.

The interconnect network is a crucial aspect of Data Centers that involves communication among numerous end-terminals. The topology for an interconnect network is critical because it sets performance bounds; in particular, latency and throughput.

There are three typical categories of existing DCN topologies as determined by the placement of routing intelligence: direct, indirect, and hybrid. Proposed direct-network topologies such as FBFLY [24], HyperX [6], and DCell [25], employ routing intelligence on End-Terminals (ETs) that host applications, such as servers. Every

switch has at least one server connected to it in a direct interconnect network(a.k.a. switch-centric). In contrast, in widely adopted existing topologies such as VL2 [26], Portland [27] and folded-Clos (a.k.a Fat-Tree [28]), ETs are restricted to the edges of the interconnect network. The routing intelligence in indirect-network topologies is placed on switches, some of which are not connected to any ETs and whose task is just to switch traffic. The third ones, hybrid topologies, combine the features of direct and indirect interconnect networks and provide trade-off designs like FSquare [29].

Commonly available topologies originate from Fat-Tree, which is an indirect topology. Although it guarantees non-blocking, as the network grows, due to the increased number of stages, the system aspects of cost, performance, and scalability (*i.e.*, maximum number of ETs a topology can support under a given dimension), degrade fast, which is not desirable for expanding DCN for real-time network applications. As a result, researchers have turned their eyes to direct topologies(*a.k.a.* server-centric) such as Hypercube [30] and Hypermesh [31]. By eliminating intermediate switch elements, every node in these topologies can connect to ETs, significantly improving system cost and topology diameter as well as system latency.

The emergence of optical technologies and components also provides for new DCN topologies [32]. Wavelength division multiplexing (WDM) supports tens or even hundreds of simultaneous optical channels in a single fiber, making the application of hyperedge topologies feasible for implementation. In addition, the availability of passive star couplers (PSCs) and arrayed waveguide grating routers (AWGRs) makes it feasible to implement optical switch-elements (SEs) in the topology, and miniaturization of optical transceivers and modulators remove the barrier for optical/electrical signal conversion in an optical element. Adoption of optical components not only reduces link complexity in the topology but also makes DCN more energy efficient in physical implementation due to the low power consumption of the optical solution.

The wide adoption of diverse cloud-based applications and services has exacerbated challenges in the design and operation of DCNs. Consider a typical mix of traffic patterns in cloud applications: mouse vs. elephant data flows. Mouse data flows are emails, web pages, data requests or any other short-lived data flows, whereas elephant data flows, on the other hand, are persistent data flows such as VM migrations, data migrations, MapReduce, and other application flows that impact network bandwidth for minutes or hours, if not longer. In the multi-tenant mode, long-lasting elephant and short-lived mouse flows share on DCN paths [33]. According to the results shown in [34], the sizes of numerous short-lived flows are usually less than 10KB, and the average load of these mouse flows is typically less than 5% [34] [35]. However, east-west traffic in the Data Center of between 75% and 95%, tends to require very low latency. On the other hand, the long-lasting heavy data flows in DCNs are typically much larger than 100KB, although the number of these large flows is extremely small compared to that of small flows [36] [37]. These elephant flows account for more than 80% of bandwidth in DCNs [35] [38]. Obviously, the competing performance requirements of the two types of flows make it challenging to schedule them on shared paths because the throughput-oriented large flows demand high bandwidth, while the latency-sensitive small flows prefer minimal queuing delays.

To provide high bisection bandwidth, DCN topologies are often multi-rooted; e.g., Fat-Tree, characterized by a large degree of multi-path networking and there are multiple routes between any two DCN endpoints [39] [26]. However, a critical issue in such network topologies is to design an efficient scheduling mechanism to balance the load among multiple available paths, while satisfying different application requirements defined in the SLAs.

Regarding to the scheduling of mouse-elephant hybrid DCN traffic, the novel scheduling scheme HOLMES [12] focuses on network partition solutions. Using a stochastic performance model, it is first theoretically proven that interference between mice and elephants is inevitable under the unified scheduler, indicating that network partitioning is a more appropriate solution for handling such hybrid DC traffic. As a result, HOLMES is proposed for such hybrid mouse-elephant

traffic in DCNs. HOLMES architecture partitions a DCN into high throughput and low-latency sub-networks, thereby decoupling the two competing scenarios and eliminating the interference in the hybrid traffic. HOLMES further deploys a stochastic and global congestion-aware load balancing algorithm that optimizes the performance of each sub-network. HOLMES^{+}, which is based on HOLMES, can also leverage local network information when a global network status is lacking.

1.2 Scope and Objectives

The dissertation is focused on the design of the architecture of the DCN and of the flow control algorithm to support dynamic and diverse $*$-cast traffic patterns [40] mixing together unicast, multicast, incast, and all-to-all-cast, provide a cost-effective high bandwidth topology, reduce power consumption, and ensure high network bandwidth utilization. *The diverse communication pattern, $*$-cast , mixes together unicast, multicast, incast, and all-to-all-cast* traffic. The proposed DCN topology HyperOXN and flow control algorithm HOLMES+ should satisfy the future requirements of Big Data and Machine Learning, in particular with regard to scalability, flexibility (*i.e.*, ability to increase network size), and high performance.

1.3 Achievements

Unlike existing topologies, which are inefficient for $*$-cast network traffic, HyperOXN is designed to inherently support $*$-cast communications leveraging state-of-the-art optical technologies. The main contribution of the thesis is based on the design of a next-generation data center network driven by cloud computing and big data applications.

To meet all the requirements of DCNs, the topology, routing, and flow control algorithm should be fully implemented. In this thesis, HyperOXN focuses on the topology and HOLMES^{+} addresses the flow control algorithm.

The contributions of the thesis are reflected in three aspects: the novel DCN topology, TCP modeling, and flow control algorithm as described in following sections.

1.3.1 HyperOXN - A Novel Topology for Next-Generation Data Centers

1. The combined merits of present direct topologies such as the bristled Hypercube and Hypermesh topologies and modified and improved to put forward a new topology called HyperOXN.

2. The HyperOXN topology is transformed into an equivalent topology, and the rearrangeable non-blocking property is demonstrated by mathematical induction.

HyperOXN has been studied in the following works:

- T. Yuan, D. Ionescu, R. Wang, and P. Li, "HyperOXN: A novel Data Center topology driven by machine learning," in 2018 APCA International Conference on Control and Soft Computing (CONTROLO), June 2018, pp. 573 - 578.

- T. Yuan, D. Ionescu, "HyperOXN: A Novel Topology for Next-Generation Data Centers," in DAAC'19 at Super Computing 2019.

1.3.2 A TCP Modeling based on SDE

1. A new formalism for traffic control is based on traffic classification by taking into account the number and size of packets, splitting DC traffic into two major traffic classes, i.e., mouse and elephant flows.

2. This will be used in the design of the DC hardware, such as optical switches in general and for the HyperOXN architecture.

1.3.3 HOLMES$^+$: Holistic Mouse-Elephant Stochastic Scheduling in Data Center Networks

1. The necessity of applying network partition solutions to mice-elephants DC traffic has been proven in a closed mathematical form which shows that unified flow scheduling schemes or congestion control mechanisms are not optimal. The HOLMES [41] [42] architecture is designed such that it partitions all the DCN paths into sub-networks, isolating the paths of elephant and mouse flows, and as a result, HOLMES eliminates the interference between the two. Moreover, the AI modules of the HOLMES architecture enables machine learning techniques in order to offer a more comprehensive analysis of the DCN status for better scheduling. HOLMES$^+$ is based on the HOLMES[41] [42] architecture, but enhances load-balancing and end-to-end path congestion detection by using local network information.

2. A novel stochastic load-balancing algorithm for HOLMES$^+$ is proposed which is global congestion aware. Using an end-to-end congestion signal, the algorithm is adaptive to link or node failures in DCN. Moreover, unlike earlier approaches such as CONGA [43] and Hedera [44], HOLMES$^+$ uses only limited global and local congestion information to improve the scheduling efficiency and sustain stability in all possible scenarios, especially for large-scale networks.

3. An analytical model is used to evaluate the adaptability to heterogeneous DCNs for HOLMES$^+$, as in [42]. The adaptability is confirmed with detailed simulations under various DCN architectures and scheduling policies. These experimental results can further guide the hardware implementation of HOLMES$^+$.

The SDE modeling and HOLMES$^+$ have been studied in the following work:

- T. Yuan, D. Ionescu, "HyperOXN: Topology and Scheduling for Efficient Large-Scale Network" in HPCC 2021(High Performance Computing and Communications).

1.4 Structure of the Thesis

The organization of this thesis is as follows. Chapter 2 first analyzes the workload related to Big Data and Cloud Computing, regarding which it can be concluded that the network requirements include measures for $*$-cast, low latency, agility, and low power consumption. Chapter 2 also introduces preliminaries that will be used throughout the thesis, and provides a literature review on DCN architecture and flow scheduling algorithms. Based on Hypermesh theory, Chapter 3 presents a novel topology for next-generation Data Centers, HyperOXN, which is evaluated at the flow-level in Chapter 4. In Chapter 5, the HOLMES$^+$ flow scheduling algorithm is introduced. As a traffic input to the HyperOXN architecture for simulations, Chapter 6 will introduce a new formalism for traffic control based on traffic classification. Performance evaluation at the packet-level for leveraging both HyperOXN and HOLMES$^+$ is studied in Chapter 7, and finally, conclusions and future work are presented in Chapter 8.

Chapter 2

Background and Preliminaries

This chapter serves as a literature review for the DCN architecture. The required background information is provided, and relevant works are cited and commented on.

2.1 Requirement Analysis

Emerging applications such as Big Data, Machine Learning and Cloud Computing pose significant challenges for current DCNs. The Data Center architecture has to meet numerous challenging requirements such as low latency, scalability, agility, energy efficiency, diverse traffic workloads and high performance. A good understanding of the requirements for the above emerging applications can guide the design of an interconnect network.

2.1.1 Workload analysis for Big Data

Distributed and Parallel Computing technology has evolved to distributed sharing memory or disk, with the DC interconnect network playing an important role.

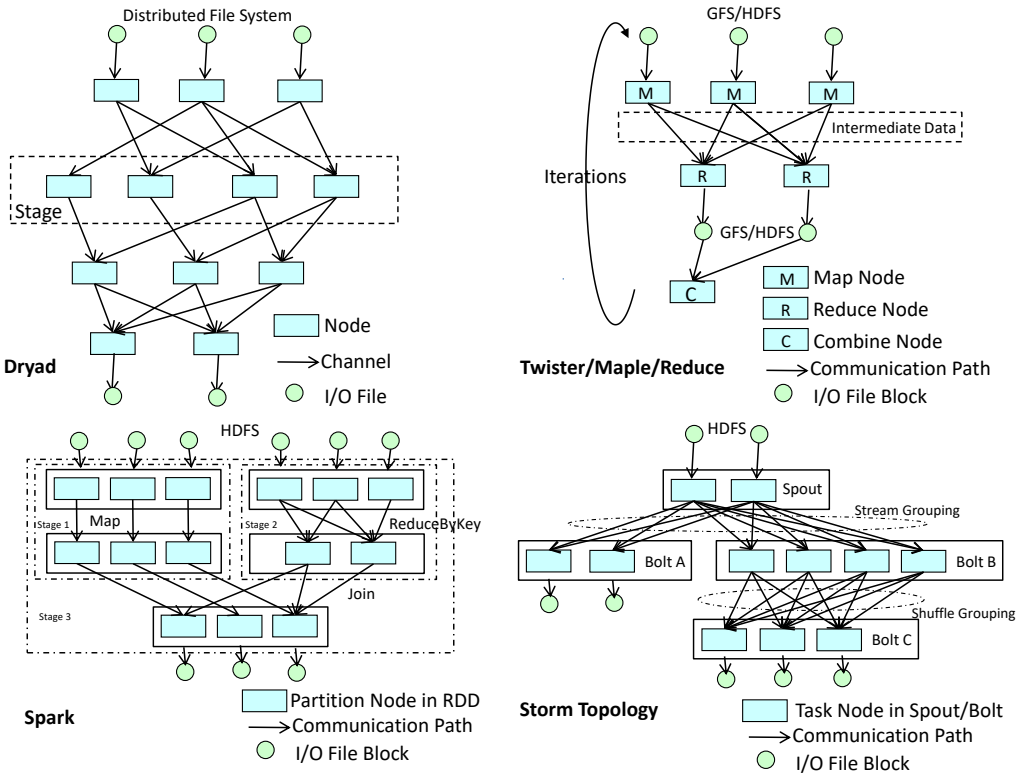


FIGURE 2.1: Data flows for big data algorithms

The emergence of big data processing and machine learning has motivated a new focus on data-centric designs, and big data brings unprecedented opportunities for reshaping the architecture of the Internet and Data Centers.

The term "big data" was first introduced by Mashey [45]. This article is a significant first step in highlighting this paradigm shift in order to address future DCN design challenges. A data-driven approach implies that current massive data and powerful computing capabilities are main reason for transforming network architectures, communication models, and resource management. Centralized network control such as in SDN (detailed in Section 2.1.2, Page 11), replaces distributed subsystems, and emerging communication models such as content-centric networks [46] compensate for inadequate network bandwidth with computational intelligence.

Figure 2.1 shows data flows for parallel and distributed big data algorithms including MapReduce [47], Spark [48], Storm [49], and Dryad [50]. The big data

algorithms mentioned above have high-demand on **∗-cast** traffic communication. Multicast, broadcast(one-to-all) and all-to-all-cast, have a significant impact on the performance of deep machine learning [51] as well.

2.1.2 Cloud Computing and SDN

Cloud computing is being rapidly adopted across the IT industry, driven by the need to reduce the total cost of ownership of increasingly more demanding workloads. Within companies, private clouds are offering a more efficient way to manage and use private Data Centers, while in the broader marketplace, public clouds offer the promise of buying computing capabilities based on a utility model.

The main goal of cloud computing is to make better use of distributed resources by combining them to achieve higher throughput and for the ability to solve large-scale computation problems. Cloud computing deals with virtualization, scalability, interoperability, quality of service and the delivery models of the cloud itself, namely private, public or hybrid.

Data Centers are compulsory parts of cloud computing. A modular Data Center is usually built with shipping-containers that encapsulate thousands of servers. However, Data Centers providing large-scale cloud services are large modular server farms.

Software Defined Networking(SDN) is considered a solution for the control of very large-scale server farms, though the industry is still shy about implementing it. This architectural approach proposed by McKeown [52], decouples the network control and forwarding functions making network devices easily programmable. Using the central controller, based on a global view of the network topology, the network can be dynamically adjusted together with the traffic flow based on traffic engineering (TE) algorithms. The objective for TE is to optimize a network's utility by balancing the flows, considering efficiency and fairness. TE is a function of the SDN controller, whose goal is to allocate network resources for network utility maximization.

Inter Data Center	9%
Data Center to users	14%
Intra Data Center	77%

TABLE 2.1: Data Center traffic distribution by destination [1]

2.1.3 Network Requirements for Next-Generation DCNs

Next-generation Data Centers have to meet the network requirements of emerging applications such as big data, machine learning, and cloud computing.

The traffic in the traditional Data Center network, as shown in Figure 2.2, flows in three directions [1]:

1. "North-South" traffic, which is limited to traffic that enters and exits the DC. It is the sort of traffic that crosses the DC boundary;
2. "East-West" traffic, which flows between DC devices and applications and never leaves the DC; and
3. "Inter-DC" traffic, which flows between multiple DCs, and also between DCs and the private/public cloud.

"East-West" traffic is primarily composed of communication between applications hosted on physical and virtual machines, as well as VM to VM interactions within the DC (such as Big Data applications which requires coordination among VMs). "North-South" traffic is primarily composed of traffic that enters and exits the DC and includes queries, commands, and specific data being either retrieved or stored (such as video streaming to mobile device, web searches, etc.).

Cisco's Global Cloud Index [1], given in Table 2.1, shows that unlike in campus networks, the dominant volume of traffic in a DC traverses in an "East-West" direction (76%), followed by "North-South" traffic (17%), and finally, inter-DC traffic, which currently comprises only 7% but is gradually growing. In campus networks, traffic is primarily 90% "North-South" traffic.

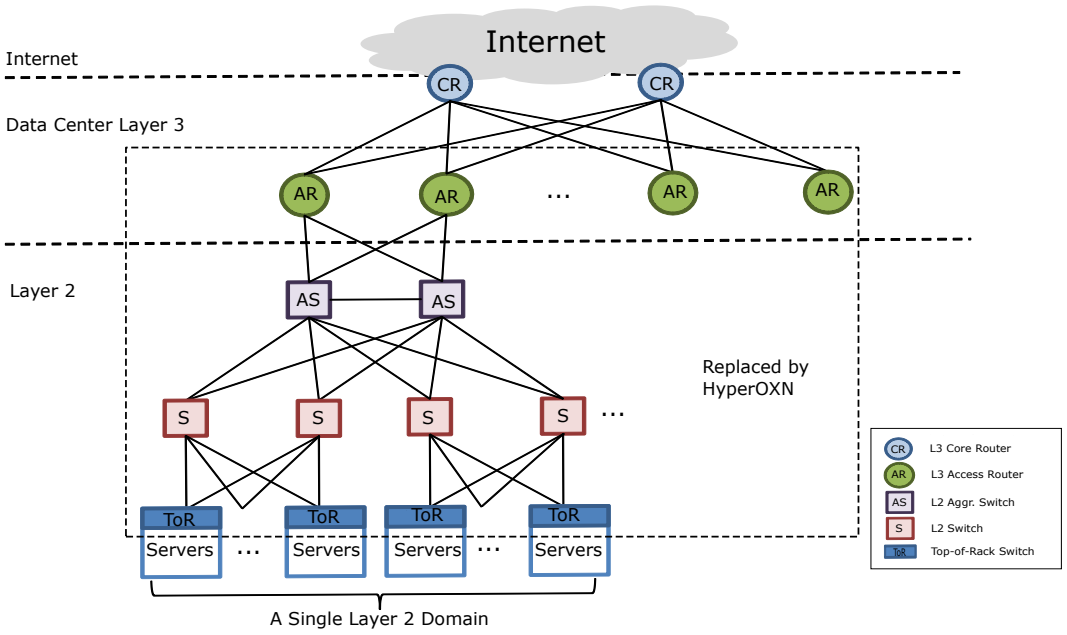


FIGURE 2.2: The architecture of traditional DCN

In the past, network design considered only the aggregated traffic case in which the total traffic from access nodes requires several times as much bandwidth as that in aggregate and core networks. Usually, oversubscription, denoted by ϕ (see details in Section 2.2.3, page 19), is less than 1/2. With the changes in traffic patterns, architectures such as FBFLY [24], DCell [25] and DragonFly [53], can not satisfy the new requirements, and non-blocking networks like Fat-Tree must be deployed in the Data Center internally to switch traffic at the line rate. DCNs are presently being re-architected as part of a transition to the next generation of Data Centers by considering big data applications and SDN technology. This change extends from power and cooling to servers and storage, as well as networking.

The push to rethink how networks support Data Centers is being driven by five key requirements:

1. ∗**-cast.** With the growth of the internet, there are increasing requirements for all kinds of cast technologies, including multicast, broadcast, etc. Multicast and broadcast can be used for data distribution, video play, parallel-computing, and distributed file system.

2. **Low latency.** Online real-time systems such as online games, meetings, bidding advertisements, social networking, file sharing, etc., all need low latency. Amazon found that every 100ms of latency cost them 1% in sales [54], and Google found that an extra 0.5 seconds in search page generation time reduced traffic by 20% [55]. A broker could lose $4 million in revenues per millisecond if their electronic trading platform was five milliseconds behind the competition [56]. With more and more applications moving to the cloud, there are more requirements for low latency.

3. **Agility.** Any server can be dynamically assigned to any service anywhere in the Data Center while maintaining proper security and performance isolation between services. If any server can become part of any service, then it is important that services are sufficiently isolated from each other that one service cannot impact the performance and availability of another. Under virtualized cloud computing environments, tremendous computing power and data are distributed in many DCs.

   A Data Center network should support the incremental growth of network size, *i.e.*, adding servers and network bandwidth incrementally to the DCN without destroying the current topology or replacing the current switches. Routing and forwarding in a DCN should rely on small forwarding states (tables) of switches and be scalable to large networks.

4. **Non-blocking.** As devices and storage systems require microbursts of up to 40Gbps, the need for a non-blocking switching architecture in Data Centers becomes critical to predictable application behavior and user satisfaction. The average speed of servers may still float below 1Gbps in most Data Centers, but traffic engineering for averages will affect application performance. Server network connections are universally moving to 10/25/100GE network interface cards in the next few years.

5. **Green.** The carbon emissions and overall environmental impact of Data Centers are becoming a greater concern for technology and IT companies.

The growing demand for Data Centers has caused the atmospheric emissions of these centers to skyrocket. Studies show that Data Center power requirements are increasing by 8% per year on average, and 20% per year in the largest Data Centers. Indeed, creating an energy-efficient Data Center is paramount to curbing runaway power consumption and accommodating greater Data Center capacity. It is worth noting that power and cooling are not being improved at "Moore's law" rates. As shown in Table 2.2, the projected allowable power consumption of networking equipment (including ToR, aggregate, and core switches) will have only a two-fold increase every four years from 0.5 (2012) to 2 MW (2020) while network traffic is expected to grow from 1 to 400 Pbytes/s. However, this will require a significant improvement in power efficiency from 13.5 mW/Gb/s (*i.e.*, 13.5 pJ/bit) to 2 pJ/bit [57].

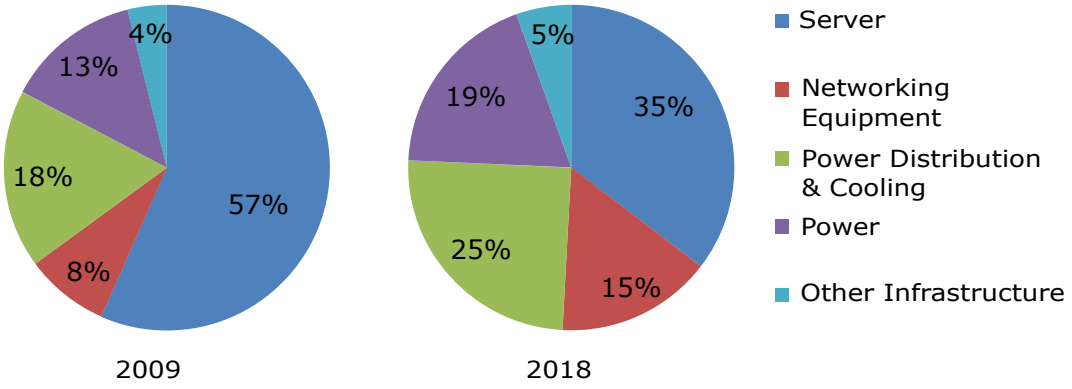


FIGURE 2.3: Monthly costs for 3-year server and 10-year infrastructure amortization in DCNs

As discussed in [58], in terms of monthly costs for 3-year server and 10-year infrastructure amortization in modern large-scale Data Centers, servers dominate overall costs to be around 65% as illustrated in Figure 2.3. For a more detailed exposition, see [58].

Our motivation is to propose a practical, faster, and more energy-efficient architecture for Data Centers to meet requirements such as $*$-cast, low latency, agility, non-blocking, and low power consumption. It is noteworthy that the network in

Feature	2012	2016	2020
Bidirection-bandwidth	1 Pbytes/s	20 Pbytes/s	400 Pbytes/s
Overall power consumption	5 MW	10 MW	20 MW
Network power consumption	0.5 MW	1 MW	2 MW

TABLE 2.2: Data Center bandwidth and power consumption projections [20]

the dashed box shown in Figure 2.2 can be replaced by HyperOXN (see details in Chapter 3, Page 54), a novel topology for next-generation Data Centers.

2.2 Preliminaries

2.2.1 Graph Operations

A network topology can be described by an undirected graph $G(V, E)$, where V represents a set of N nodes and E is a set of L links denoted by $(v1, v2)$, $v1$ and $v2 \in V$. The directed edges in a graph are network links and the vertices of the graph are network nodes (switches, routers, servers). The ***distance*** between two vertices in a graph is the length of the shortest path between them, and the ***diameter*** of an interconnect topology is the maximum distance among all vertex-pairs. The ***degree*** of a vertex is the number of edges incident to it. Each vertex has the same degree in a ***regular*** graph, which means that each node can be implemented by the same switches with an identical number of communication ports. This ***symmetric*** topology looks the same from each vertex. Besides *regularity* and *symmetry*, ***flexibility***, which is the ability to increase network size, is also a metric used to evaluate topologies.

There are some basic topologies such as *bipartite*, *fully-connected*(*i.e.*, *cliques*), *mesh*, *rings*, *stars*, *trees*, *bipartite* and *Caylay* [59] graphs. These graphs can be used to construct large complex networks by graph operations including ***overlay***, ***join***, ***recursive***, ***product***, ***compound*** and ***composition*** [60]. Kowalski [61] introduced graph ***lifting*** to construct large expander topologies. Among all the above graph operations the recursive, product, compound and lifting ones

have been massively implemented to facilitate the design of many state-of-the-art architectures for today's high performance DCNs. In general, the performance of a topology is heavily dependent on the basic graph chosen to construct large network and related operations.

1. **Recursive**: Many known large networks are constructed by recursive graph operation; for example, a n-ary d-level Fat-Tree ($n = 4$ ports for each switch and $d = 3$ layers in Figure 2.8, see details in Section 2.3.1.1, page 24), which can be obtained from n $(n-1)$-ary d-level sub-Fat-Trees recursively. For a more detailed exposition, please refer to generalized Fat-Trees in [62]. Hypercube-like topologies such as Hypercube [30], Torus, FBFLY [24], HyperX [6], and BCube [63] are primarily recursive in nature. As such, recursive graph operations have already become popular fundamental designing tools for hybrid graphs.

2. **Product**: The *product* graph [60] $G(V, E) = G_2(V_2, E_2) \times G_1(V_1, E_1)$ of two graphs is defined as below. V is the Cartesian product $V_2 \times V_1 = \{(v_2, v_1) | v_2 \in V_2 \; and \; v_1 \in V_1\}$. The node number in G is a combination of two address fields with one each from the two basic graphs G_1 and G_2. Two vertices (x_2, x_1) and (y_2, y_1) of G are adjacent if and only if $x_1 = y_1$ and x_2, y_2 are adjacent in G_2, or $x_2 = y_2$ and x_1, y_1 are adjacent in G_1. Figure 2.4 shows an example of a product graph G from a triangle graph G2 and a line graph G2. All Hypercube-like graphs, such as Hypercube, Torus, FBFLY, BCube, and HyperX, fall into this category by making Caylay graphs as their basic construction blocks, as shown in Section 2.2.2.

3. **Compound**: The compound graph $G = G_2(G_1)$ is constructed by replacing each node of G_2 with a copy of G_1 and there is only one link between the two copies of G_1. Figure 2.5 shows an example of how to get the compound graph G, which combines the characteristics of both G1 and G2. Also large compound graphs can be designed *recursively* on the basic graphs. Several novel proposals for DCNs are designed using the compound graphs, e.g. DragonFly [53], DCell [25], MDCube [64], LongHop [65], etc.

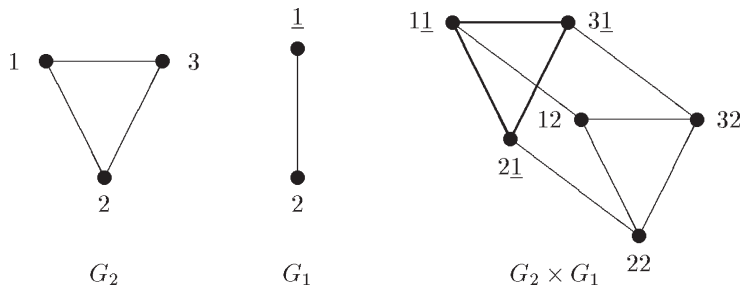


FIGURE 2.4: An example of product graph

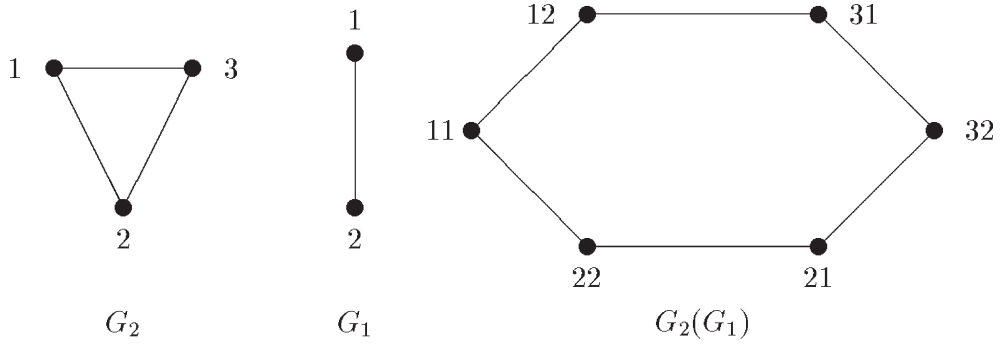


FIGURE 2.5: An example of compound graph

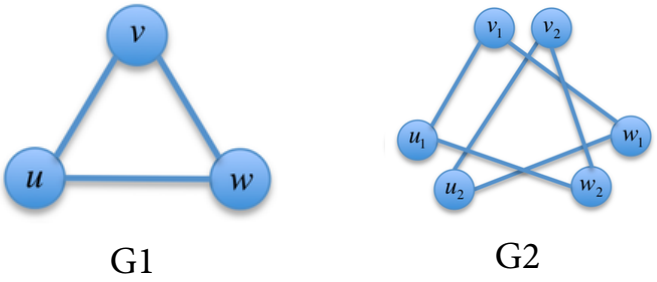


FIGURE 2.6: An example of lifting graph

4. **Lifting**: Figure 2.6 shows a *2-lifting* of the basic triangle graph G_1 to get an expander graph G_2 in two steps as follows: (1) Replace every vertex v in G_1 with two vertices v_1 and v_2, which have no any direct connections; (2) Insert two links between the two copies of u and v for each link (u, v). A large *k-lifing* can be constructed by a generalized 2-lifing. The large-scale interconnect Xpander[66], SlimFly [67] and Jellyfish [68] can be designed based on some basic graphs such as cliques and rings by *k-lifting*.

2.2.2 Cayley Graph

Cayley graphs [59] are often used for the design of interconnect network topologies. They have the merits of symmetry, self-routing, good scaling, low latencies and high bisections. Some well-known Cayley graphs are Hypercube, FBFLY, HyperX, BCube [63], MDCube [64] and DragonFly [53]. The topology looks the same from each vertex in a Cayley graph. HyperOXN is a type of Cayley graph as well. As such, the Cayley graph will be detailed as follows.

Let $\boldsymbol{S}$ be a nonempty subset of a finite group $\boldsymbol{G}$ with n elements $\{g_1, g_2, ..., g_n\}$ (see Group Algebra Theory in [69]). It is called $S = \{h_1, h_2, ..., h_m\}$ a generating set of $\boldsymbol{G}$ if both of the following hold: (1) the identity $\boldsymbol{I} \notin \boldsymbol{S}$, means no self-loops; and (2) $\boldsymbol{S}$ is closed under the inverse, $g_i^{-1} \in \boldsymbol{S}$ if and only if $g_i \in \boldsymbol{S}$, which means bi-directionality.

The Cayley graph $\boldsymbol{CG(V, E)}$ has vertex-set $\boldsymbol{V = G}$ and edge-set $\boldsymbol{E} = (g_i, g_i \cdot h_j)$, i=1, ..., n; j=1, ..., m. Each vertex in $\boldsymbol{CG(V, E)}$ is connected to m vertices.

A d-dimensional Cartesian product group $\boldsymbol{Z^d} = \boldsymbol{Z_{n_1}} \times \boldsymbol{Z_{n_2}} \times ... \times \boldsymbol{Z_{n_d}}$, where $\boldsymbol{Z_{n_d}}$ is a cyclic group with a set of integers $\{0, 1, ..., n_d - 1\}$. There are $|\boldsymbol{Z^d}|$ (*i.e.*, cardinality) $= n_1 * n_2 * \cdots * n_d$ elements in the group $\boldsymbol{Z^d}$, and each element has a unique integer address which can be viewed as d-dimensional coordinates, denoted as $(x_{d-1}, x_{d-2}, ..., x_1, x_0)$ with bit-wise XOR as the group operation, $x_i \in [0, n_i - 1]$ and n_i is the number of vertices in dimension i^{th}. Every neighbor vertex coordinate only differs in one dimension.

For example, let $\boldsymbol{Z^3} = \{000, 001, ..., 321\}$ with 24 vertices, $\boldsymbol{S_6} = \{001, 010, 020,$ 100,200, $300\}$, and then $\boldsymbol{CG(Z^3, S_6)}$ stands for a $4 * 3 * 2$ Generalized Hypercube as shown in Figure 2.7.

2.2.3 Bisection Bandwidth

Considering a bisectioning cut [3] that splits a network into two nearly equal sections $S1$ and $S2$, then $B(S1, S2)$ is the total bandwidth of edges crossing the

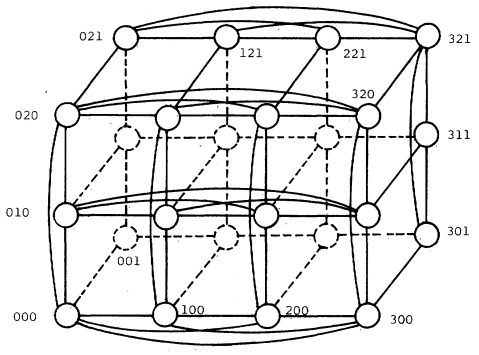


FIGURE 2.7: A $4 * 3 * 2$ Generalized Hypercube

two bisections. Figure 2.8 illustrates a bisectioning cut as a dotted vertical line. The bisection bandwidth of Graph G [3] is defined as the smallest B among all possible bisectioning cuts, that is,

$$BB(G) \equiv \min_{bisections} B(S1, S2) \quad [3] \tag{2.1}$$

The bisection bandwidth sets an upper-bound on the worst-case network throughput. Given that there are $T/2$ ETs in each bisectioning cut section, where T represents the number of all ETs, the oversubscription ϕ is defined as the ratio:

$$\phi \equiv \frac{BB}{T/2} = \frac{2 * BB}{T} \quad [3] \tag{2.2}$$

A network with $\phi = 1$ is said to have full bisection bandwidth (FBB) while all ETs in $S1$ have the capability to communicate with other ETs in $S2$ over its full link bandwidth, and no matter how all T nodes are partitioned in two sets of equal size, there are always $T/2$ channels going from one set to the other when $\phi = 1$.

Generally, $\phi = 1$ is a necessary condition for rearrangeable non-blocking, but it is not sufficient because there may be bottlenecks either in $S1$ or $S2$. The identical ϕ should be set when making performance comparison for different topology networks. As the merit of a Fat-Tree network is non-oversubscribed($\phi = 1$), it is unfair to compare it to others with different ϕ over-subscription ratio [24].

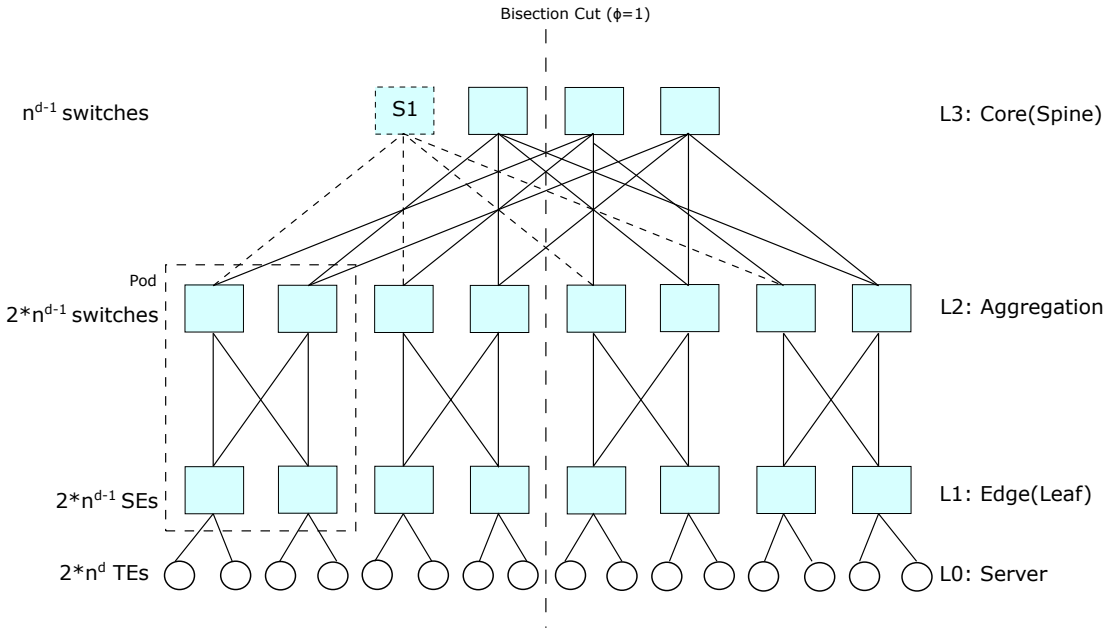


FIGURE 2.8: 4-ary 3-level Fat-Tree network

2.2.4 Non-blocking Network

It should be noted that the terms *message* and *packet* will be used throughout this thesis. A packet can be viewed as an atomic unit of data transfer from the end-hosts, *i.e.*, processors, to the network. A packet may be a complete message or part of a longer message. The network breaks an application message into parts of a certain size in bytes. These are the packets. In the latter case, a single message can be split into one or more packets at the interface to the network. Aside from user messages, each packet also contains a *header* that holds the destination address, plus possibly some additional information for routing, error control, flow control, and a payload body that carries the actual data. A typical packet contains perhaps 1,000 or 1,500 bytes.

The blocking/non-blocking characteristic is used to describe the connection capacity for the circuit-switched network. In a packet-switched network, there is no process to build and tear down the connections/calls in physical devices such as routers and switches, which exists in a circuit-switched network. Different packet flows share the channel or even all the network resources under some policies and rules, such as the bandwidth, buffers, etc., but non-blocking is still meaningful to

a packet-switched network because packet flows in a packet-switched network can interfere with each other under the constraints of network resource such as channel bandwidth and buffers.

Regarding a network with N inputs and N outputs, how many connections can be set up independent of the call/packet arrival order is a key indicator of network performance; it is defined as the maximum connection number C, that is to support C connections at the same time while keeping it non-blocking. It always has C edge-disjoint or no-conflict paths to route C calls/packets simultaneously. For example, a 4-ary 3-level Fat-Tree, as shown in Figure 2.8, has $C = 8$.

Blocking is defined as failure to satisfy a connection requirement, and it depends strongly on the combination properties of the switching networks. There are usually three kinds of non-blocking as follows. It is noteworthy that the non-blocking conditions for unicast demands are not equal to ones for multicast. A multicast connection is a connection where an input port can be simultaneously connected to more than one output port (but an output port can be connected at most to one input port at a time). A nonblocking multicast network is a multicast network which can realize all new multicast connections regardless of the current network state [70]. For an $N \times N$ network, a strictly non-blocking unicast interconnect, which can route all possible one-to-one input-output mappings(i.e., totally $N!$ permutation assignments), is still heavily blocking for multicast communications because the arbitrary multicast has N^N assignments.

1. **Strictly Non-blocking**: A switch is strictly non-blocking if a connection can always be set up between any idle input and output without the need to rearrange the paths taken by existing connections.

2. **Widesense Non-blocking**: A path can be set up between any idle input and any idle output without disturbing existing connections, provided that certain rules are followed. These rules prevent the network from entering a state for which new connections cannot be made.

3. **Rearrangeabley Non-blocking**: When establishing a path between an idle input and an idle output, paths of existing connections may have to be changed (rearranged) to setup that connection.

2.3 Overview of DC Network Architecture

As mentioned in [71], all Data Center topologies can be classified into three categories: ***direct(server-centric)***, ***indirect(switch-centric)*** and ***hybrid*** architectures. In switch-centric architectures, packet routing and forwarding are implemented using switches, whereas server-centric ones use servers for packet forwarding. Servers in server-centric architectures have dual functions: (1) to run applications, and (2) to act as routers to relay the traffic in the system. The third class, hybrid architectures, utilize both switches and servers for packet forwarding.

From the view of intrinsic graph operations(see Section 2.2.1, page 16), all those proposals can also be implemented by operations such as ***product***, ***recursive***, ***compound***, ***lifting***, etc. Both the basic graphs and operations can be used to construct complex large networks, and thus have significant impact on the performance of topologies.

In this section, a topology is classified as a ***static*** architecture if the interconnection is statically wired and cannot be altered dynamically; otherwise, it is called a ***dynamic*** network topology, which is flexible and reconfigurable and optimizable in real time. The dynamic ones tackle problems in which there are only a few hotspots (servers with heavy traffic) while most of the network is underutilized in large Data Centers.

2.3.1 Static Topologies: Recursive

2.3.1.1 Fat-Tree (a.k.a. Folded-Clos)

A Clos network [3] is designed to be a three-stage topology, with an ingress stage, a middle stage, and an egress stage, in which each stage consists of several crossbar switches. A multi-tiered Clos networking design with rearrangeable non-blocking is a traditional Data Center architecture. and is commonly used in many medium-to-large enterprises. A three-tiered topology (see Figure 2.9, page 25) contains core switches at the root level, aggregation switches at the middle level, and access level switches connected to the hosts.

A $G(m = 3, n = 3, r = 4)$ symmetric Clos network, depicted in Figure 2.9, has $r = 4 \times n \times m$ input switches, $m = 3 \times r \times r$ middle-stage switches, and $r = 4 \times m \times n$ output switches. All switches are crossbars. Masson [72] gave the sufficiency condition that a $v(m, n, r)$ Clos network is non-blocking for arbitrary uni-cast assignments if the number of middle stages switches is $m \geq 2n - 1$. Yang [70] showed that a $v(m, n, r)$ network is non-blocking for arbitrary multicast assignments if $m \geq c(n * r)$, where $c \approx 1$, and algorithms are proposed to reduce the number of middle switches to $m \geq 3(n - 1) * (\log r / \log \log r)$.

Fat-Tree topologies [28] have been dominant in the design of architectures for DCNs, and even for HPC with IB. The design of Fat-Tree network is motivated by the fact that the price differential between high-end switches (switches with higher link bandwidth or higher number of ports) and low-end switches is considerable.

The main idea behind the construction of a Fat-Tree topology is to replace the high-end switches in a multi-tiered topology by interconnecting several low-end switches. A Fat-Tree is a folded-Clos interconnection which can be constructed recursively. The main difference in the Fat-Tree is that all of the aggregation-level and core-level switches are replaced with interconnections of a set of low-end switches. Each subset, called a *Pod*, comprises a set of aggregation switches and edge switches together as shown in Figure 2.8. As the number of uplinks and downlinks for each Pod are equal, the Fat-Tree has full bisection bandwidth.

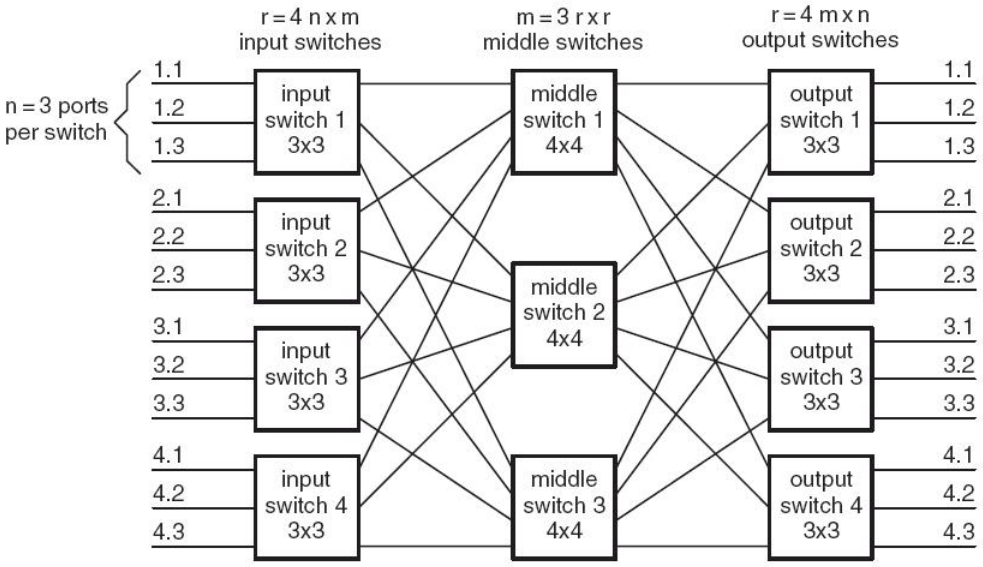


FIGURE 2.9: Clos interconnection [3]

Bisection bandwidth is the maximum bandwidth that can be transferred across the midpoint of the system. Moreover, since all the switches in the Fat-Tree topology are inexpensive low-end access level switches, this network topology is supposed to be highly scalable and economical.

A n-ary d-level Fat-Tree has: $(n/2)^{d-1}$ core switches; $2 \times (n/2)^{d-1}$ edge switches; $(2d-1)(n/2)^{d-1}$ switches in total; and $2 \times (n/2)^{d}$) servers. Each edge switch with n ports attaches $n/2$ servers and connects upper aggregation-level switches by the remaining ports. Using 48-port switches, a 3-level Fat-tree network can accommodate up to 27,648 servers.

Fat-Tree is also a rearrangeable non-blocking multi-path network in which there are many equal-cost paths between adjacent layers. This eliminates the bandwidth bottleneck in the core layer. Hence, Fat-Tree has appealing advantages over other direct networks and accomplishes the same performance for both benign and adversarial traffic. However, worse-case scenarios do not happen at any given times, which makes such a full-bisection bandwidth design seem wasteful and expensive.

A fundamental disadvantage of Fat-Tree is that the throughput drops dramatically when switch failure increases or an interconnect network is oversubscribed. With

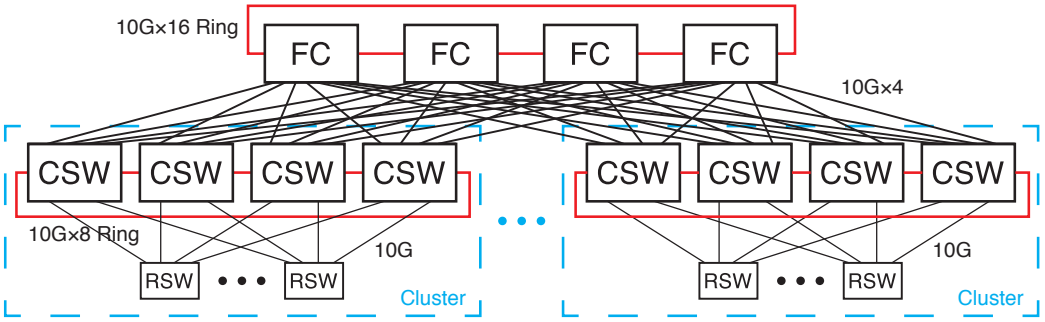


FIGURE 2.10: Facebook's pre-Altoona aggregated cluster design [4]

core switch $S1$ breakdown, shown in Figure 2.8, each server would get only 75% of the bandwidth if all four servers in one Pod communicate with ones in another Pod while only $2/n = 50\%$ servers are involved.

As mentioned above, in order to satisfy non-blocking multicast assignments, Fat-Tree has to increase the number of middle-stage switches from $m >= 2n - 1$ to $m >= c(n * r)$. In addition, servers only have a small number of network ports connected to switches in a switch-centric topology such that multicast communication cannot be supported well (which will be explained in Chapter 4).

Cisco FabricPath [43] is a fabric composed of multiple switches which form a flat two-layer Spine-Leaf Fat-Tree network.

Facebook [5] uses the same Fat-Tree architecture for DCN shown in Figure 2.11, evolving from previous aggregated cluster design [4], and Juniper networks [73] for Data Center evolution is also based on a multi-layer Folded-Clos.

Figure 2.10 shows Facebook's Data Center network before 2014. Each rack switch (RSW) has up to forty-four 10G downlinks to servers and four 10G uplinks, one for each cluster switch (CSW). A cluster consists of four CSWs and their connected RSWs. Each CSW has a 40G link to each of the four "FatCat" aggregation switches (FCs). An 80G protection ring connects the four CSWs within each cluster, and the four FCs are connected to a 160G protection ring. Most of the problems with this aggregated and oversubscribed network design stem from the necessity of very large switches for the CSWs and FCs. Any CSW or FC failure

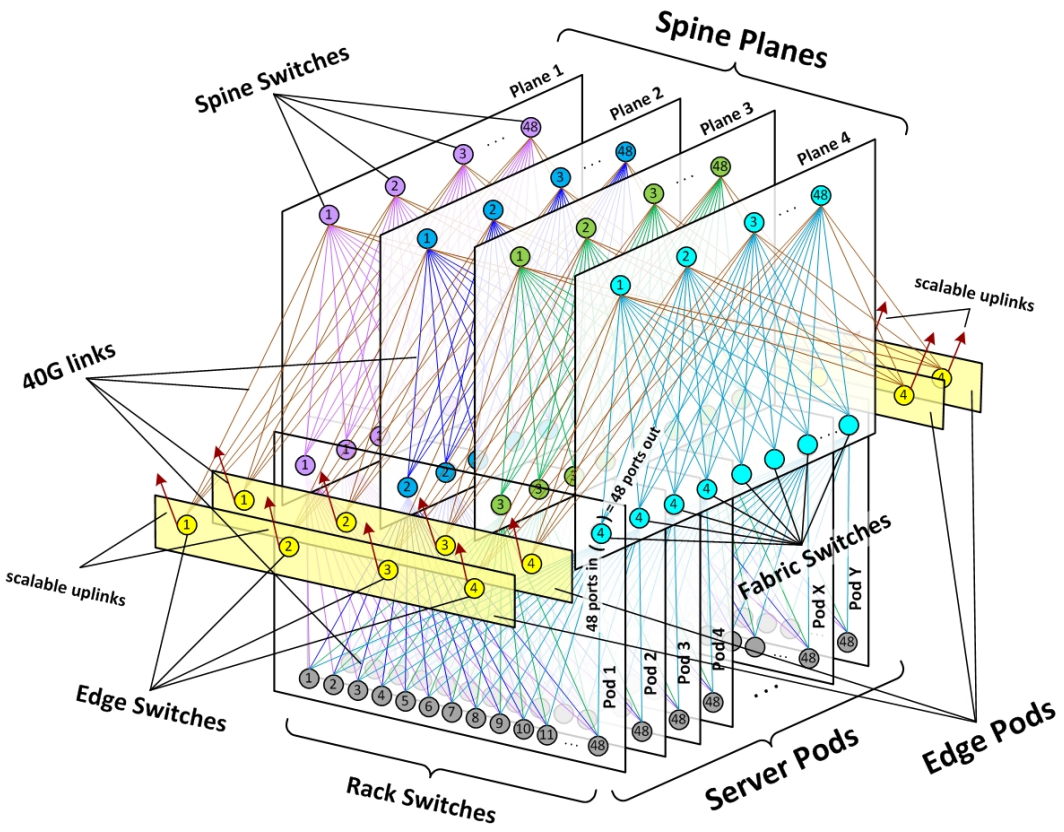


FIGURE 2.11: Facebook DCN architecture [5]

reduces inter/intra-cluster capacity to 75%. In addition, large switches lack the ability to scale out very rapidly and tend to have higher per-port CAPEX and OPEX.

To address the problems of the cluster architecture, Facebook proposes a next-generation DC architecture, shown in Figure 2.11, using a Fat-Tree topology, as shown in Figure 2.12.

Each Pod consists of 48 ToRs connected by four 40G uplinks to four fabric switches, each with forty-eight 40G downlinks and uplinks respectively. Each fabric switch is connected via forty-eight 40G uplinks to one spine plane with up to forty-eight spine switches, as shown in Figure 2.12. This fabric is non-blocking, and there is no oversubscription between the individual Pods. This highly modular design allows DCNs to quickly scale capacity in any dimension by adding more spine switches or edge Pods. Also the top layer no longer needs very large spine switches, which reduces the cost accordingly.

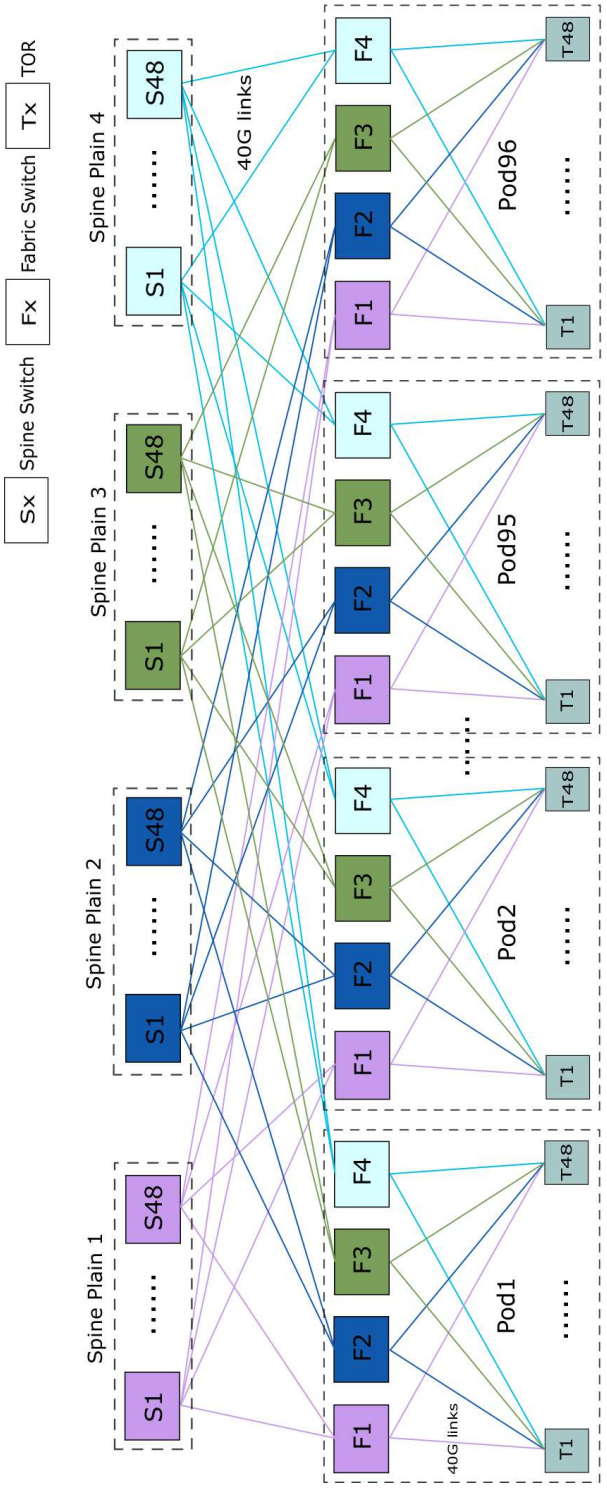


FIGURE 2.12: Facebook DCN architecture

2.3.1.2 VL2

To support non-blocking communication, the Fat-Tree [28] architecture requires a large number of switches in the core and aggregation layers as well as wiring to interconnect those switches, which increase deployment cost, energy consumption,

and management complexity. To solve this problem, VL2 [26] replaces Fat-tree's cheap core and aggregation switches with expensive high-speed electrical switches employing higher bandwidth links (40G or 100G) between the aggregate and core switches to reduce the amount of cables. VL2 adopts Valiant Load Balancing among all equal cost multiple paths at flow granularity and uses a resource management server to decouple server locations and IP addresses. Like Fat-Tree, VL2 is also based on a Clos topology and has the inherent weakness that the network throughput degrades dramatically as switch failure increases. In addition, the Clos topology's non-blocking design appears to be overkill for some traffic workloads, such as random workloads.

2.3.2 Static Topologies: Product Based on Cayley Graph

2.3.2.1 FBFLY

The FBFLY topology [24] was originally proposed for on-chip interconnect networks. The k-ary n-flat FBFLY is a multi-dimensional symmetric topology which takes advantage of high-radix switches [24] to create low-diameter networks. Here, n denotes the dimension of the topology and k is the number of switches in each dimension. Figure 2.13 shows a 4-ary 2-flat FBFLY. Each square in the figure represents a switch, and each of the eight switches interconnects with the other seven switches. In addition, each switch is linked with eight host nodes ($i.e.$, servers). A k-ary n-flat FBFLY is constructed from a k-ary $(n-1)$ FBFLY and a k-ary 2 FBFLY. For instance, an 8-ary 3-flat FBFLY can be constructed by copying the 8-ary 2-flat eight times, then interconnecting each switch in one group with the corresponding seven switches (one in each of the other seven groups). A k-ary n-flat FBFLY consists of k^{n-1} electrical switches.

Intrinsically, FBFLY can also be described as a generalized Hypercube or a regular HyperX [6] so that it can be constructed by product of Cayley graphs as mentioned in Section 2.2.2.

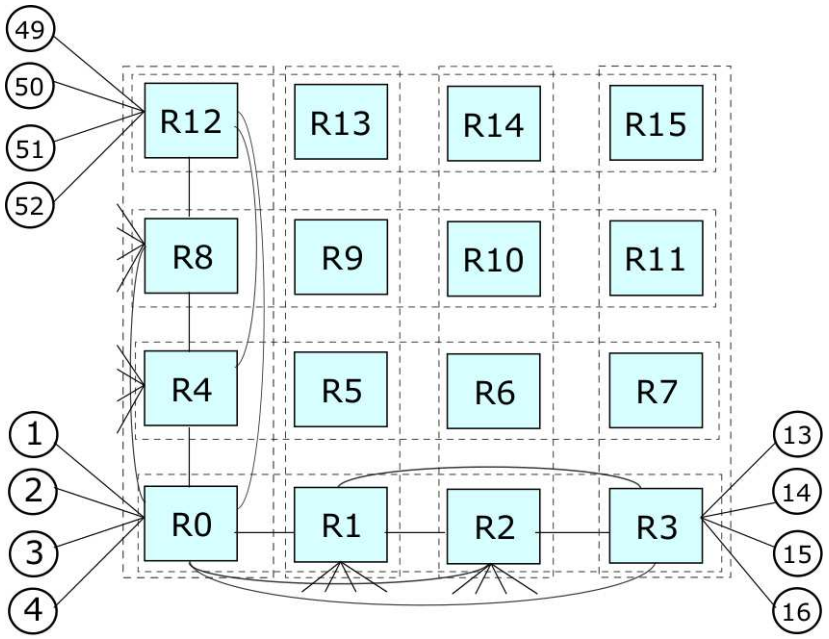


FIGURE 2.13: 4-ary 2-flat FBFLY structure

2.3.2.2 HyperX

A HyperX [6] is designed as a generalized FBFLY to optimize the low-cost configuration satisfying switch ports, the number of servers to support, and bisection bandwidth requirements. As such, HyperX is more flexible than FBFLY. Figure 2.14 illustrates a regular ($L = 2$, $S = 3$, $K = 1$, $T = 4$) HyperX, where each switch connects to T terminals, L represents the number of dimensions, and S and K denote the number of fully connected switches and the relative link bandwidths respectively in each dimension.

2.3.2.3 K-dilated K-way Bristled Hypercube

As a typical orthogonal topology, the Hypercube topology has been widely adopted in parallel computing systems thanks to its small diameter and lower cost. Without being able to fully prove or disapprove it, early research assumed Hypercube typology's non-blocking property and subsequent research by Liu [74] used an oblivious routing algorithm to prove its non-blocking property. Even though Hypercube topology is non-blocking, its scalability runs into limitation as DCN grows

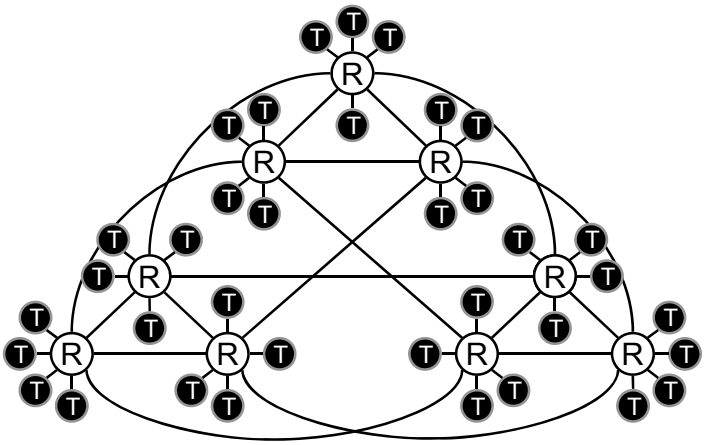


FIGURE 2.14: A regular (L = 2, S = 3, K = 1, T = 4) HyperX [6]. Dark circles (marked T) are terminals, light circles (marked R) are switches.

exponentially. Since high-radix switches have emerged, research on the high-radix network topology has made significant progress [6]. As an SE in Hypercube topology can only connect one ET, high radix switch advantages cannot be exploited in the Hypercube topology. As a result, researchers have proposed the k-dilated k-way bristled Hypercube [75]. Bristled topology allows each SE to connect to k ETs while $k > 1$, making it possible to connect to many more ETs without a change in radix. Through dilate SE, its bisection bandwidth accordingly increases k times making it non-blocking. With the above improvements, the bristled topology not only keeps the Hypercube topology's non-blocking property, but in the meantime increases the number of connected ETs k times. Figure 2.15 shows the topology of a 4-dilated 4-way bristled Hypercube.

2.3.2.4 K-ary Hypermesh

Hypermesh topology [76] is also an orthogonal multi-radix topology. It replaces all the links on the same radix with a **hyperedge** in order to make the topology simpler, reducing link complexity. In order to model a distributed 4×4 crossbar switch implemented by an optical passive star coupler, as shown in Figure 2.16(a), there are two traditional graphs that can be used: (1) a graph including 4 nodes and 1 switch node with a diameter of 2 and 4 edges, as illustrated in Figure

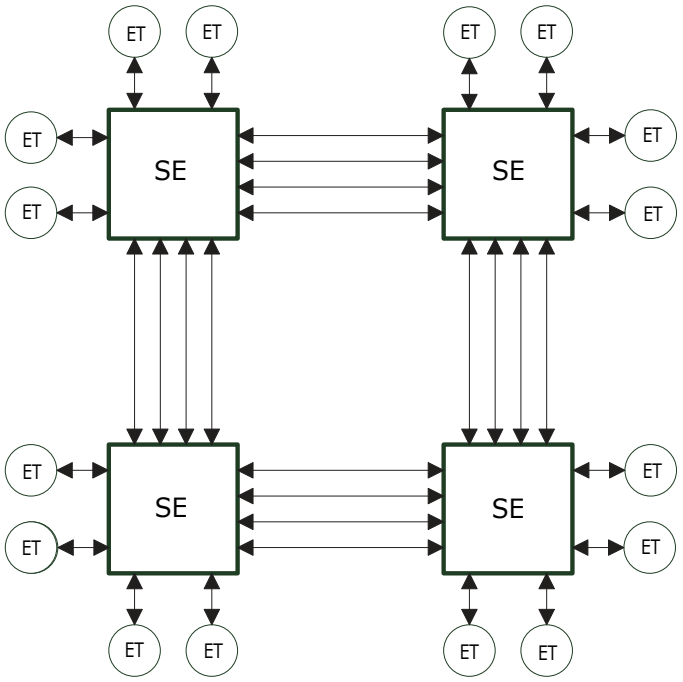


FIGURE 2.15: 4-dilated 4-way bristled Hypercube

2.16(b); and (2) a 4-nodes full-mesh graph with a diameter of 1 and 6 edges, as shown in Figure 2.16(c). However neither of them are accurate enough to model a real system in which all 4 nodes are neighbors with a diameter of 1. The first approach creates a virtual switch node which a real system does not have, and the second one has a BB larger (*i.e.*, 4) than 2, which it really is. As such, in order to reflect a real system, a 4-node hyperedge is introduced to model a distributed crossbar switch.

Szymanski [76] defined a dual-hypermesh as a hypermesh where each hyperedge can support two simultaneous transmissions from each member vertex and two simultaneous receptions into each member vertex.

The Generalized Hypercube topology [77] as developed from the Hypercube topology. Since there are only two SEs in every dimension of a Hypercube, the number of vertices grows significantly and the topology's dimensions increase fast, compromising its latency performance. By increasing the number of SEs in each dimension and using full mesh to connect all the SEs, the Generalized Hypercube topology

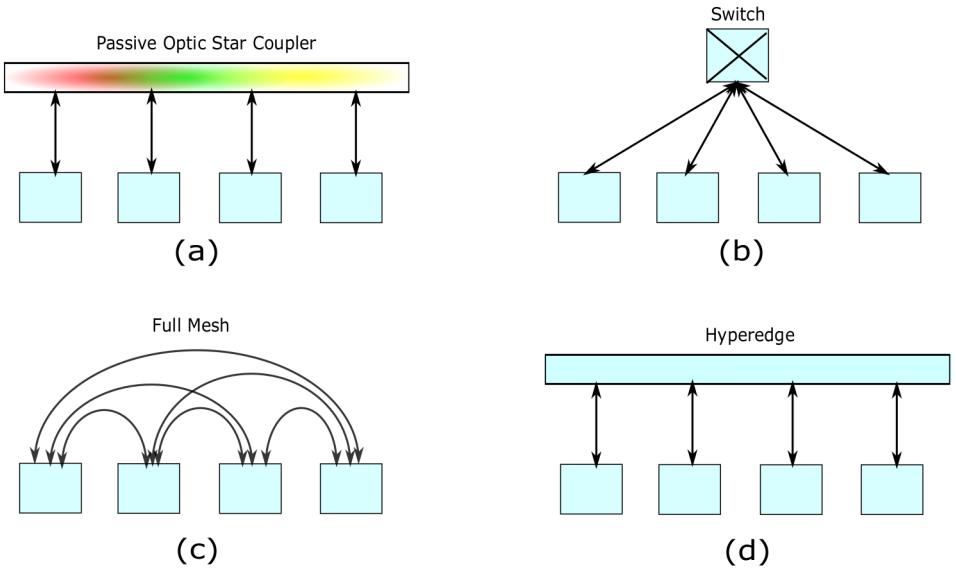


FIGURE 2.16: Model a distributed crossbar switch with a hyperedge

reduces the dimensions while keeping Hypercube's benefit, making it more suitable for larger scale DCNs.

The Hypermesh topology combined the advantage of the reduced dimensions of the Generalized Hypercube and low link complexity of hyperedge, making it the most attractive among all the topologies. In particular, the Hypermesh topology shown in Figure 2.17 is quite suitable for optical implementation [76].

2.3.2.5 CamCube

The CamCube [7] is a type of torus topology in which each server port is connected directly to six other servers. The topology of the CamCube is base on 3D Torus. Figure 2.18 depicts a 3-ary 3-cube CamCube.

2.3.2.6 DCell

DCell [25] uses servers with multiple network ports and mini-switches to construct its recursively defined architecture. The weakness of DCell is its low bisection bandwidth, which may cause traffic jams in the network.

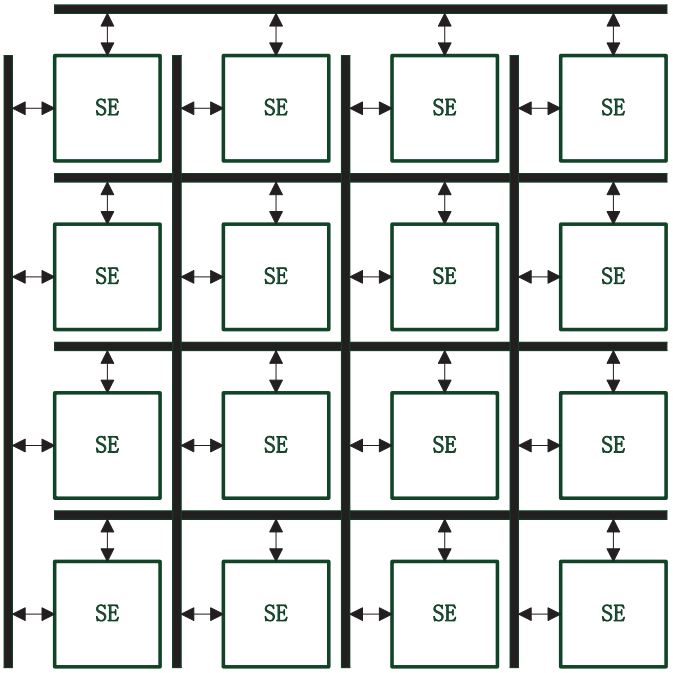


FIGURE 2.17: 4-ary Hypermesh

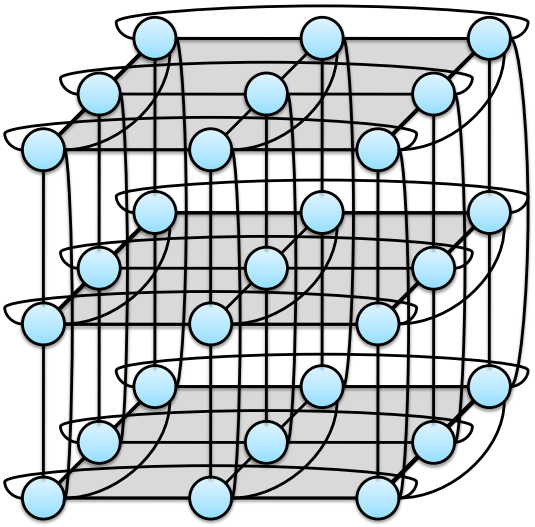


FIGURE 2.18: 3-ary 3-cube CamCube [7]

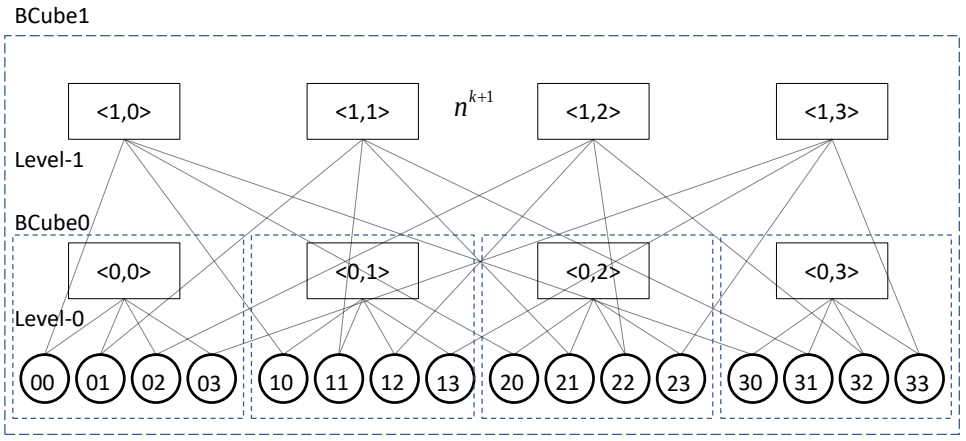


FIGURE 2.19: Bcube construction

2.3.2.7 BCube

To solve the traffic jams in DCell, Guo [63] proposed BCube, which provides more bandwidth in the top layer. In BCube, multi-port servers connect with multiple switches deployed in each layer. DCell and BCube can support extra-large-scale networks with more than a hundred thousand servers. The BCube architecture [63] uses both switches and servers for routing traffic. BCube k is recursively defined from BCube_0, which contains n servers and an n-port switch, and each server connecting to a switch port. There are no direct connections between any two servers. A BCube_k ($k \geq 1$) network can be constructed using n BCube_{k-1} topologies and n^k n-port switches. In each BCube_k, there are $k+1$ levels of switches, with n^k n-port switches in each level. Then, the number of servers in a BCube_k is

$$N = n^{k+1},\ k \geq 1\ [63].$$

Figure 2.19 shows an example of the BCube architecture. The BCube structure is closely related to that of the generalized Hypercube.

2.3.2.8 Long Hop Networks

Long Hop Networks [65] are designed from Cayley graphs to maximize the bi-section bandwidth based on optimizing the codeword distance for linear error correction codes. Although Long Hop Networks can achieve optimal throughput, the interconnections between every two long-hops are complicated, which hinders deployment.

2.3.3 Static Topologies: Compound

2.3.3.1 DragonFly

The topology DragonFly [53] inside each group can be any topology. Several groups are connected together using all to all links in a full-mesh fashion(a.k.a. compound of both graphs, one of them is a full-mesh); *i.e.*, each group has at least one direct link to every other group. In order to lower the cost, DragonFly assumes that the intra-group cables are optical, but that the cables inside the group are electrical. This aims to reduce the number of long links and the network diameter.

2.3.3.2 DCube

Guo proposed an expansible network structure for Data Centers called the DCube using hierarchical compound graphs [78]. The DCube consists of one or multiple interconnected sub-networks, each of which is a compound graph made by interconnecting a certain number of basic building blocks using a Hypercube-like graph.

2.3.4 Static Topologies: Lifting

Expander topologies built through lifting graph operations, such as recent the Jellyfish [68] and Xpander [66], interconnect edge switches directly by static links.

Unlike Fat-Trees, they adopt a flat structure without breaking the network into layers. Expander topologies can achieve near-optimal performance compared with all other static ones.

2.3.4.1 Jellyfish

Jellyfish is constructed based on a random graph, which hinders its deployment.

2.3.4.2 Xpander

Valadarsky [66] proposed Xpander based on lifting graph operations, which provides a deterministic and symmetric structure, but its topologies are hard to use for transparent incremental deployments without rewiring. Kassing [19] employed a routing algorithm combining both ECMP and VLB for the Xpander architecure. However, ECMP has some fundamental challenges (mentioned in Chapter 5), and exponential numbers of potential path sets as illustrated in Section 5.3.1.2 if using congestion awareness to alleviate the disadvantages of ECMP. Also, whether Xpander can leverage state-of-the art optical technologies to reduce cost, power consumption and number of cables, is deferred for future research.

2.3.5 Dynamic Networks

To tackle only a few hotspots in Data Center networks, numerous proposals [8] [79] [80] [9] [81] identify the hotspots by online measurement, and schedule flows dynamically to alleviate them. In general, the dynamic topology part, *i.e.*, direct links between edge switches, is implemented by reconfigurable optical or wireless connection matrices shown in Figure 2.20, which have to take a few milliseconds to set up paths. As of high latency to get forwarding paths ready, it is not suitable for bursty mice-flows, and can schedule only elephant-flows and the fixed topology functions to deliver short-lived flows.

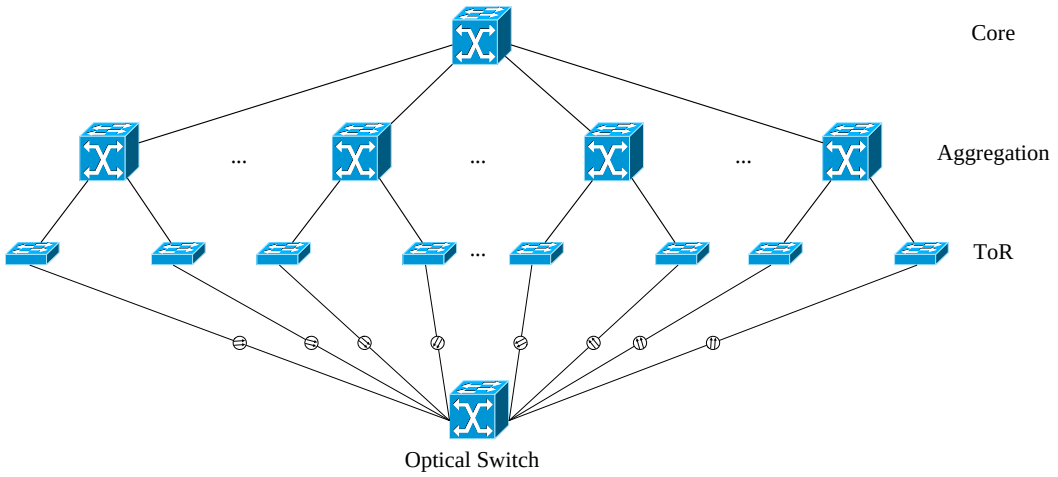


FIGURE 2.20: The Diagram for c-Through [8]

2.3.5.1 c-Through

To reduce networking costs, optical circuit switches are used in c-Through [8] to build a hybrid electrical/optical architecture that consists of two parts: a tree-based electrical fixed topology and a reconfigurable optical switching network which dynamically sets up connections between each pair of edge switches, as shown in Figure 2.20.

2.3.5.2 Helios

Helios [80] is another dynamic hybrid topology with both electrical and optical switches. Unlike c-Through, Helios delivers bursty mice-flows by electrical packet-switching interconnections and long-lived flows by the optical network. Each edge switches forwards packets in only one hop by optical connections in Helios and c-Through. However, both Helios and c-Through suffer from time delays of up to a few milliseconds because of circuit reconfiguring and limited connectivity among racks.

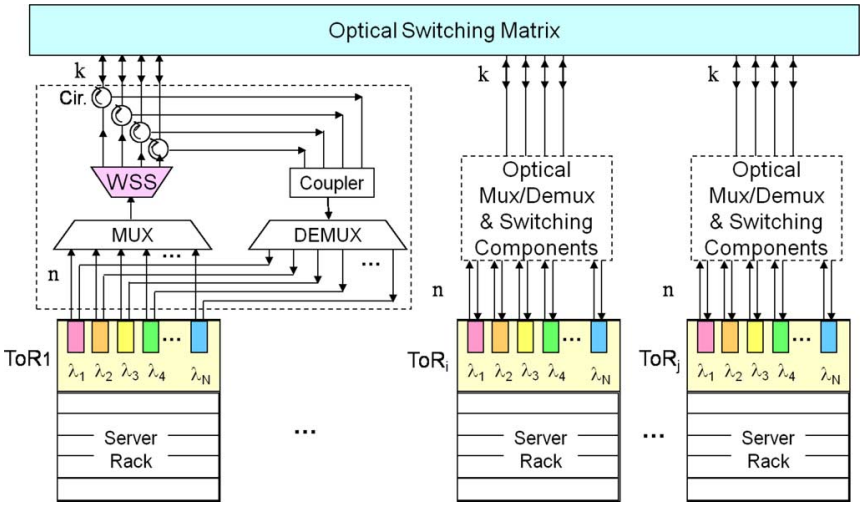


FIGURE 2.21: The OSA architecture [9]

2.3.5.3 OSA

OSA [9], shown in Figure 2.21, uses an optical switching network to interconnect electrical edge switches, but unlike Helios and c-Through, there is not any fixed electrical interconnect network in OSA, so packets have to be routed in multiple hops by transit ToRs. Also OSA suffers from bandwidth constraints between racks, and making it hard to implement in practice.

2.3.5.4 ProjecToR

ProjecToR [79] is a recent novel free-space optics based proposal using a digital micromirror device and mirror assembly combination as a transmitter and a photodetector on ToR as a receiver. Each ProjectToR switch provides free-space links independently based on only that ToR's local demand. While ProjectToR alleviates the hotspots problem as Helios does, it still involves numerous practical concerns on reliance under real DCN environments (*e.g.*, vibration and dust). Although ProjecTor largely reduces reconfiguration time, down to $12\mu s$, buffering is a nontrivial problem. It also faces challenges regarding the strategy of dynamical connections management.

2.3.5.5 RotorNet

Mellette [81] proposed a novel dynamic architecture, RotorNet, which dynamically reconfigures its constituent circuit switches by cycling through a series of pre-determined optical port matchings. It does not rely on any dynamic optimization in response to traffic estimation, which is totally different from the prevalent approaches including OSA, Helios, ProjectToR, etc.

2.3.5.6 MIMO-OFDM

Optical circuit switching networks can provide high throughput by 10G, 40G, or 100G links; however, for many DCNs, only a fraction of nodes may require high-bandwidth connectivity at any given time. To offer flexible bandwidth-sharing at a fine granularity, Ji [10] introduced a novel DCN architecture, shown in Figure 2.22, leveraging multiple-input multiple-output (MIMO) orthogonal frequency division multiplexing (OFDM) technology with parallel signal detection (PSD) and AWGR devices. In this architecture, each ToR, built by an array of directly modulated lasers (DMLs) and an OFDM modulator, can use the same wavelengths. The key insight in [10] is that only a single PD built at each ToR can receive all wavelength-division multiplexed channels concurrently by PSD technology, and accordingly at much lower cost for the receiving side. Further, the power consumption is largely reduced by using static and passive AWGRs. However, the expensive and challenging DMLs may put the scheme at a disadvantage.

2.3.5.7 Wireless

Wireless technology [82] is also used in DCNs without wired cables. In general, wireless DCNs adopt static topologies as mentioned above. Daniel [83] proposed adding extra multi-gigabit 60GHz wireless links, called “flyways”, to a wired Fat-Tree DCN between ToRs in order to relieve network congestion, and thus improve network performance. The wireless DCN falls into the category of dynamic networks to address the communication hotspots problem, instead of using optical

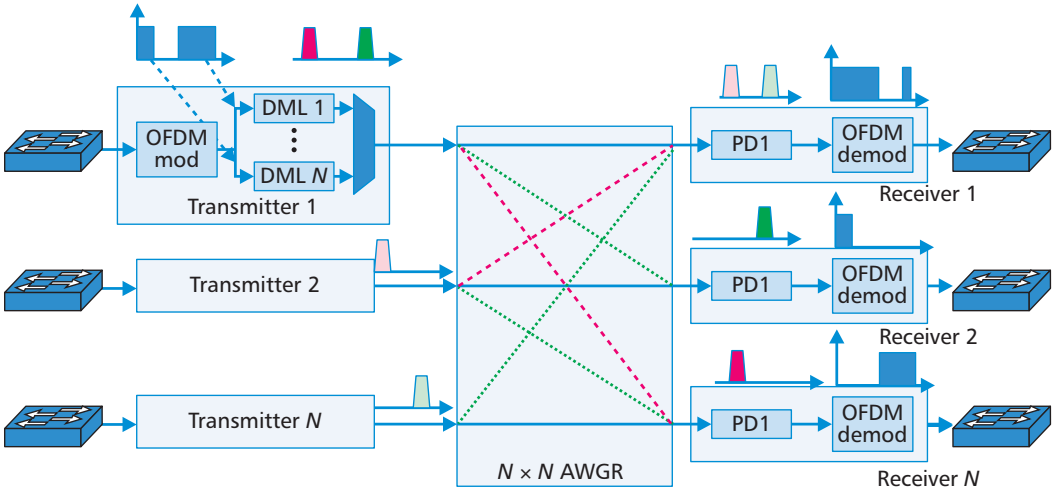


FIGURE 2.22: The optical MIMO OFDM-based architecture [10]

links. There are still a few concerns over wireless interference, link reliability, and limited bandwidth.

2.3.6 Topologies for HPC

HPC (High Performance Computing) platforms are traditionally tightly coupled and use frequent inter-processor communication and synchronization to perform huge parallel computations in a short time. High-performance computing demands very low latency or jitter, typically of the order of up to tens of microseconds. However, the emergence of big data analysis and machine learning has motivated a unified Data Center fabric that also supports high-performance computing, resulting in greater efficiency and productivity.

Cisco [84] proposed the third generation of Cisco Unified Fabric in order to support all applications by unifying LANs, SANs and high-performance computing networks. Amazon Elastic Compute Cloud (EC2) [85] has already enabled HPC in the cloud. Juniper [73] released Qfabric based on the Folded-Clos topology to deliver HPC applications.

In general, supercomputer systems like Cray XC30, IBM Blue Gene, etc., are built on IB or Ethernet technologies, and IB was more massively dominating ten years

ago, but a full 48% of the top 500 supercomputers [86] now use Ethernet as their standard networking technology.

From the most recent top 10 supercomputing systems in 2018 as shown in Table 2.3 (data from [86]), it is noted that Fat-Tree, Torus, and DragonFly, play important roles in the custom topologies for HPC.

Rank	System	Cores	Rmax(TFlop/s)	Topology
1	Summit-IBM Power System AC922	2,397,824	143,500	Fat-Tree/IB
2	Sierra-IBM Power System S922LC	1,572,480	94,640	Fat-Tree/IB
3	Sunway TaihuLigh SW26010	10,649,600	93,014.6	∼Fat-Tree/IB
4	Tianhe-2A	4,981,760	61,444.5	Fat-Tree
5	Piz Daint-Cray XC50	387,872	21,230	Dragonfly
6	Trinity-Cray XC40	979,072	20,158.7	Dragonfly
7	ABCI-PRIMERGY CX2570	391,680	19,880	Fat-Tree
8	SuperMUC-NG-ThinkSystem SD530	305,856	19,476.6	Fat-Tree
9	Titan-Cray XK7	560,640	17,590	3D-Torus
10	Sequoia-BlueGene/Q	1,572,864	17,173.2	5D-Torus

TABLE 2.3: Interconnections for top10 HPC systems in 2018

With the evolution of HPC applications such as Big Data and Machine Learning, the market share for **Fat-Tree** is approximately around 84%, and **Torus** and **DragonFly** occupy around 16%. As mentioned in Section 2.3.1.1, Fat-Tree topologies have been dominating in the design of architectures for HPC and DCNs as well.

2.4 Overview of Flow Scheduling Algorithms

The *de facto* DCN flow scheduling scheme ECMP [87] cannot meet the dynamic performance requirements for Data Centers. Hash collisions in ECMP can cause congestion, degrading throughput [44] [88] [89] as well as the tail latency [34] [90] [91] of DCN flows. To balance the load between DCN switches and paths, stateful schedulers have been proposed, e.g., CONGA [43], Hedera [44], etc., which monitor the congestion state of each path and direct flows to less congested paths, and hence are more robust for asymmetric networks without control plane reconfigurations [92] [93]. Since maintaining global congestion information at scale is challenging, local congestion-aware or stochastic scheduling schemes have been

proposed, e.g. Expeditus [93], DRILL [94], Hula [95], etc. Using simple or local information collection, these mechanisms are more efficient and applicable for complex DCN architectures, e.g., 3-tier Clos topologies. However, the majority of these scheduling mechanisms focus on balancing DCN loads according to the congestion information, without any consideration of cloud applications or Data Center traffic patterns.

Existing solutions to mouse-elephant hybrid DCN traffic-scheduling fall into two categories. The first category deploys unified schedulers for both mice and elephants on shared paths, in spite of the competing performance requirements of the two. Based on the analysis of DC traffic patterns, these studies design novel scheduling algorithms or congestion signals [96] [97] and strike the right balance between throughput and latency on shared DCN paths. The main challenge though is the interference between the elephant and mouse flows. The second category deploys network partition schemes that transfer the two types of flows over separate paths [98] [99] [8] [80]. By isolating elephant and mouse flows, network partition solutions avoid the aforementioned interference. This is particularly attractive as hardware and software costs continue to drop. Nonetheless, new policies are required to adaptively partition the DCN paths, given dynamic DC traffic patterns and varied DC architectures.

Latency and throughput optimization for DCN has attracted increasing attention. A series of solutions have been proposed to improve the performance of scale-out multipath Data Center topologies, such as Clos networks [39] [26], FBFLY [24], HyperX [6], DragonFly [53], etc. In general, the performance optimization mechanisms can be classified into two categories: temporal solutions (*i.e.*, congestion control mechanisms) and spatial solutions (*i.e.*, load balancing mechanisms). Specifically, one can classify the existing solutions according to Figure 2.23.

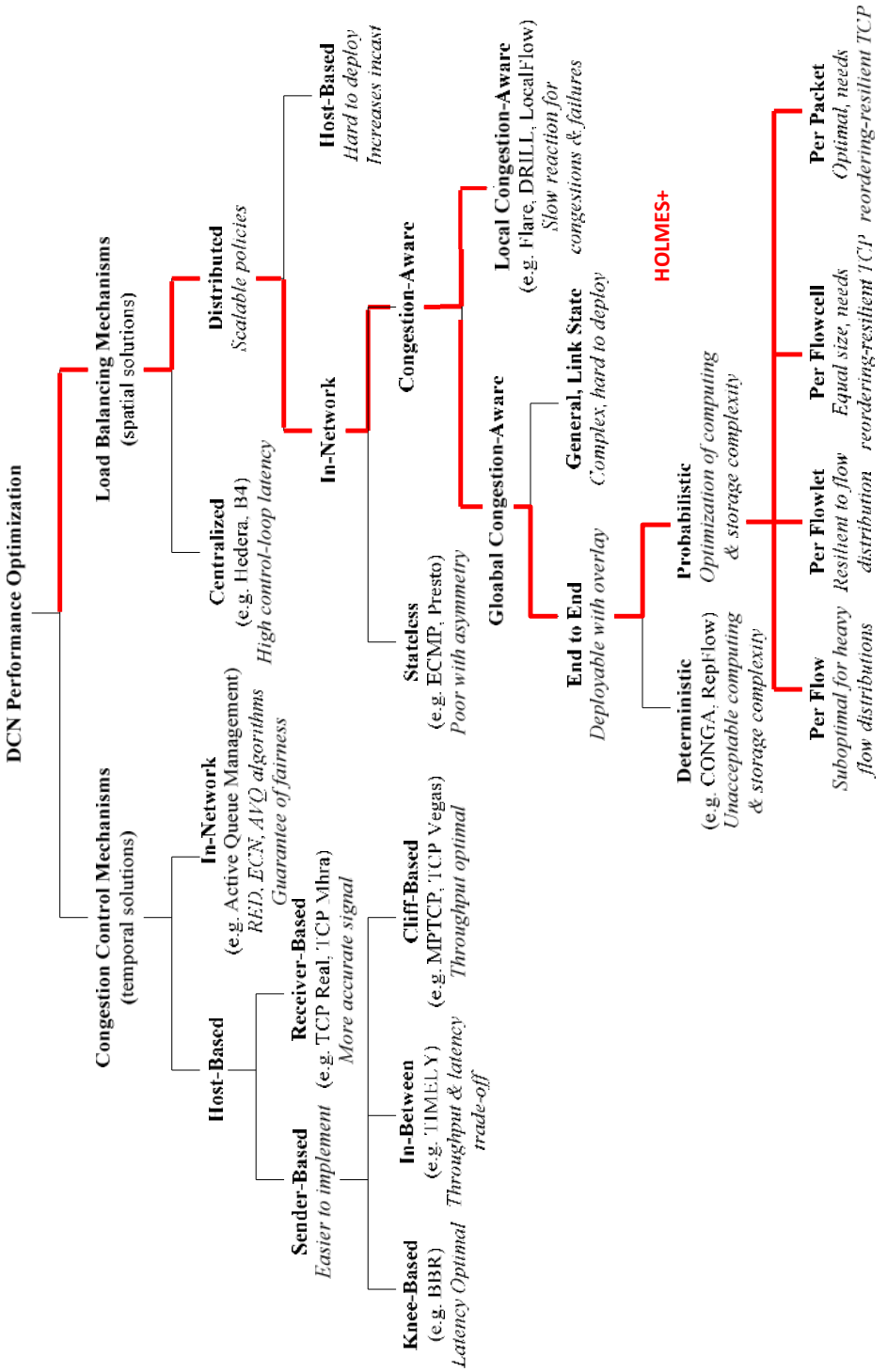


FIGURE 2.23: Category of DCN Flow control and load balancing algorithms.

2.4.1 Congestion Control Mechanisms - Temporal Solutions

DC congestion control is a deeply studied topic. Generally, the congestion control mechanisms adjust the traffic transmission rates or the packet-dropping strategies

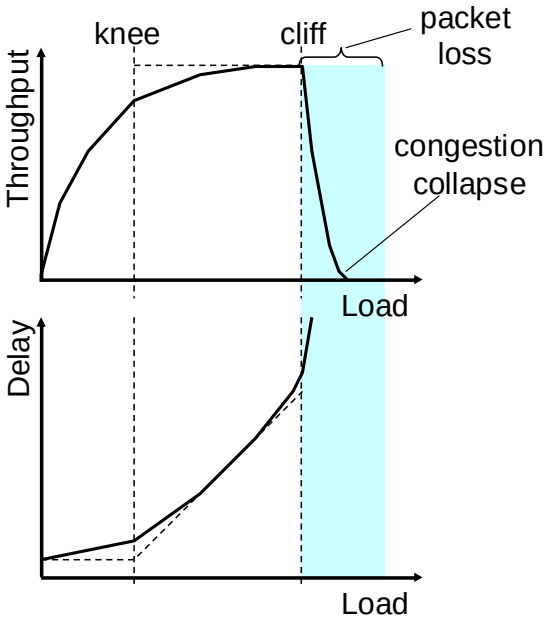


FIGURE 2.24: Network performance as a function of the load [11]

according to the feedback congestion signals. The control mechanisms can be implemented on either the end hosts or the in-network equipment.

2.4.1.1 Host-based congestion control mechanisms

Optimization of transport protocols is usually done through host-based solutions. The host-based control mechanisms can be implemented on either the sender [34] [96] or the receiver of a transportation path [100]. Jain and Ramakrishnan [11] studied the general patterns of response time and throughput of a network as the network load increases, and described the changing trend of network performance curve using two factors: cliff point and knee point. As shown in Figure 2.24 [11], the point of congestion collapse is defined as a cliff due to the fact that the throughput falls after this point (packets start getting lost); and the point after which the increase in throughput is small (buffers of a path start to be filled) is described as a knee point. Correspondingly, host-based congestion control policies can also be categorized by using the two factors.

Cliff-based mechanisms: Most of the modified transportation protocols based on the traditional TCP protocol are cliff-based mechanisms such as MPTCP [101],

DCTCP [34], etc. The cliff-based mechanisms are loss-based solutions, which interpret packet losses as congestion and attempt to fulfill the buffers of the TCP paths while avoiding the occurrence of packet losses. These solutions deploy different types of feedback information from the last time point as congestion signals in order to guide the traffic control at the current time point. For example, DCQCN [102] combines Explicit Congestion Notification (ECN) markings with a Quantized Congestion Notication (QCN) inspired rate-based congestion control algorithm to control DCN flows. Ionescu *et al*. [103] proposed Token-Based Congestion Control (TBCC) based on a closed-loop congestion control mechanism, which controls token resources consumed by an end-user while providing a fair best effort service. Cliff-based mechanisms are usually the throughput-optimal solutions.

Knee-based mechanisms: Cardwell *et al*. [104] pointed out that the operating point of cliff-based mechanisms is not optimal, and argued that when the scale of a DCN is large enough, quantities of packets will accumulate in the buffers of a path. Thus, compared with the data transmission time in the network links, the queuing time in the buffers tend to dominate the overall data transmission latency. Thus, cliff-based solutions are not applicable for the optimization of data transmission latency. Furthermore, they propose a novel congestion control mechanism, BBR, which adjust the operating point from the cliff point to the knee point in order to optimize the data transmission latency of a DCN. Therefore, cliff-based congestion control mechanisms are usually latency-optimal solutions.

In-between knee-based and cliff-based mechanisms: Different from the above two types of solutions, which optimize the throughput or latency of a DCN respectively, a few mechanisms focus on handling the trade-off between latency and throughput, and attempt to find the right balance of the two conflicting factors. Hayes [97] proposed a delay gradient algorithm for TCP congestion control of wide-area networks. Similarly, taking inspiration from Compound [105] and FAST [106], TIMELY [96] also deploys a delay gradient as the congestion signal and proposes a gradient-based algorithm to jointly optimize the latency and throughput of a DCN in different time periods. The TIMELY mechanism works

during the time period between the knee and cliff points, and dynamically adjusts the importance of the throughput and latency issues.

2.4.1.2 In-network congestion control mechanisms

Network congestion usually occurs in the in-network devices, e.g., switches and routers. Thus, compared with the end host-based solutions, in-network congestion control mechanisms achieve more accurate congestion information and react more quickly to congestion and failures. Taking this advantage into account, many researchers have proposed migrating some status monitoring and flow control functions from end hosts to innetwork devices. Correspondingly, a series of in-network congestion control mechanisms have been proposed.

Most in-network congestion control protocols adjust the congestion window size by managing the queues of the DCN routers [103]. A number of Active Queue Management (AQM) algorithms have been proposed to generate congestion signals according to the real-time queue lengths in DCN routers, such as Adaptive RED [107], Adaptive Virtual Queue (AVQ) BLUE [108], etc. Hollot [109] applied classical control system techniques to design novel controllers that were better suited for AQM. Similarly, Firoiu [110] modelled the AQM RED algorithm as a feedback control system and discovered fundamental laws governing the traffic dynamics in TCP/IP networks. pFrabic [90] preemptively schedules flows using packet forwarding priorities in switches, and Detail [91] deploys a similar mechanism that gives priorities to latency-sensitive mouse flows; however, this simple control mechanism makes a mismatch between injection traffic and network capacity, which results in packet loss and bandwidth wastage. These in-network solutions react quickly to real-time congestion and failures; moreover, they can also generate feedback congestion signals to the end hosts, and cooperate with host-based control mechanisms.

2.4.2 Load Balancing Schemes: Spatial Solutions

Different from congestion control mechanisms, load balancing schemes try to improve the DCN performance from a spatial aspect. This kind of solution is especially applicable for the traditional multipath DCN topologies.

Similar to traditional traffic engineering techniques, some studies discuss using a centralized scheme to schedule the DC traffic. Software-driven wide area network (SWAN) [111] and B4 [112] collect statistical information from switches to a central controller, and push forwarding rules to balance the load for inter-datacenter WANs. Fastpass [113] deploys a centralized scheduling algorithm to ensure that queues stay small while the queuing delay remains near optimal. Hedera [44] also applies a centralized scheduling scheme and focus on load balancing across multiple DC paths.

The main shortcoming of these centralized solutions is that they suffer from high control-loop latency in large-scale Data Centers, which is not applicable for handling highly volatile DC traffic in time [95]. Addressing this issue, a number of scalable distributed load balancing schemes have been proposed. One can further categorize these solutions as stateless solutions and congestion-aware solutions.

2.4.2.1 Stateless load balancing schemes

ECMP [87] is a simple hash-based load balancing scheme that is widely used as the de facto scheme in switch ASICs today. The coarse-grained per-flow load balancing and congestion-agnostic hash collisions in ECMP have been shown to cause performance degradation in asymmetric DCN topologies [43] [114] during link failures.

To overcome the aforementioned shortcomings in ECMP, a number of solutions have been proposed to improve the traffic splitting granularity or the load balancing algorithm. Packet-level traffic splitting (PLTS) [115] and Digit-Reversal Bouncing (DRB) [88] are per-packet load balancing schemes that schedule DC

traffic with the granularity of packets. Presto [116] splits traffic into 64KB-sized TSO (TCP Segment Offload) segments. and round robin fashion [89] is deployed in DRB and Presto to spray DC packets or flowcells. Based on Valiant Load Balancing (VLB [117]), some other solutions have been put forward to improve the failure tolerance of homogeneous and heterogeneous network topologies [118].

None of the above solutions are state-aware, however, which causes performance degradation during link failures.

2.4.2.2 Congestion-aware load balancing schemes

The main drawback of the stateless schemes is that they cause performance degradation during link failures. To address this issue, a series of congestion-aware load balancing schemes have been proposed. Based on global or local congestion information, the congestion-aware solutions are more applicable for asymmetric topologies or link/switch failure scenarios.

Global congestion-aware schemes: Global congestion-aware load balancing schemes deploy an end-to-end congestion signal as the feedback metric to schedule the DC traffic among multiple paths. TexXCP [119] are adaptive traffic-engineering proposals that balance the load across multiple ingress-egress paths in a wide-area network by using per-path congestion statistics. CONGA [43] proposes similar solutions for datacenters, by spraying DC traffic among multi-rooted networks based on the congestion state of each end-to-end DC path. RepNet [120] replicates each mouse flow to opportunistically use the less congested path and to reduce the overall flow completion time in DCNs. Inspired by Minimum Delay Routing [121], HALO [122] studies load-sensitive adaptive routing and implements its solution in the router software. These solutions are aware of the overall congestion status of the DCN and react fast to local failures or congestion.

Local congestion-aware schemes: Using global congestion-aware schemes to make load balancing decisions involves scalability challenges. Although the distributed architecture can improve the scalability of the scheduling schemes, they

require coordination between switches or hosts. In large-scale Data Centers with high transmission rates, the continuous reaction to each congestion information introduces additional latencies and degrades the overall flow scheduling performance [94]. Consequently, local congestion-aware solutions have drawn large interests. LocalFlow [123] and Flare [124] studied the switch-local solutions that balance the load on switch ports, without taking the global congestion information into account. Based on the power-of-two-choices model [125] [126], Ghorbani [94] proposed a stochastic switch-local scheduling scheme that further reduces the polling range of local solutions, and improves the execution rate of the flow scheduling algorithm.

As an improvement of CONGA [43], HULA [95] tracks the next hop for the best path and its corresponding utilization for a given destination, instead of maintaining per-path utilization congestion information. This novel strategy makes the load balancing scheme applicable for more complex DCN topologies, besides the two-tier Leaf-Spine topologies. Based on limited congestion information, the local congestion-aware solutions provide suboptimal routing decisions, while improving the overall policy execution rate. However, the stability and convergence of these switch-local solutions need to be ensured; worse, as aforementioned, the local congestion-aware scheduling policies have been proven to react slowly to link failures [43] [95] and are prone to forming congestion trees.

Implementation of the deterministic congestion-aware load balancing schemes requires recording the real-time load status of all the available paths or links in order to make optimal global or local choices. As discussed in Section 5.4, keeping the status information and calculating the optimal solution in large-scale Data Centers, introduce unacceptable storage and computing complexity while handling $O(n^{(d-1)^2})$ multiple-path information growing exponentially at each SE, as shown in Figure 2.8. More importantly, it is impossible to collect information for all paths in real-time. To address this issue, a probabilistic global congestion-aware load balancing algorithm is deployed in HOLMES [12] in order to optimize storage complexity while improving the execution rate of the load balancing algorithm.

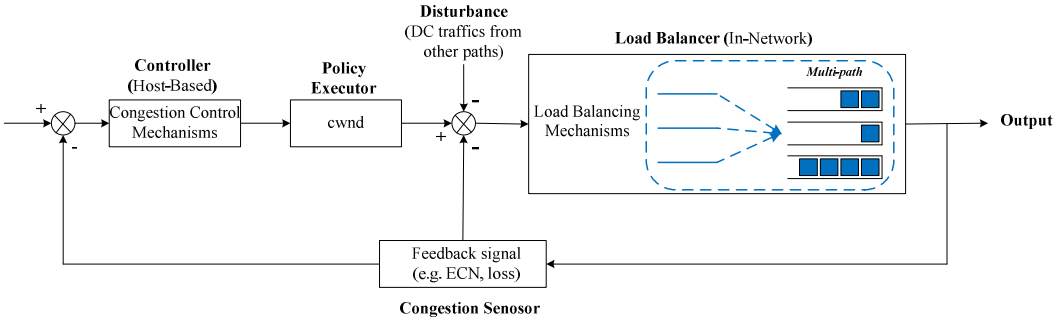


FIGURE 2.25: A control theoretic model for DCN congestion controller and load balancer [12]

2.4.3 Correlations between the Temporal and Spatial Solutions

Both the temporal congestion control mechanisms and spatial load balancing mechanisms aim to optimize the throughput or latency of a DCN. Next, the correlations between the two types of solutions are presented.

Hollot [109], applied classical control system techniques to design controllers and analyze the stability of the same network system under different congestion control mechanisms. Since the publication of the first seminal paper about TCP modeling, by Kelly [127], the Network Utility Maximization (NUM) framework has been widely applied in network resource allocation algorithms as well as congestion control protocols. Inspired by these solutions, a *approximate* control theoretic model is designed to describe the correlations between the congestion control mechanisms and the load balancing mechanisms.

As shown in Figure 2.25, closed-loop based control mechanisms are usually host-based mechanisms, which are generated in a controller located in the host and will be executed later in the in-network devices. A congestion control mechanism typically operates on a single end-to-end path, and it concentrates on the performance optimization of that one path. The main disturbance during the policy execution process is the traffic from other multi-rooted paths. For example, the load balancing policy has to be adjusted accordingly if congestion (disturbance caused) occurs on any of the multi-rooted paths.

Load balancing mechanisms focus on multi-path scenarios, and schedule the traffic of multiple paths to improve the overall performance of all the end-to-end paths. Either solution needs a feedback signal to guide the traffic scheduling in the upcoming control loop. Thus, the feedback signal will be transmitted to both the congestion controller and the load-balancer after each RTT. Load balancing mechanisms are often implemented together with congestion control policies. For example, MPTCP enables parallelized TCP transmissions among multiple paths using its load balancing policies, but still realizes the traffic congestion control using traditional TCP congestion avoidance algorithms. Therefore, it can be considered as part of the closed-loop control system shown in Figure 2.25.

2.4.4 Architecture Improvements

Some researchers have also tried to improve the architecture of the scheduling schemes based on the DC traffic patterns or application features. Freeway [128] dynamically partitions multiple DCN paths into low latency and high throughput paths, and schedules the elephant and mouse flows separately. DevoFlow [129] uses multipath and changes the design of SDN-based OpenFlow switches to enable easier flow aggregation, improving DCN latency and throughput. ADN [130] devolves into the application level, which concentrates on serving the up layer applications. It deploys a novel architecture that slices the whole DCN into logically isolated sub-networks in order to serve different types of applications. The architecture improvements can be implemented together with the aforementioned load balancing and congestion control mechanisms to provide a more comprehensive performance optimization scheme for DCNs.

2.5 Summary

In practice, most of today's Data Centers are based on switch-centric architectures. Switch-centric architectures are similar to traditional network architectures

which makes it easier to upgrade traditional switches although their scalability and flexibility are not optimal. Most of the network components and protocols in conventional networks can be used in switch-centric architectures either directly or with slight modifications.

Although notable improvements have been achieved in previous proposals, some issues, such as complicated management and unaffordable power consumption, have not been adequately solved, and neither can those DCNs satisfy the requirements of new cloud computing applications. As well, as mentioned in [71], prior proposals are insufficient to meet demands on $*$-cast, low latency, high scalability, low cost and power efficiency for DCNs and HPC. In particular, the efficiency of $*$-cast has been neglected in past studies.

Chapter 3

HyperOXN - A Novel Topology for Next-Generation Data Centers

Hyper Optical Cross-Connection Network, HyperOXN for short, is flexibly tunable for cubelike topologies. HyperOXN takes advantage of high-radix switch components leveraging state-of-the-art colorless wavelength division multiplexing technologies. HyperX is used mainly for electrical interconnection, FBFLY and DragonFly assume that all cables are electrical, and HyperX combines hybrid cables, in particular, optical cables using fiber ribbons. HyperX is not suitable for distributed cross-bar networks, but many advantages from HyperX can be inherited in HyperOXN.

This chapter is organized as follows. Section 3.1, concluding and summarizing the present topologies, puts forward the definition of the HyperOXN topology and describes of its some relative properties, and hardware implementation is proposed in Section 3.2. In Section 3.3, the advantage of mathematical induction is shown in order to innovatively demonstrate the rearrangeable non-blocking property of the HyperOXN topology by transforming the HyperOXN topology into a multistage interconnect networks topology. The embedment property of HyperOXN is

discussed in Section 3.4, and finally, in Section 3.5, the properties of HyperOXN are analyzed. The analysis shows that HyperOXN can meet the new challenges for modern DCNs.

3.1 HyperOXN Definition

In this section, the design of HyperOXN is introduced in detail and its main important and fundamental properties are shown. Table 3.1 briefly lists all the symbols used in the thesis.

T	Number of network end-terminals
P	Number of all switch engines in the network
ϕ	Over-subscription ratio
k	Number of end-terminals attached to a switch engine
r	Switch radix
d	Network dimension
n_i	Number of switch engines in the i^{th} dimension
n	Number of switch engines
c	Number of independent channels for each switch
l_i	Link bandwidth or trunking factor for each channel in dimension i
l	Link bandwidth or trunking factor for each channel
δ	Average distance of the network

TABLE 3.1: Symbols used in the model of HyperOXN

Given k-dilated k-way bristled Hypercube's bristle and non-blocking properties, as well as k-ary Hypermesh's simplified link complexity and each dimension's scalability properties, the HyperOXN topology is defined as follows.

As shown in Figure 3.1, each bold line represents a hyperedge, and one hyperedge connects four SEs, each of which connects four ETs. Every SE has four links with identical unit capacity. Each SE can communicate with other SEs on the same hyperedge simultaneously by one independent channel. It is called a 2-dimensional 1-permutation 4-dilated 4-ary HyperOXN.

Definition A Cayley Graph (see details in Section 2.2.2, page 19) $\boldsymbol{CG = (V, E)}$ represents a d-dimensional orthogonal non-directional graph (bidirectional

channels), where $\boldsymbol{V} = \boldsymbol{V}(\boldsymbol{CG})$ is a vertex set and $\boldsymbol{E} = \boldsymbol{E}(\boldsymbol{CG})$ is a *hyperedge* set. As introduced in Section 2.3.2.4, modeling distributed crossbar switches entails the concept of a hyperedge. A hyperedge represents a logical relationship among an arbitrary number of vertices instead of just two vertices. In a ***CG*** graph, like Hypermesh, every vertex is uniquely addressed by an integer, $0, 1, ..., |\boldsymbol{V}| - 1$. The number of all vertices is $P = |\boldsymbol{V}| = \prod_{i=0}^{d-1} n_i$, where n_i represents the number of vertices in the i^{th} dimension. ***Every vertex in the dimension of the*** i^{th} ***hyperedge can communicate with any other distinct*** c_i ***vertex in the same hyperedge simultaneously by*** c_i ***independent channels, where*** $c_i \geq 1$.

Each vertex ($a.k.a$ SE) is connected to some fixed number k of ETs in a HyperOXN, where it is also assumed that each link between an ET and a SE is of unit capacity. A regular HyperOXN is one for which $n_i = n$ and $l_i = l$ for $\forall i \in [0, d-1]$. Thus denote a regular HyperOXN as a 5-tuple (d, c, l, n, k), which is also called d-dimensional c-permutation l-dilated n-ary k-way HyperOXN, as illustrated in Figure 3.1 and Figure 3.2.

From the above definition, a Hypermesh is endowed with a regular HyperOXN-$(d, 1, 1, n, 1)$. A FBFLY is a regular HyperOXN$(d, n, 1, n, n)$.

HyperOXN is well suited for optical hardware implementation as discussed in Section 3.2 below. One channel in HyperOXN represents a wavelength of DWDM. It is worth noting that the hyperedge can function efficiently for $*$-cast communications, which achieves substantial performance benefits for big data analysis and machine learning.

By definition, when $l = 2$, every vertex has two channels, where each channel can only communicate with one out of n vertices in the same dimension in a given time. The above is consistent with Szymanski's 2-dual Hypermesh definition [76], as mentioned in Section 2.3.2.4.

According to the above definition, a regular HyperOXN(d, c, l, n, k) has the following properties:

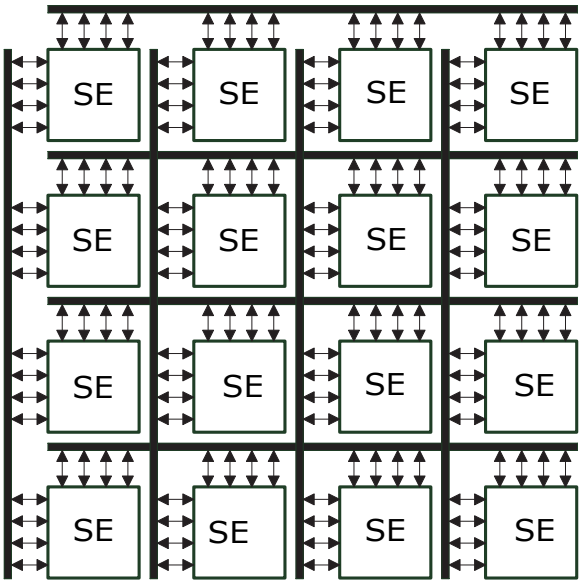


FIGURE 3.1: 4-dilated 4-ary 4-way HyperOXN

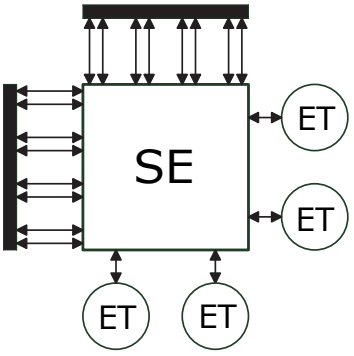


FIGURE 3.2: A unit in the HyperOXN

Property 1: There are $P = n^d$ SEs, where each SE can connect k ETs, and then there are $T = k * n^d$ ETs in the topology.

Property 2: Each hyperedge connects to n vertices on the same dimension, where there are n^d/n hyperedges for each dimension and has $d * n^{d-1}$ hyperedges.

Property 3: Since each vertex has c channels in each dimension and each channel is fully duplex, the whole topology has $c * n^d$ channels.

Property 4: Diameter of the topology is d.

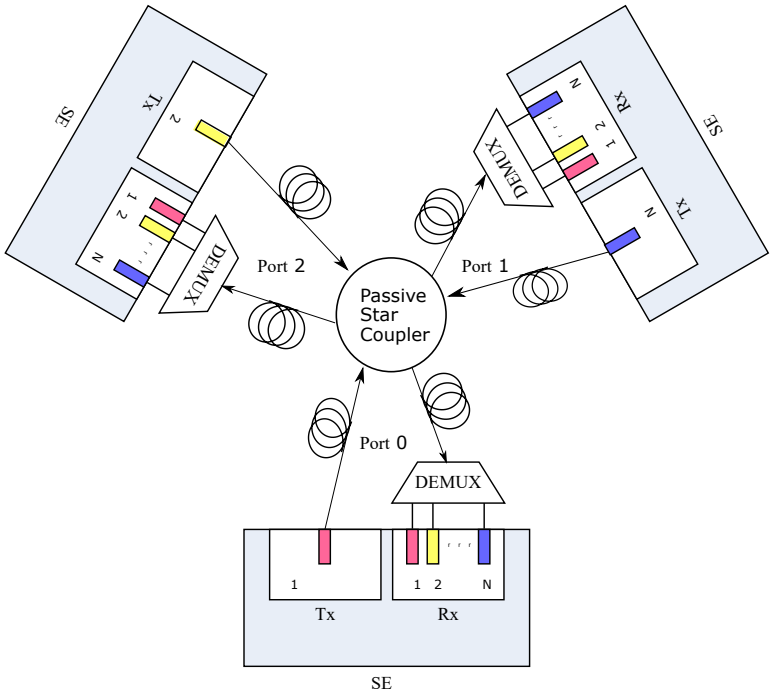


FIGURE 3.3: A sample of optical implementation for HyperOXN$(1, 1, 1, n, 1)$

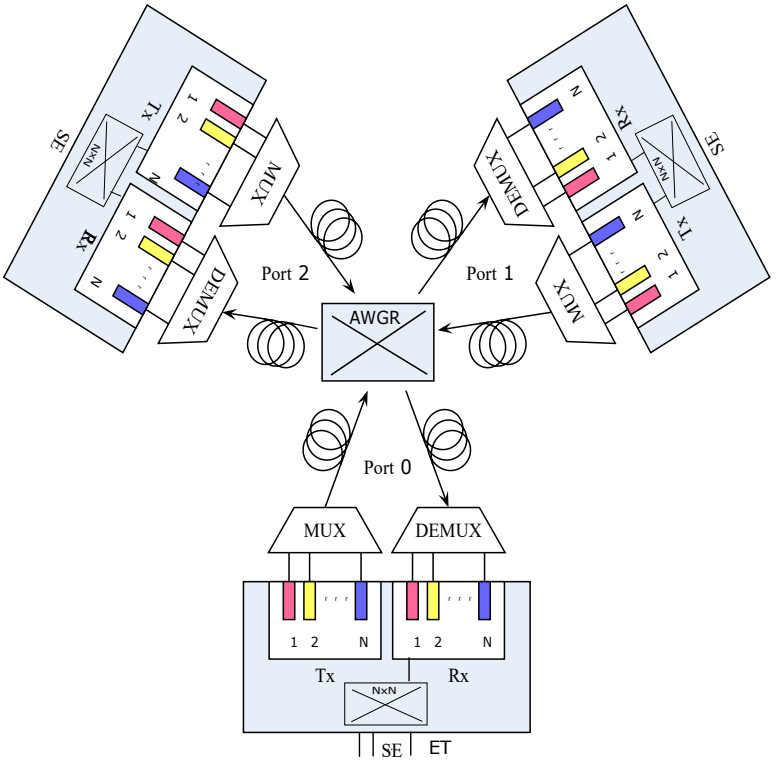


FIGURE 3.4: A sample of optical implementation for HyperOXN$(1, n, 1, n, n)$

Property 5: The degree per vertex is $d * c$ without considering physical implementation and without counting the connections between SEs and ETs.

Property 6: The distance between any two vertices is equal to its Hamming distance.

Property 7: A d-dimensional regular HyperOXN can be recursively described by $n * (d\text{-}1)$-dimensional HyperOXNs where $n * (d\text{-}1)$-dimensional HyperOXNs are connected through the vertex's hyperedge.

Property 8: There are d ways to decompose a d-dimensional HyperOXN into $n * (d\text{-}1)$ dimensional HyperOXNs, where $n * (d\text{-}1)$-dimensional HyperOXNs are symmetrical.

Property 9: The topologies including FBFLY , Hypercube, HyperX and Hypermesh can be embedded into HyperOXN, which means that routing algorithms or other known results obtained for those topology networks can be used directly for HyperOXN.

Property 10: The average distance δ in the regular HyperOXN is roughly equivalent to

$$\delta \approx d * \frac{(n-1)}{n} \tag{3.1}$$

which is similarly given in [30] when P is large for large-scale networks.

For example, as seen in Figure 3.1 and Figure 3.2, HyperOXN topology has 16 SEs, 64 ETs, eight hyperedges and 128 channels. The degree of every SE in the Figure 3.2 is 8, the maximum Hamming distance of any two nodes is two, and the topology consists of eight 1-dimensional HyperOXNs.

3.1.1 Topology Optimization

The topology optimization is intended to find optimal HyperOXN configurations with the constraints of bisection bandwidth, network size, latency, and switch radix requirements.

The bisection cut for each dimension divides the HyperOXN into two equal halves, $S1$ and $S2$, when dimension i is bisected . The first half, $S1$, has $P/2$ switches, and each one has c channels crossing through $S2$ when p is less than $n_i/2$; otherwise, it is equal to $n_i/2$ while $c > n_i/2$ according to the above definition for c-permutation. As such, the bisection bandwidth of a HyperOXN(d, c, l, n, k) is

$$BB_{HOXN} = \min |P/2 \cdot MIN(c, n_i/2) \cdot l_i|,\ 1 \leq i \leq d \tag{3.2}$$

where the MIN function returns the minimum value of two arguments.

To satisfy some demands on network size (number of ETs) T_C, switch radix R_C, latency L_C, and minimum bisection bandwidth B_C, these constraints are denoted as follows.

First, the topology should satisfy the minimum network size constraint,

$$e * P = e * \left(\prod_{i=0}^{d-1} n_i \right) \geq T_C \tag{3.3}$$

Second, it should meet the switch radix constraint,

$$e + \sum_{i=1}^{d} c_i \cdot l_i \leq R_C \tag{3.4}$$

Third, set the latency demand:

$$\delta \leq L_C \tag{3.5}$$

and the minimum bisection bandwidth requirement:

$$\phi \geq B_C \tag{3.6}$$

We have implemented an algorithm to find a general HyperOXN that meets the requirements of Section 3.1 and has the fewest switches.

A simple brute-force search of the design space or linear planning optimization algorithms can be used to achieve the best solution with the fewest switches under constraints in (3.4), (3.5), (3.3) and (3.6). Practically, the design space can be restricted to speed up searching by limiting some parameters, *e.g.*, setting the dimension to less than 5. The irregular HyperOXN is more attractive than the regular one regarding cost and flexibility, but the regular HyperOXN is easier to construct and maintain.

For example, for the design parameters: $R_C = 128$, $T_C = 2^{17}$, and $B_C = 1$ (full bisection-bandwidth) ignoring the latency constraint here, through a brute-force search of the design space, the best regular solution is HyperOXN$(4, 10, 3, 10, 10)$ which has four dimensions with $10,000$ switches, and the general one is HyperOXN $\big(3, (5, 38, 38), (8, 1, 1), (5, 38, 38), 19\big)$ with only 3 dimensions and $7,220$ switches.

3.2 Hardware Implementation

The DWDM technology has been seen as dominating in high throughput DCNs. It can significantly reduce power consumption and tame the flying cable complexity. Researchers have made fast progress in CDC(Colorless, Directionless and Contentionless), which allows operators to future-proof their optical networks so that they are easy to scale, optimize, and flexibly meet any bandwidth demands in the future. With current technology, one optical fiber can support up to 192 simultaneous wavelength channels, each having speeds of 25Gbps [131].

With the advances in cutting-edge silicon photonics, which is a disruptive technology used to revolutionize Data Centers, CMOS fabrication can be used resulting in high-yield production at low cost. To date, 168 channels and 12×14 vertical-cavity surface-emitting laser (VCSEL) and photo-diode (PD) matrices have been hybrid integrated into on-chip CMOS package to demonstrate the delivery of data rates of up to 1.34 Tb/s full duplex optical interconnect [132]. The complexity of transmitting port with VCSELs is much higher than that of receiving port with PDs because VCSELs have to provide laser sources with different wavelengths, but

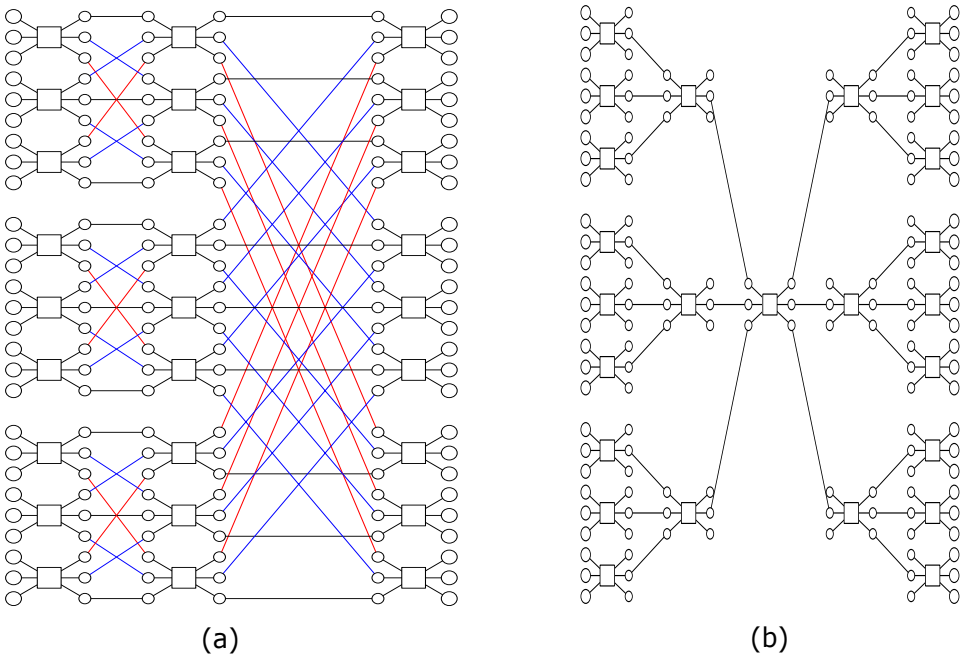


FIGURE 3.5: Interconnect topology within a 27 × 27 coupler fabric [13] (a) Banyan topology (b)Native combine-and-split topology

PDs just function as identical inactive detectors. However, the colorless DWDM technology can be leveraged to largely reduce VCSEL complexity and cost.

Inspired by POXN [13] and MegaSwitch [133], two optical implementation samples are given as shown in Figure 3.3 and Figure 3.4 and the DWDM and passive optical devices (POSC, AWGR, etc.) can be leveraged to implement HyperOXN with much lower power consumption (see details in Section 3.6, page 87).

3.2.1 Passive Optical Star Coupler (POSC)

Two low-cost designs of the $N \times N$ coupler fabric, shown in Figure 3.5, were proposed using 4.77-dB 3×3 unit couplers in [13]. The power loss at any output in Figure 3.5(b) using native combine-and-split topology is $5.47 \cdot (2 \cdot \lceil \log_3 N \rceil - 1) - 0.2$ dB, considering the fundamental power split loss (4.77 dB), excess loss (0.5 dB), and fiber splice loss (0.2 dB). The design in Figure 3.5(a) is more efficient than the one in (a) by using a Banyan interconnect, corresponding to a power loss of $5.47 \cdot \lceil \log_3 N \rceil - 0.2$ dB at each output. The optical passive star coupler can be

scaled up to $N = 3^4 = 81$ with a power loss of 21.68 dB. In general, the power loss of a POSC will be reduced when using high-radix unit couplers, but may lead to an increase in manufacturing complexity.

3.2.2 Arrayed-Waveguide Grating Router (AWGR)

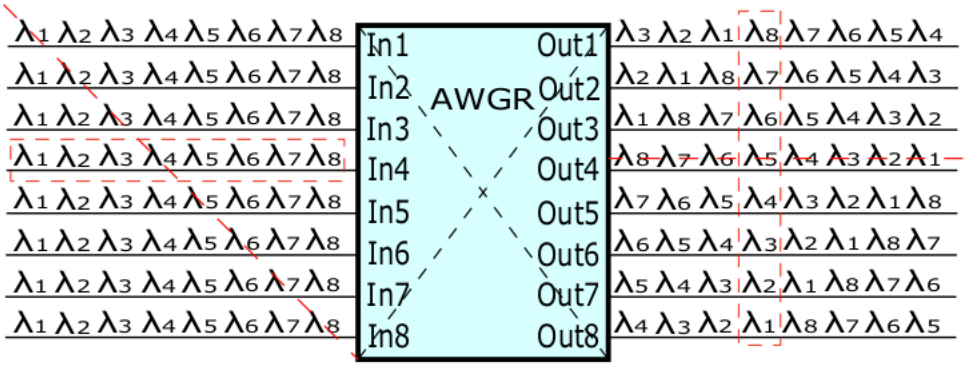


FIGURE 3.6: The routing property of an 8×8 AWGR with all-to-all connections using 8 wavelengths

An AWGR, shown in Figure 3.7, consists of I/O waveguides, an array of waveguides called phased array (PA) waveguides and two free propagation regions (FPR) star couplers. The two FPR are connected with PAs at different lengths, and can function as both multiplexers and de-multiplexers. The input FPR expands an optical signal consisting of multiple wavelengths, and the PA captures this diverging signal and propagates toward the input aperture of the output FPR. The optical signal interferes constructively and converges at one single focal point on the focal line in the output FPR. By placing each phased waveguide in the correct position, the field for each wavelength of the input optical signal can be coupled into the respective output waveguide connected with the output port.

The $n \times n$ AWGR diagram depicted in Figure 3.6, routes the signal from one input to reach one output only on a particular channel determined by its cyclic routing property: the signal carried by λ_i at input $j(1 \leq j \leq n)$ will be forwarded to output $k(1 \leq k \leq n)$ if and only if $(i - j - 1)\ mod\ n = k$, where $\lambda_i \in \{\lambda_i, 1 \leq i \leq n\}$. AWGR has the intrinsic merit of non-blocking shuffle wavelength signals to realize $*$-cast communication.

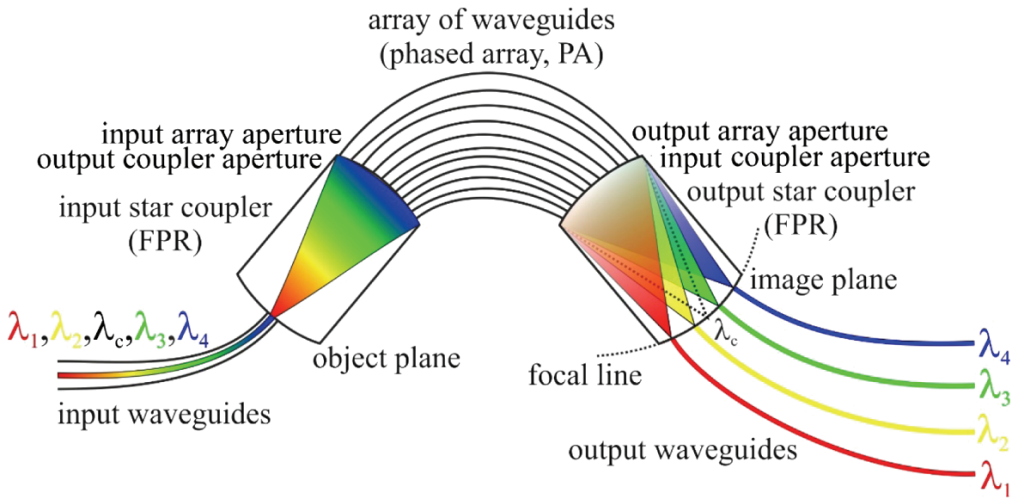


FIGURE 3.7: Principle of an AWGR [14]

Unlike passive optical star couplers, AWGRs have fixed insertion loss (typically, around 5 dB) regardless of the number of supporting wavelengths.

Due to limits in current technology, it is harder to get large-ports AWGRs. One can leverage smaller ports of AWGRs with corresponding smaller number of wavelengths to scale up in a large system. An 8×8 is constructed by four 4-port AWGRs, as shown in Figure 3.8, while accordingly using only 4 wavelengths.

3.2.3 Non-blocking $*$-cast for Optical Hyperedges in HyperOXN

To support diverse $*$-cast traffic patterns by utilizing a low-latency optical $B\&S$ architecture, prior designs [16] [134] [13] generalized optical splitter interconnection based on WSDM through Hyperedge modeling introduced in Section 2.3.2.4, page 31. By definition, the hyperedge has an obvious, intuitive appeal to support $*$-cast traffic well. The sender transmits messages to all nodes in a one-time effort, and the hyperedge itself delivers messages accordingly as shown in Figure 3.9 [15]. It is composed of two different stages. Each node transmits messages on a different bus channel that connects all receivers in the first stage, and then each receiver node selects the messages destined for that node in the second stage. This is

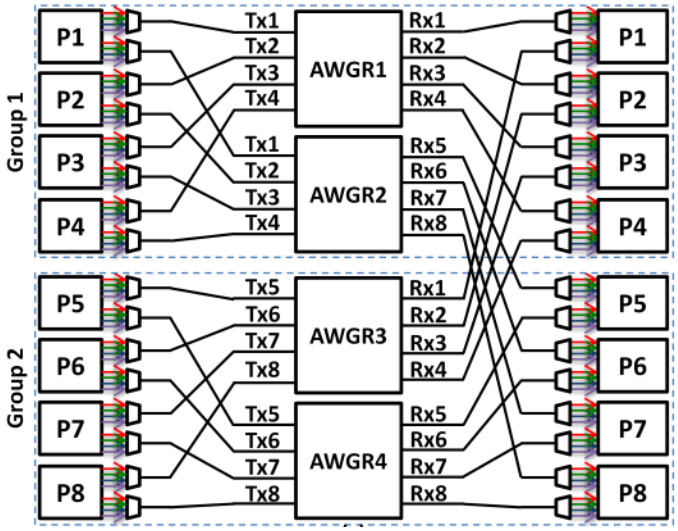


FIGURE 3.8: 8×8 by 4 4-port AWGRs

similar to the publish-subscribe message bus. Since the bus is the most well-known hyperedge, this similarity is not a coincidence. An optical hyperedge does this bus job in the physical layer. As shown in Figure 3.3, each node transmits packets on a different wavelength. These wavelengths are multiplexed, split and broadcasted to all the SEs through a POSC. Each SE uses a demultiplexer and selects the wavelengths that will be forwarded to itself.

Besides optical hyperedges based on the *B&S* architecture support non-blocking one-to-one, many-to-many multi-cast, broadcast traffic patterns, they can also meet in-cast (many-to-one) demands efficiently.

As shown in Figure 3.3, suppose the transmitter Tx1 of port 1, Tx2 of port 2, Tx3 of port 3 and Tx4 of port 4 would all like to send data to the Rx1 of port 1 at the same time. This can be supported by a hyperedge, but is impossible for all other existing switched-based architectures.

The characteristic of supporting diverse $*$-cast traffic patterns can be intuitively expanded to multi-dimensional hyperedges in HyperOXN.

A distributed *B&S* non-blocking switch that is based on an optics-implemented hyperedge intrinsically and efficiently supports $*$-cast traffic patterns. In particular, the selection stage of this design is delayed to the destination nodes, which

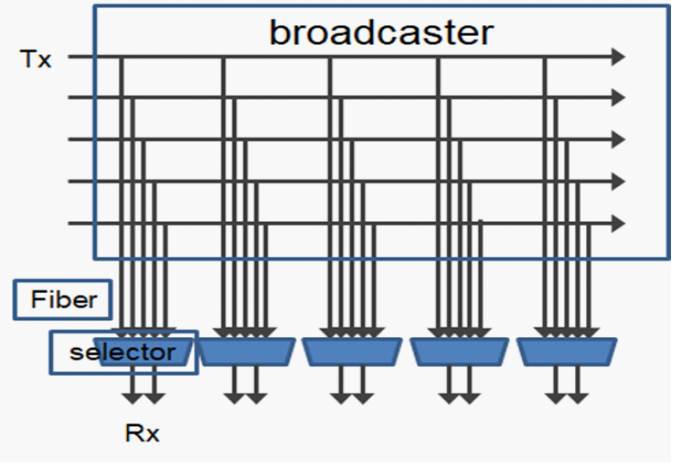


FIGURE 3.9: The Architecture of a Broadcast and Select Bus [15]

is different from traditional electronic broadcast-select crossbar switches. The benefit of this design is that it isolates the problem of multi-cast group over-subscription into destination nodes. Another benefit of the design is its capability of non-blocking in a combination of uni-cast and multi-cast traffic. The cost of that benefit is that it needs much larger bandwidth of an optical connection cable. Nowadays, optical fiber has big bandwidth and is commercially cost effective.

A hyperedge based on an optical $B\&S$ architecture may appear to be equivalent to a crossbar, but it has much more functionality. As mentioned in [16], unlike a traditional electronic hyperedge, it avoids the arbitration latency, switching configuration, head-of-line blocking, and need to inform the sending node that the connection is complete. The ability to support multiple simultaneous broadcasts is a unique feature of this hyperedge which efficiently supports $*$-cast traffic patterns.

As there is a limit on the number of wavelengths of a single fiber in today's optics technology, one can use ribbon optical cables, each of which is composed of up to 48 fibers, expanding the number of nodes a hyperedge may support as shown in Figure 3.10(a). Current technology can offer 46 fibers in one carbon. One can also employ four N-node hyperedge to scale up to a $2N$-node hyperedge, as demonstrated in Figure 3.10(b).

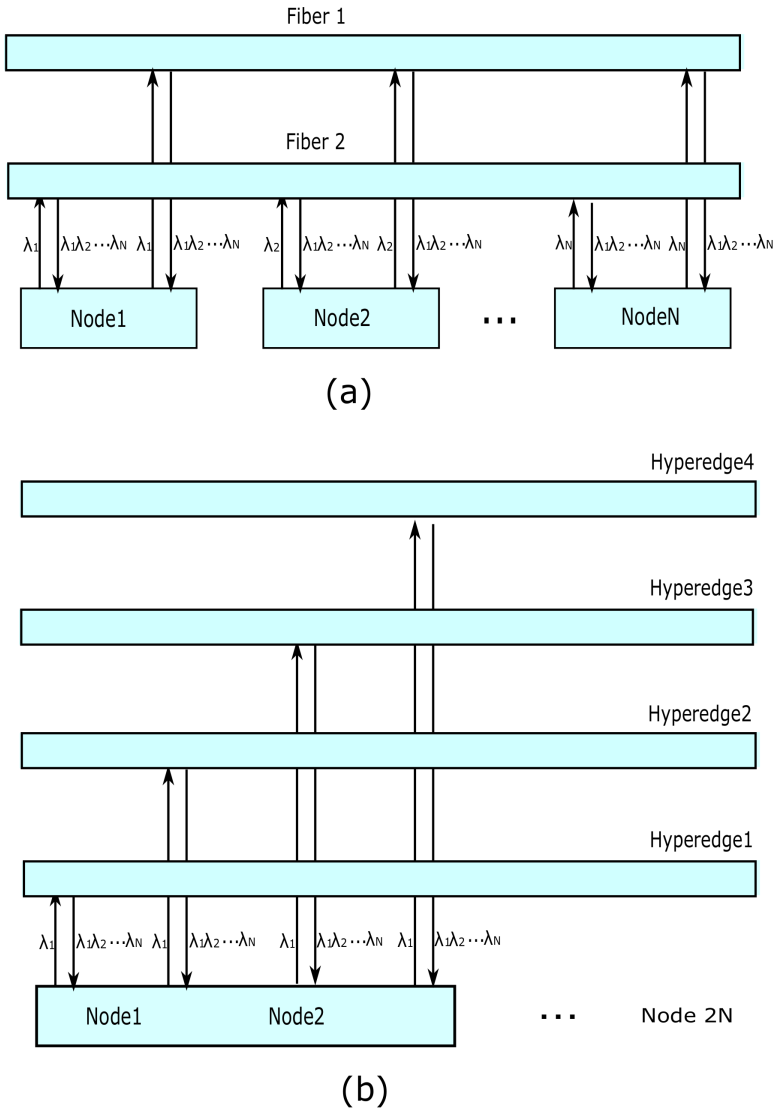


FIGURE 3.10: Scale up for Hyperedges

Plant [135] took advantage of advanced silicon-photonics and opto-electronics technology and developed a 256 channel bidirectional optical interconnect using VCSELs and PDs on CMOS, but a hyperedge based on *B&S* architecture only needs one or two transmitters and a low-cost array of PDs, which is easier to manufacture than to integrate an array of active VCSELs into one chip. Katsinis [16] proposed a 256-node hyperedge-like SOME-Bus engineering an array of amorphous silicon photodetectors and directly writing Slant Bragg gratings [16] into the fiber core as shown in Figure 3.11.

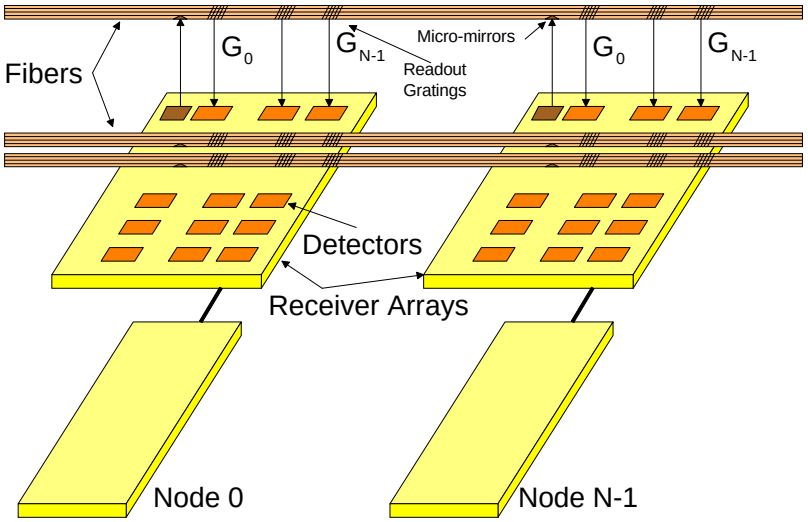


FIGURE 3.11: Parallel receiver array [16]

3.2.4 Implementation Cases

In Figure 3.3, each SE has a transmitting port with only one VCSEL and receiving port, and the SEs can be plugged into servers. POSC can be implemented as Passive Optical Fabric, which has no power requirements or active electronic parts. Each transmitting port works with fixed distinct channels. The receiving port needs to detect all wavelength signals and filter out packets not belonging to itself. A simple circuit logic can be designed to fulfill this filter function.

The hyperedge in Figure 3.3 functions as a multi-send-all-receive broadcast-and-select network, which is well suited for $*$-cast traffic patterns. Note that the physical layer address distinguishing each specific receiving port may be coded in a packet with only a minor bandwidth penalty.

Provided that the basic topology HyperOXN$(1, 1, 1, n, 1)$ is extended to HyperOXN $(3, 1, 1, n, 1)$ with $d = 3$ dimensions and $n = 128$ channels, $128 \times 128 \times 128$ SEs connectivity can be achieved, the capacity of which is more than two million ETs and only needs P pairs of optical cables. The number of required optical cables is much smaller than those for FBFLY and Fat-Tree.

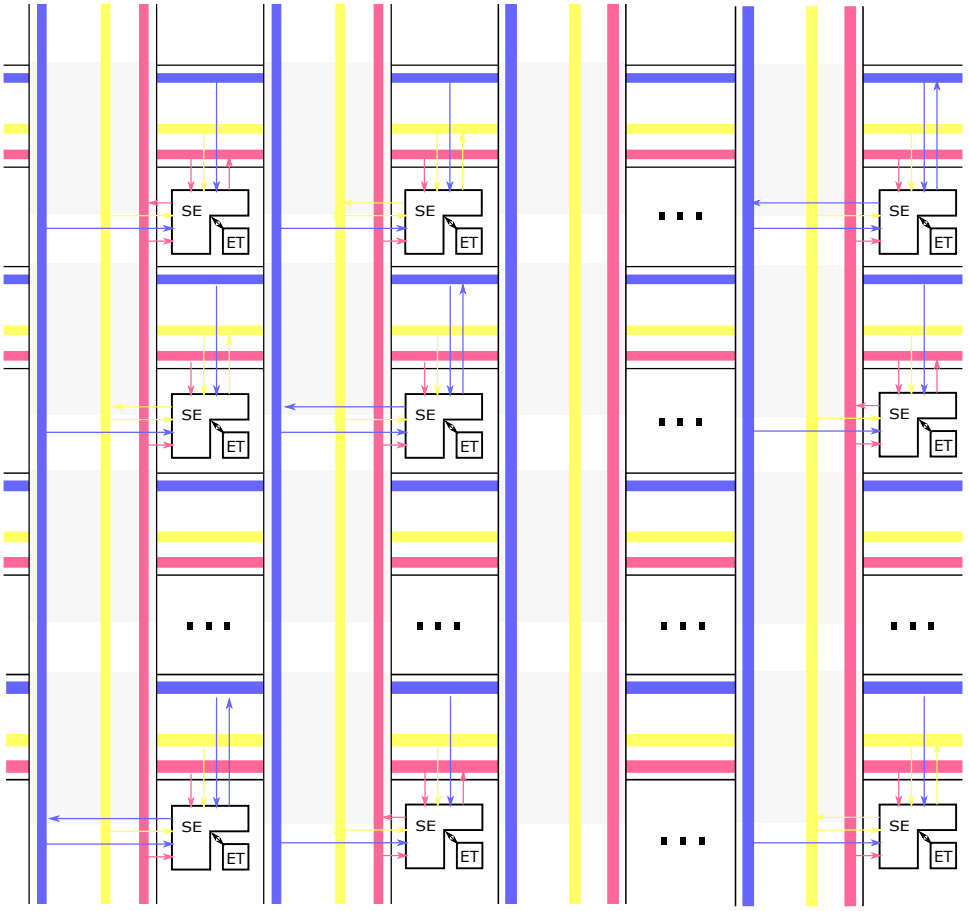


FIGURE 3.12: The block diagram for HyperOXN$(2, 1, 1, n, 1)$

Figure 3.12 depicts a two-dimension HyperOXN$(2, 1, 1, n, 1)$ with one wavelength per channel. Each SE is connected to one horizontal and one vertical hyperedge providing a broadcast-and-select interconnect network. At each SE node, one dedicated broadcast wavelength for sending is assigned as well as a set of channels for selective receiving at each dimension. Messages can be forwarded and broadcast between the two-dimensional hyperedges. As with different optical domains, all the hyperedges may use the same wavelengths and allow all attached nodes to send messages at the same time.

Figure 3.13 shows the block diagram of an SE node in Figure 3.12. Both of the receivers at each node are employed by two address filters to examine the packet header, in which the physical layer address is coded. It also detects if the incoming packet will be ignored or received. If a broadcast or multi-cast packet is detected in one dimension, this packet will be bypassed in a cut-through manner to another

dimension. The switch in Figure 3.13 forwards packets among the attached ET and the two ports connected to two hyperedges.

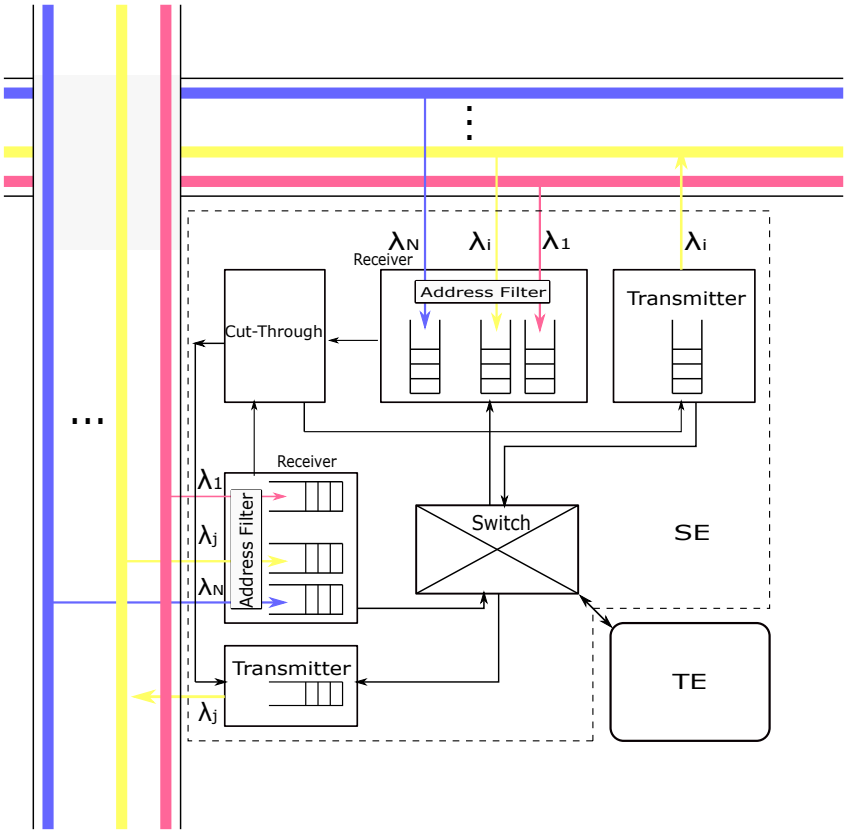


FIGURE 3.13: The hardware block diagram of SE for HyperOXN$(2, 1, 1, n, 1)$

Figure 3.4 shows how an AWGR can be constructed as a passive optical fabric. Leveraging the AWGR's property of non-blocking shuffle switching to realize $*$-cast communication, each SE is equipped with multiple receivers and multiple transmitters working on different channels, with low cost and more efficiency.

HyperOXN$(1, n, 1, n, n)$ shown in Figure 3.4, can be extended to multiple dimensions. Assuming $n = 128$ channels, HyperOXN$(2, n, 1, n, n)$ can accomplish $2,097,152$ ETs with a $3n$-radix switch. Compared with HyperOXN$(3, 1, 1, n, 1)$, the complexity of the high-radix switch in HyperOXN$(2, n, 1, n, n)$ might limit some hardware implementations and also increase cost ,as shown in Figure 3.4, although the dimensions needed can be reduced even to two, resulting in low latency. In some cases, SE can be designed as an independent ToR switch separated with servers.

As well, the design diagram in Figure 3.4 can also be used to construct any HyperOXN with different c, $1 \leq c \leq n-1$; *e.g.*, send packets over d channels simultaneously.

3.2.5 Transceiver Cost and Power for HyperOXN

In this section, the transceiver cost and power consumption for HyperOXN are quantitatively analyzed.

3.2.5.1 Cost

A conventional optical transceiver module incorporates both a transmitter (Tx) and receiver (Rx), as illustrated in Figure 3.14. Optical transceivers convert the signal from electrical to optical, and vice versa, at different link rates depending upon the chosen standard.

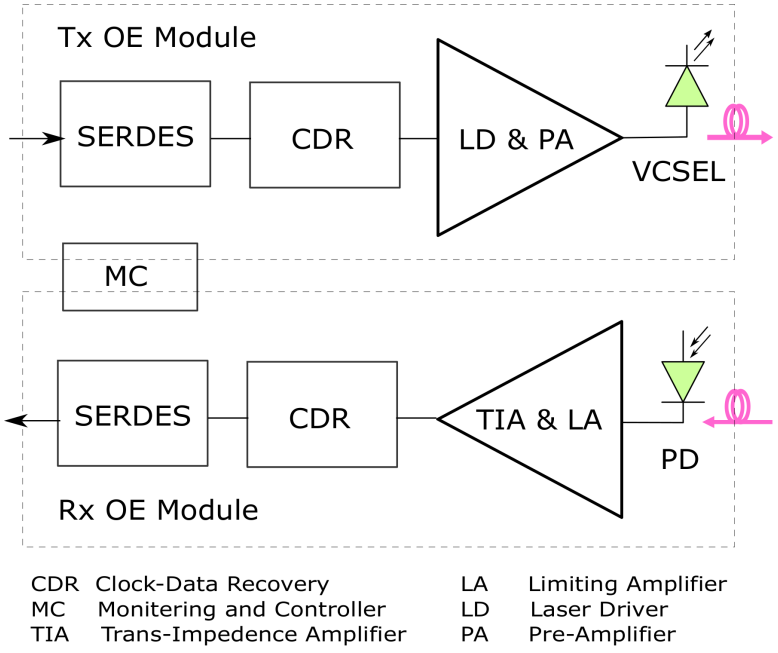


FIGURE 3.14: The block diagram for a transceiver

The transmitter is equipped with an active vertical-cavity surface-emitting laser (VCSEL) followed by a laser driver and pre-amplifier (LD & PA) to boost the electrical signal, clock-data recovery (CDR), and a serializer/deserializer (SERDES). A photo-diode (PD), which converts the optical intensity of the light into electrical current, and a trans-impedance and limiting amplifier (TIA & LA) are used on the receiver side. Also, A monitor controller (MC) is added to manage the transceiver for monitoring to get status information and for configuration.

Component	Cost ratio
LD & PA	10%
CDR	16%
VCSEL	32%
PD	20%
SERDES	10%
Others (power circuit, PCB, fabrication, etc.)	10%

TABLE 3.2: Cost breakdown of a conventional transceiver

According to the ratio for component costs in Table 3.2, Tx has approximately 60% of the cost of a transceiver, and nearly 40% for Rx, which implies that the dominant cost in a conventional transceiver comes from the Tx side.

The electronic physical layer in a conventional transceiver, as shown in Figure 3.14, employs a SERDES for handling clocking, time-multiplexing and data retiming issues, and a CDR to process data packets. It is well known the always-on SERDES and CDR consume much power [136] and become a significant factor affecting the energy dissipation of transceivers. However, thanks to the technology of silicon photonics, optics can potentially eliminate the use of SERDES and CDR by leveraging "quantum impedance conversion" (QIC) [137]. As such, the cost and power consumption for SERDES and CDR will be ignored throughout the following discussion. Therefore, the Tx cost rises up to about 80% of the overall cost of a QIC-based transceiver to be used in HyperOXN, and the Rx cost is reduced down to only 20% . Further more, an array of passive PDs based on silicon-photonics technology for exploring multi-channel receiver will be employed to drive down the Rx cost. The Rx cost will not grow much compared to only one PD increasing by a factor of δ_1 ($\delta_1 < 1$), or may use only one PD for multi-wavelength channels

by using the PSD technology mentioned in Section 2.3.5.6, page 40. The novelty of HyperOXN lies in the varied capabilities of the transceiver with relatively low cost and power consumption while supporting only one transmitter and large n-wavelength receivers simultaneously.

Vendors	10G SFP+ SR Transceiver ($)	10G ToR Port ($)
Intel	125	n/a
Mellanox	123	124
Myricom	185	n/a
Solarflare	150	n/a
Pica8	110	140
QuantaMesh	n/a	104
Edgecore	n/a	100

TABLE 3.3: Prices for a 10G commodity short-range SPF+ transceiver and a 10G port of electronic packet switches

In summary, let C_t represent the cost of a transceiver using technologies such as QIC and silicon-photonics, and then the cost of a HyperOXN transceiver supporting n PDs and one VCSEL is estimated to be approximately

$$C_{ht} = C_t(1 + 20\% * (n-1) * \delta_1). \tag{3.7}$$

The prices in Table 3.3, from the online Colfaxdirect [138] (September 2018) are very consistent with the earlier analysis in ProjecTor[79] using costs of $80 per 10G SFP+ short-range transceiver and $90 per 10G port on a ToR switch. These data imply that the cost of a transceiver is roughly equal to that for one port of an electronic ToR switch for a 10G transportation speed. In terms of 25G, 40G, and 100G scenarios, today's technologies may not achieve the target ratio of $1:1$ (denoted as $\delta_2 = 1$) with higher optical costs, but following the trend, this ratio will be used throughout the comparisons below.

3.2.5.2 Power Consumption

The power efficiency of CMOS technology should meet critical challenges and is much more difficult to improve on while approaching the end of Moore's Law.

However, silicon photonics is a promising technology for providing low-cost and energy-efficiency for emerging intra- and inter-rack high speed interconnects in data-center and high-performance computing applications. As mentioned above, low-cost photonic transceivers should significantly reduce the total cost for Data Centers. Also, because of its extensive use in DCNs nowadays, designing energy-efficient photonic transceivers with high-speed data-rates can revolutionize the topologies of Data Centers and achieve substantial performance benefits over electronic links.

The past decade has produced a large number of proposals on photonic transceivers for DCNs, and many impressive results have been reported. Most recently, Stojanovic *et al*. (2018) [139] proposed a state-of-the-art 40 G monolithic microring-based wavelength-division multiplexing photonic transmitter in a 32/45 nm silicon-on-insulator CMOS process. The transmitter consumes only 2.74 mW, i.e, 685 fJ/b (femtojoule per bit) total energy efficiency (42 fJ/b modulator and driver energies), and the receiver achieved 0.36 pJ/b at a 12 Gbs data-rate. This design can combine resonant photodetectors and easy to implement multi-wavelength receivers and can be potentially extended to the design of transceivers for HyperOXN. Also, the receiver power efficiency can be further improved if using a custom process such as being based on pure Ge materials. Li [140] presented a 5×25 Gb/s carrier-depletion microring-based wavelength-division multiplexing (WDM) transmitter in a 65 nm CMOS, with each transmitter channel consuming 113.5 mW while achieving a power efficiency of 4.54 pJ/b at the receiver side. In general, in terms of active photonics and laser power (0dBm - 5dBm), each transmitter consumes $2 - 10$ times (denoted as a factor δ_3) more power in total compared to the passive receiver, as shown in Table 3.4. As new nanotechnologies emerge in the future, at least two orders of magnitude improvement in the energy-efficiency at the receiver side in comparison with the transmitter [139], it should be very beneficial when implementing HyperOXN to drastically reduce cost and improve energy-efficiency.

The power consumption ratio of one electronic ToR port to a photonic transceiver is assumed as δ_4. As shown in Table 3.3 and 3.5, $\delta_4 \geq 2$ with current technologies

and should grow up at least one order of magnitude if using silicon-photonics.

	Tx Speed	Tx Energy (pJ/b)	Rx Speed	Full Rx Energy (pJ/b)
Berkeley [139]	40G	0.685*	12G	0.36
IMEC [141]	20G	0.61†	20G	0.56
HP [140][142]	25G	4.54*	25G	0.68
IBM [143]	25G	10.8†	25G	3.8
STM [144]	56G	5.35†	25G	1.24

$\star$ Full transmitter energy except laser power
† Only for modulator and driver

TABLE 3.4: Power consumption of optical transmitters and receivers based on silicon-photonics

Letting P_t denote the power consumption of a photonic transceiver for one channel, P_{tx} and P_{rx} stand for the power dissipation of the Tx and Rx for one channel respectively, and δ_4 is defined as the ratio of the P_{tx} to P_{rx}. Thus,

$$P_t = P_{tx} + P_{rx} = (1 + \delta_4) \cdot P_{rx} \tag{3.8}$$

The power consumption of a HyperOXN transceiver supporting an n-wavelength receiver and one wavelength transmitter, denoted as P_{ht}, is estimated to be about

$$P_{ht} = P_{tx} + n \cdot P_{rx}. \tag{3.9}$$

3.2.6 Power Budget Analysis

There is no need to use any active optical devices in the design of HyperOXN in order to greatly reduce power consumption. Consequently, the power budget is a significant factor in the scale of a hyperedge.

The scale of an optical system is determined by the emitting power of laser transmitters, the sensitivity required of the receivers, and all the losses incurred between the transmitters and receivers. In a typical optical network, a laser output power of 7 dBm (with a 1550nm wavelength and 5mW launch power in long-reach optical transport networks) is used, and most Pin-Diode based receivers have a typical

sensitivity of -28 dBm. This leaves a maximum attenuation of 35 dB in a passive optical network, which is equal to the transmitter output power subtracting the receiver sensitivity. As such, a system power budget of 35 dB can be reasonably presumed for HyperOXN architecture design.

Let L_{mux} be the total insertion loss of the multiplexers in a HyperOXN system, and L_demux be the total insertion losses of the demultiplexers. Let L_{posc} be the total loss of the POSC and L_{awgr} be the loss of the AWGR, and L_f represents the fiber loss between a transmitter and receiver. The fiber loss is assumed as 0.25 dB/km for DWDM, and the maximum reach requirement for an intra-datacenter optical interconnect is 10 km. Consequently, the maximum L_f is 3 dB. The total span transmission loss in the HyperOXN is given by the following equation:

$$L_{total} = Lmux + L_{demux} + L_f + Lposc + Lawgr + L_{others},$$

where L_{others} depicts all other losses from splices, connectors, etc., with typically 5 dB considered in the design. There is still some margin, yielding a 128×128 POSC used for the hyperedge in Figure 3.3, showing that the HyperOXN system is feasible because of the system power budget of 35 dB. This value 128×128 is well within the capabilities of current passive optical star coupler technology. The link loss of the HyperOXN using $N \times N$ AWGRs in Figure 3.4 is independent on the number of ports, N, which gives much room to scale up such networks.

Note that details of the hardware designs and the prototype will be introduced in future work.

Unlike the HyperOXN$(1, 1, 1, n, 1)$ shown in Figure 3.3, the HyperOXN$(1, n - 1, 1, n, n)$ in Figure 3.4 leverages the AWGR's unique feature to fulfill all-to-all connections, but it is not good at supporting diverse $*$-cast traffic patterns because AWGR cannot provide an intrinsic $*$-cast characteristic. One may combine both of their advantages, thereby supporting $*$-cast traffic in HyperOXN$(1, 1, 1, n, 1)$ and all-to-all connections in HyperOXN$(1, n-1, 1, n, n)$, into a HyperOXN$(1, n, 1, n, n)$ with a higher cost, as depicted in Figure 3.15.

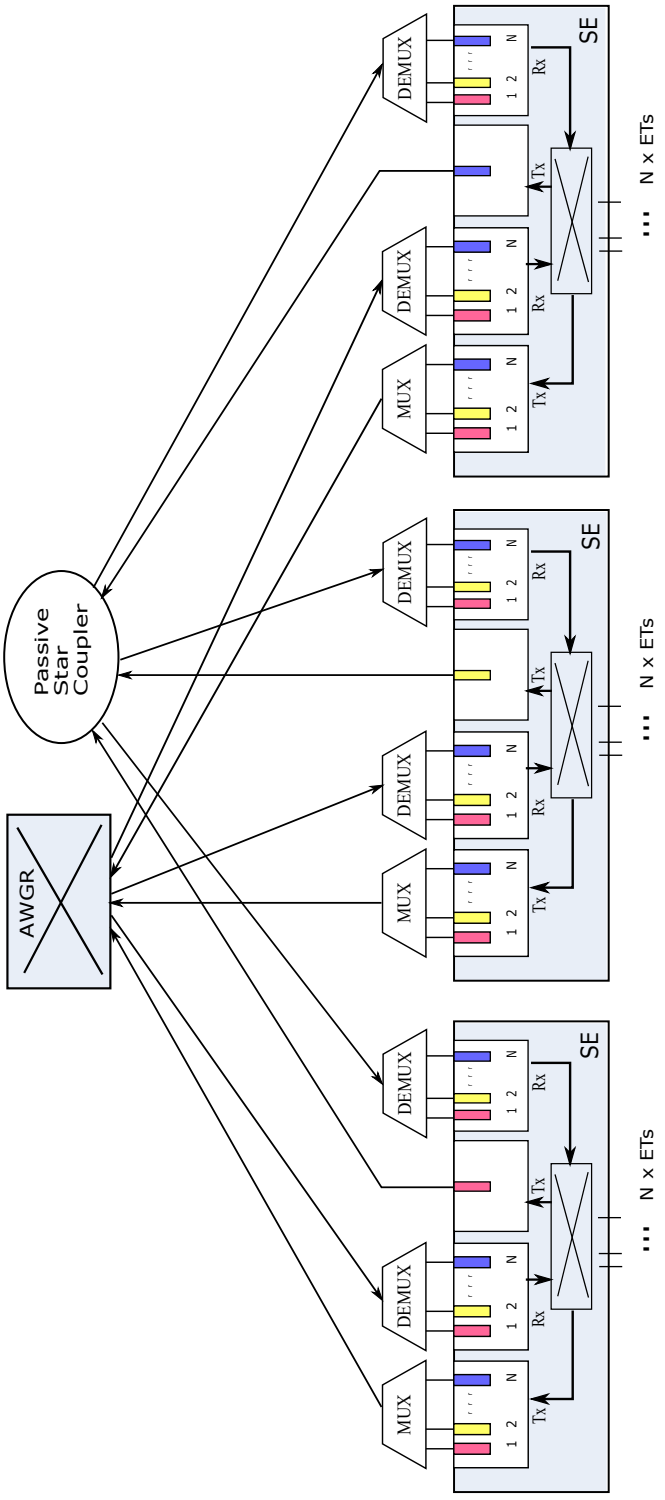


FIGURE 3.15: The block diagram for HyperOXN$(1, 1, 1, n, 1)$

In order to reduce the cost, one may build an aggregated $N : K$ broadcast-selection, instead of a full configuration of $N : N$ broadcast-and-selection. However, this introduces over-subscription, and a practical multiplexing method is necessary.

3.3 Analysis of Rearrangeable Non-blocking

Because the rearrangeability of multistage interconnect networks has been applied and studied extensively [145] [146] [147] [70], the regular HyperOXN topology is transformed into multistage interconnect networks topology, and the rearrangeability of the multistage interconnect network topology is analyzed in order to understand the rearrangeability of HyperOXN. Referring to the method used by Choi [148], the steps are as follows:

1. First, duplicate the vertices of the topology d times according to the d-dimensional 2-dilated HyperOXN topology (see Figure 3.1), and arrange the vertices of every dimension in each stage of the multistage interconnect networks topology. Then, duplicate the vertices of every stage of multistage interconnect networks topology two times as the input and output stage of per stage of the multistage interconnect networks topology. Finally, according to the means of connection, connect per input and output vertices of the multistage interconnect network topology as an R_i-stage, respectively. Through the above steps, a d bipartite graph can be formed.

2. Combine the output vertices of stage i and the input vertices of stage $i-1$ of the multistage interconnect network topology, $i \in [d-1, d-2, \cdots, 1]$, and then connect every vertex respectively. Define the formed d-stage multistage interconnect network topology as G_d.

3. Define the output vertex of the 0-stage into the axis of symmetry and make a mirror graph of G_d; by this, every ET of HyperOXN topology can denote two channels in every dimension. Define the mirror graph into G'_d.

Figure 3.1. can be transformed into a multistage interconnect network topology by the above three steps, as Figure 3.16.

The rearrangeablity of multistage interconnect networks has been well studied [146] [147] [31] [70]. The rearrangeablity of HyperOXN is explained into multistage

interconnect network, and then its rearrangeablity can be demonstrated. Using Choi's rearranging method [146], the procedure is as follows:

1. For a d-dimensional regular HyperOXN, duplicate each vertex in the topology d times where the duplicated vertex in each dimension serves as each stage in a multistage interconnect network. Then, duplicate each stage's vertex, serving as the input and output vertex for the stage. If one follows each HyperOXN dimension's connection pattern and connect each stage's input and output to form R_i, then, a d-bipartite graph is created.

2. Collapse stage-i's input vertex with the (i-1)-stage's output vertex for all i to form a single vertex. Define the newly formed n-stage multistage interconnect network as G_d.

3. Use the 0-stage's output vertex as the axis to create G_d's image graph. As a result, each HyperOXN's end ET's two channels in each dimension are represented. Define the image graph as G'_d.

Following the above three steps, Figure 3.1 and Figure 3.2 are transformed into a multistage interconnect network as shown in Figure 3.16.

Next, one shall prove that the multistage interconnect network transformed by the above procedure is rearrangeably non-blocking, which is equivalent to proving its rearrangeability.

Per $2l$-HyperOXN's definition, each vertex has $2l$ channels, and each ET communicates with other ETs in the same dimension using two channels. Accordingly, there are $l * n^d$ ETs on the topology. A permutation of these $l * n^d$ ETs is decomposed into an n^d l-permutation of the vertex, labeling as $p_0, p_1, \ldots, p_i, \ldots, p_{l-1}$.

1. When $d = 1$, the HyperOXN is only one dimensional. Following the above three-step transformation procedure, the HyperOXN can be transformed into a multistage interconnect network, as shown in Figure 3.16. Each vertex j is labeled as $0, 1, \ldots, n-1$. The following routing method is taken: for

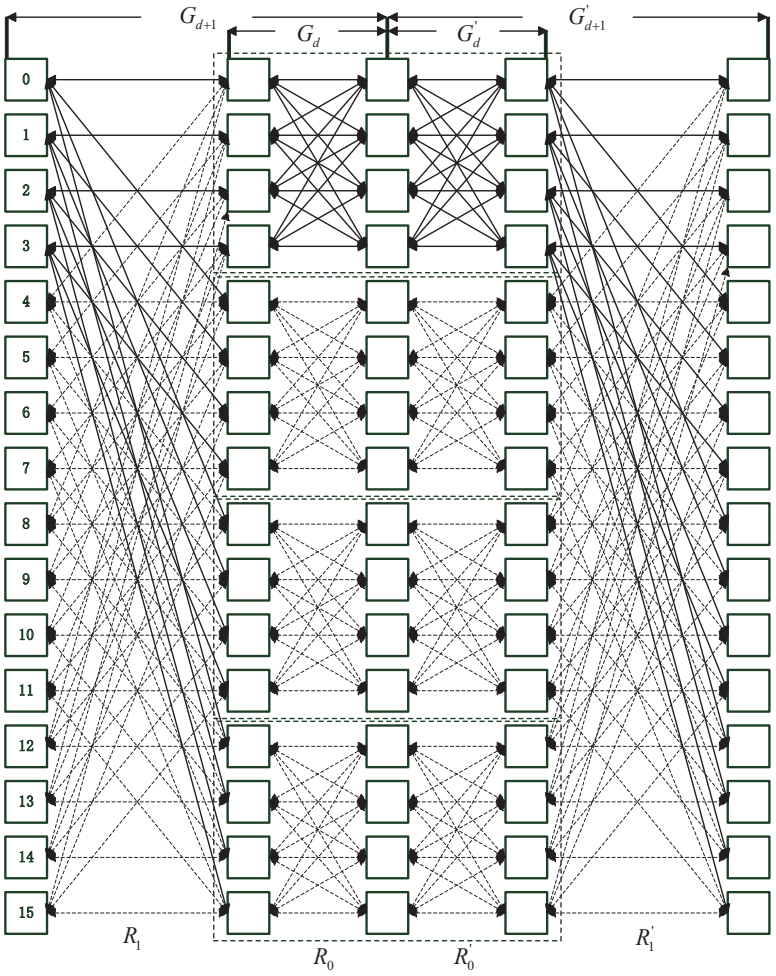


FIGURE 3.16: $2d$ multistage interconnect networks

any p_i permutation, input vertex j chooses any intermediate vertex M_m for routing, m is $[0, n-1]$ and has l possibilities. The intermediate vertex then reaches the output vertex through G'_d's associated connections, where there are l possibilities. Along with this process, the path of any p_i permutation's vertex j is non-blocking. Accordingly, it is known [70] that the transformed multi-stage interconnect network is rearrangeably non-blocking. It is that 1-dimensional n-vertex HyperOXN is of l-rearrangeability, and therefore the 1-dimensional HyperOXN with $l * n$ ETs is of 1-rearrangeability, called re-arrangeability for short.

2. Next, one shall prove that $d+1$ dimensional n^{d+1} vertex HyperOXN is of l-rearrangeablity, which is the same as to say $d+1$ dimensional $l * n^{d+1}$

ET is of rearrangeability. First, let us assume the d-dimensional n^d vertex HyperOXN is l-rearrangeable, where $d >= 1$. Therefore, the associated $2d$-stage multistage interconnect network is l-arrangable as well. According to HyperOXN's recursive property, the $d+1$ dimensional n^{d+1} vertex HyperOXN can be demonstrated by the $2*(d+1)$-stage multistage interconnect network using the above three-step transformation procedure. It is to increase the current $2*d$-stage multistage interconnect network by two dimensions. The detailed connection is illustrated in Figure 3.17. Every vertex in the figure in labeled as $0, 1, \ldots, n^{d+1}-1$. The following routing method is taken: For any p_i permutation, the input vertex j at stage-0 chooses any intermediate $2*d$-stage multistage interconnect network for routing. The intermediate $2*d$-stage interconnect network then reaches output vertex through the $2d+1$-stage multistage interconnect network. Along this process, there are l possibilities for an input vertex to reach any intermediate $2*d$-stage interconnect network, and there are l possibilities for an intermediate $2*d$-stage interconnect network to reach any output vertex. Each intermediate $2*d$-stage interconnect network is l-rearrangeable. Therefore, $2*d$-stage multistage interconnect network is also l-rearrangeable. That is, the d-dimensional n^d vertices 2-dilated HyperOXN is l-rearrangeable, which also means that the $d+1$ dimensional $l*n^{d+1}$ ETs are of rearrangeablity. This proves that the $2l$-HyperOXN is rearrageably nonblocking.

The above proof can show that each vertex can choose l permutations. Therefore, each vertex can connect l ETs without blocking. When $l = n$, the topology reaches maximum in size as n^{d+1}. In this proof, the number of vertices does not have to be 2's power or 2's multiples, as required in Szymanski's [76] study. It can be of any value, reducing the restriction on the vertex in DCNs, making it more convenient for DCN expansion.

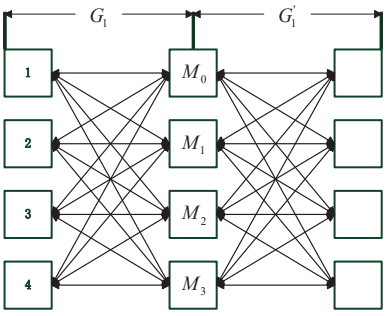


FIGURE 3.17: Two multistage interconnect networks

3.4 Analysis of Embedment of Topologies

The embedment property of topology refers that the nodes and edges of guest graph $G(V_1, E_1)$ are mapped one-to-one to the nodes and edges of the host graph $H(V_2, E_2)$ by the function b and d, as

(1) $b(u) \rightarrow v$, $u \in V_1$, $v \in V_2$;

(2) $d(u_1, u_2) \rightarrow (b(u_1), b(u_2))$, $(u_1, u_2) \in E_1$, $(b(u_1), b(u_2)) \in E_2$.

Considering the embedment property of a topology, the two main reasons are that: (1) Realize the algorithm of guest graph in host graph topology perfectly. (2) Great characters of host graph can optimize the algorithm of the embedded guest graph. Four properties, load, congestion, dilation and expansion, can evaluate whether the function of a topology is good or not [149]. Load refers to the number of nodes that guest graph is embedded into host graph, and congestion refers to the number of edges that guest graph is embedded into host graph. Dilation refers to the largest length that guest graph is embedded into host graph, and expansion refers to the rate of nodes that guest graph is embedded into host graph. Ted Szymanski proved that in the 2^n Hypercube, a guest graph with the condition of dilation k, congestion C and load L can be embedded perfectly with the condition of dilation $<= k$, congestion $C * log_2 N / log_d N$ and load L. In [150] a kind of labeling strategy was put forward to make the n-dimensional ($n \geq 2$) mesh skeleton be embedded optimally into a 2-dimensional hypermesh and finally generalize to n-dimensional hypermesh and not square hypermesh by the theory statement.

Because the HyperOXN topology replaces a one-to-one edge with a hyperedge and every SE can carry d ETs, the original definition of topology embedment does not correspond to the condition of the embedment of a HyperOXN topology and the new concept of topology definition should be expanded. The new definition of topology embedment is that the nodes and edges of guest graph $G(V_1, E_1)$ can be mapped into corresponding nodes and edges of host graph $H(V_2, E_2)$ by the function b and d. in other words:

(1) $b(u) \to v$, $u \in V_1$, $v\ in V_2$;

(2) $d(u_1, u_2) \to (b(u_1), b(u_2))$, $(u_1, u_2) \in E_1$, $(b(u_1), b(u_2)) \in E_2$.

The embedment of nodes and edges is not one-to-one and the k guest graph G can be embedded into the same position of host graph H at most, including the many-to-one mapping of nodes and edges. Consider the labeling strategy of the document and realize the embedment of guest graph into HyperOXN topology. Assuming the embedment of the guest graph of M nodes into the q-dimensional HyperOXN topology of N^q SEs and demanding N is the integer multiple of M, it should be demanded that the number of edges of the guest graph is the integer multiple of q. Because every SE can carry d ETs in HyperOXN, the q-dimensional HyperOXN of N^q SEs sums up $d * N^q$ nodes. According to the relationship of the number of nodes, $d * a * N^{q-1}$ guest graphs can be optimally embedded into HyperOXN, and labeled as G_0^0, G_0^1, ..., G_0^{d-1}, G_1^0, G_1^1, ..., G_1^{d-1}, ..., $G^0_{a*N^{q-1}-1}$, $G^1_{a*N^{q-1}-1}$, ..., $G^{d-1}_{a*N^{q-1}-1}$, respectively.

The embedment steps are as follows:

(1) Label M nodes of G_i^j as $0, 1, \ldots, M-1$, $i \in [0, a^q - 1]$, $j \in [0, d-1]$ randomly;

(2) Label edge of G_i^j as $0, 1, \ldots, q-1$ averagely; in other words, the number of every labeled edge is equal;

(3) Have the per coordinate of k-ary coordinate of HyperOXN mod M and then label them;

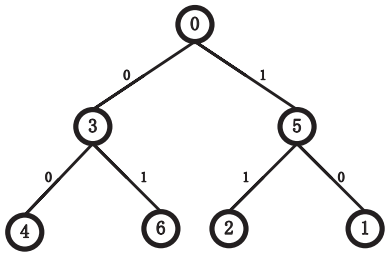


FIGURE 3.18: Tree topology

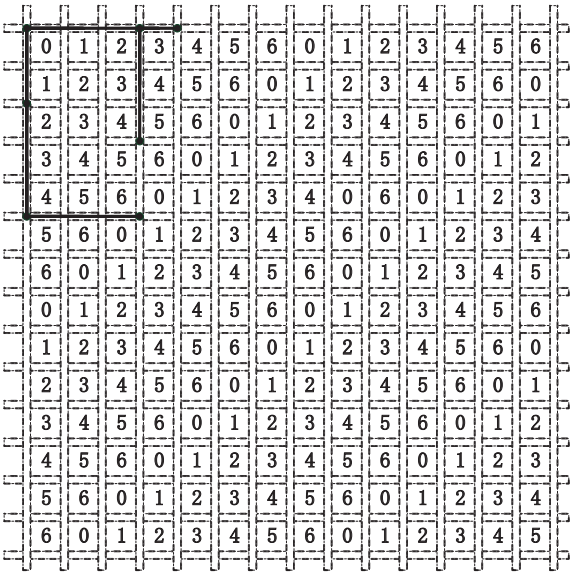


FIGURE 3.19: Embedment of HyperOXN topology

(4) Put the 0-labeled node of G_i^j into any position labeled 0 of HyperOXN. The G_i^j should be assigned into the same position labeled 0 when i is same in G_i^j;

(5) Assign the another $1, 2, \ldots, M-1$ node into the relative position combining the labeled dimension of edge and the next node following the former node as a benchmark. So far, the embedment of the $d * a * N^{q-1}$ guest graph of M node into HyperOXN is completed perfectly. The guest graph is considered as a complete binary tree of 7 nodes and the host graph as HyperOXN of 14^2 nodes and realize the result of embedment of $G_0^j (j \in [0, d-1])$ as shown in Figure 3.18 and Figure 3.19.

Through the above process of embedment, the number of realizing guest graph embedded into host graph is $14^2 * d$ and the number of edges is $168 * d$. After

being embedded, the length of every edge is 1 and the embedment rate of node is 1 to realize the optimal index of load, congestion, dilation and expansion. This method is for optimal embedment.

3.5 Routing algorithm for the HyperOXN

The set of shortest paths between two SEs in a HyperOXN is determined by their coordinate vectors. The k-slack shortest-path routing scheme is used in Chapter 5 combined with the HOLMES$^+$ flow control algorithm. To leverage massive multiply short paths efficiently in a HyperOXN, an edge (u, v) for a flow from s to t is only allowed to be chosen as routing satisfying conditions as below.

$$L_{slack} \leq L + k,$$

where L_{slack} is the length of the path for $u \rightarrow$...shortest path...$\rightarrow u \rightarrow v \rightarrow$... shortest path...$\rightarrow t$; L is the length of the shortest path from $s \rightarrow t$.

As HyperOXN$(d, 1, 1, n, 1)$ plays a significant role in the design, the following discussion about routing algorithms will accordingly focus on this type of HyperOXN topology.

3.5.1 Multi-routes for One-to-one Traffic

One can take advantage of multiple routes to do load balancing which reduces transmission time and increases availability.

One can construct one path between any two SEs in each dimension of a HyperOXN $(d, 1, 1, n, 1)$, and also those paths are edge-disjoint. As such, there are d edge-disjoint paths between any two SEs in a HyperOXN$(d, 1, 1, n, 1)$. The multiple paths can also be fully utilized to speed up one-to-one traffic. Edge-disjoint means that a hyperedge in one route does not appear in all the other ones.

It is worth noting that a $B\&S$-based hyperedge could be regarded as overkill if the traffic mode is only one-to-one. However, its intrinsic power becomes clearer when one considers $*$-cast traffic which is increasingly important for DCNs, especially for parallel applications.

3.5.2 Routing for In-cast Traffic

For in-cast traffic, each source has d parallel edge-disjoint paths to the sink, and all sources can leverage all these multiple paths independently at the same time. Unlike in the Fat-Tree topology, in-cast traffic does not arise any congestion in the network, while the receiver selectively decides to choose traffic from different senders under the $B\&S$ architecture.

3.5.3 Routing for Multicast Traffic

One can construct d edge-disjoint spanning trees to replicate multicast traffic. Edge-disjoint here means that a hyperedge in one spanning tree does not appear in all the other ones.

3.5.4 Routing for Broadcast Traffic

There are d edge-disjoint spanning trees to deliver broadcast traffic with a speed-up factor d.

3.5.5 Aggregate Bottleneck Throughput for A2A Traffic

As defined in [63], the aggregate bottleneck throughput (ABT) is the number of flows times the throughput of the bottleneck flow. ABT reflects the network capacity under the all-to-all (A2A) traffic pattern, under which every SE establishes a flow with all others in the network. In a HyperOXN$(d, 1, 1, n, 1)$, ABT

is $\frac{n}{n-1}(N-1)$, where N is the number of SEs. As a result, the ABT increases linearly as the number of SEs grows (see the proof in [63]).

3.6 Properties Comparisons

The evaluation of HyperOXN topology performance, latency, scalability, and cost is the primary factor to consider and compare. Since this research's focus is also on the topology of the rearrangeable non-blocking property, $\phi = 1$ is chosen for comparing for each topology. As a type of multistage interconnect network, Fat-Tree's rearrangeable non-blocking properties have been well proven. Hypercube topology's non-blocking characteristic has been verified, and the derived k-dilated k-way bristled Hypercube topology achieves better scalability while still having the non-blocking property. Based on the FBFLY topology, Thamarakuzhi [31] proposed the 2-dilated FBFLY (2DFB) topology and proved its rearrangeable non-blocking properties by describing the existence of a conflict-free static routing schedule. Szymanski [76] also proved indirectly that a 2-dilated Hypermesh has the non-blocking property. The two topologies mentioned above are compared with the HyperOXN topology and performance.

As a comprehensive model, Hypermesh, FBFLY, HyperX and Generalized Hypercube are special cases of HyperOXN as mentioned in Section 3.1, page 55. Fat-Tree topology has been intensively adopted in current product DCNs, and as such, in this section, a comparison between Fat-Tree and HyperOXN is undertaken, choosing $\phi = 1$.

3.6.1 Cost Comparison

3.6.1.1 Methodology

When comparing different novel topologies for DCNs, both performance and cost metrics must be considered. The performance metrics fall into two categories:

static and dynamic. By static analysis for specific topologies, some metrics such as latency (*e.g.*, the number of average hops between servers) and throughput (*e.g.*, bisection bandwidth or sparsest cut bandwidth) can be calculated mathematically or simulated by software tools. For simplicity, the static analysis method is explicitly independent of systems-level issues such as routing and congestion control, and does not consider scenarios with a variety of traffic workloads. However, translating performance in static analysis to that at the flow level or packet-level can be more challenging, and in particular, with dynamic and changing traffic workloads. For example, how does one evaluate the dynamic metric FCT (flow-completion time) of a topology under the worst-cast skew traffic? In this section, only static metrics will be measured between the Fat-Tree and HyperOXN and the detailed dynamic performance evaluation will be elaborated on in Chapters 4 and 7.

Accounting for cost, most of the focus has been on driving down the cost of IT equipment (*e.g.*, servers, switches, routers) in DCNs. However, power and cooling are not being improved at "Moore's Law" rates, and are improving far more slowly than that of IT equipment as aforementioned in Chapter 2 and on page 14. Thus, the cost of power and cooling infrastructure has been increasing at an alarming rate, and reached up to 45% in 2018. It turns out that power has become an incredibly important factor in building a large-scale DCN.

The total cost of ownership (TCO) of a Data Center facility is the sum of the initial capital expenditures (CAPEX) and ongoing long-term operational expenditures (OPEX). The capital costs of IT equipment (e.g., servers, switches, routers, cables) and infrastructure for power distribution and cooling comprise a significant fraction of the CAPEX. In general, infrastructure is typically meant to last for very long time periods, e.g., 10 years, and IT equipment is amortized over $3 \sim 5$ years. Accordingly, the power draw of IT equipment and infrastructure clearly has the largest impact on the OPEX. As a consequence, reducing the power draw of IT equipment can also save money on infrastructure for power delivery and heat evacuation.

The networking cost model in [151] is adopted throughout the following comparisons for different Data Center topologies. The networking capital expenditure consists of the following components: switches, NICs plugged in servers, cables and other interconnect devices such as POSCs and AWGRs. In addition, the power draw of networking equipment and its related infrastructure amortization needs to be considered in the comparisons. Other costs such as maintenance, management, etc., are neglected for due to their nontrivial quantification.

To estimate the total power draw for IT equipment and infrastructure, denoted as P_{total} in Watt, assume a reasonable utility price of ρ_1 per KWH, and infrastructure cost rate for cooling and power distribution of ρ_2 per Watt per year according to amortization over 10 years [58]. Thus,

$$P_{total} = P_{IT} + P_{infr} \tag{3.10}$$

and:

$$P_{infr} = (PUE - 1) * P_{IT} \tag{3.11}$$

The total cost of 3-year energy use denoted as C_{3total} including infrastructure amortization can be calculated as follows:

$$\begin{aligned} C_{3total} &= P_{total}/1000 * \rho_1 * 3 * 365 * 24 + P_{total} * \rho_2 * 3 \\ C_{total} &= P_{total}/1000 * \rho_1 * 3 * 365 * 24 \end{aligned} \tag{3.12}$$

where PUE (Power Usage Effectiveness) is the ratio of the total power of the facilities of the Data Center over the amount of energy used by the IT equipment. The total facility power is the energy delivered to operate a Data Center, including power for operating the IT equipment and the cooling infrastructure. For example, a PUE of 1.5 means that for every Watt of energy spent on IT equipment, 0.5 Watt is spent for cooling and power distribution. $P_{total} * \rho_2$ stands for yearly infrastructure amortization of the power draw of P_{total}.

It is worth noting that in many past designs [24] [136] [152] seeking to demonstrate cost advantages of FBFLY over Fat-Tree, the corresponding results are unfair for

cost comparisons using FBFLYs with $\phi = 0.5$ while Fat-Trees have $\phi = 1$.

A n-ary d-level Fat-Tree with the over-subscription ϕ is constructed using a tapered folded Clos [6]. First, fix the amount of uplinks and downlinks at level 1 according to ϕ, and then build a full-bandwidth Fat-Tree for $\phi \cdot N$ ETs.

3.6.1.2 Conventional Network Interface Card

The power consumption and cost breakdown for current network interface cards (NICs) plugged in servers is shown in Table 3.5. Unlike switches, the cost of an NIC's electronic part is shared by only one or few ports, and is 4-8 times more expensive than the transceiver's cost. However, following the trend mentioned above, the cost ratio, denoted as δ_5, of one transceiver to one NIC electronic port is assumed as $1:2$ for a NIC with 1-4 ports and $1:1$ for a more-than-five-port NIC. Also, assume that the power ratio follows δ_3 (see Section 3.2.5.2, page 73) throughout the following comparisons, while neglecting the cost of CPU cores in servers.

Data-rate	10G (SPF+)	100G (SPF+)
Photonic Transceiver Power (W)	1.2	2.5
Photonic Transceiver Cost ($)	10	100
Electronic Power (W)	6-8	19
Electronic Cost ($)	80	350

TABLE 3.5: Power consumption and cost breakdown for a NIC

3.6.1.3 Cabling Cost

For simplicity, assume that the average length of each intra-rack and inter-rack cable is 300 meters with a price of 0.3$/meter for 10G SFP+ optical cables, as used in ProjecToR [79]. An n-ary d-level Fat-Tree with $n/2$-port switches has $(n/2)^3$ switch-to-switch cables.

3.6.1.4 Total Cost (TOC) Comparisons

The following assumptions are made throughout all the comparisons with 10G port throughput:

Assuming an average PUE of 1.45 [58] and an average industrial electricity rate of \$0.1 per kWh [151], the power consumption can be converted to cost.

It is worth noting that when comparing different topologies, a fixed bisection bandwidth and a constant number of servers which each topology needs to support will be assumed in the following. Across all experiments, we test a wide range of parameters, varying the network size, node degree, and oversubscription.

All comparisons between topologies are made using identical r-radix switches, unless noted otherwise. The evaluated interconnect networks are also assumed as fully burdened for power consumption.The following parameters are used in the comparisons:

- δ_1 (Section 3.2.5.1) = 0.1;
- δ_2 (Section 3.2.5.1) = 1;
- δ_3 (Section 3.2.5.2) = 10;
- δ_4 (Section 3.2.5.2) = 4;
- δ_5 (Section 3.6.1.2) = 0.5;
- The cost for a 10G SPF+ transceiver = \$10.

Figures 3.20 and 3.21 show that the cost of 3-year energy use dominates the overall costs and grow higher in Fat-Tree or HyperOXN topologies for larger DCNs. The vertical axis of the figures depicts the relative cost normalized by the network size (total ETs) with or without power consumption, while the horizontal axis stands for the network size. The metric for power consumption will play a significant role in the design of large-scale Data Centers.

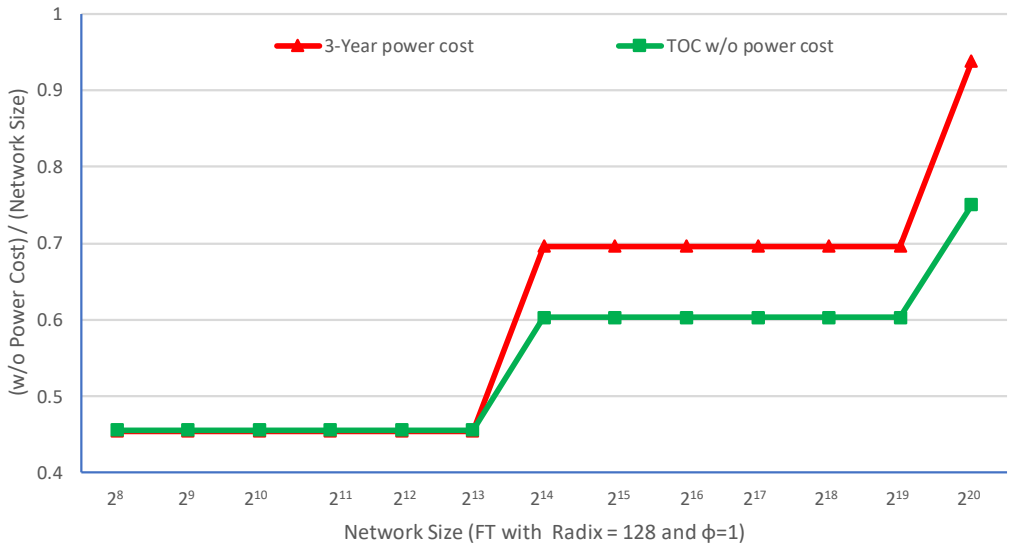


FIGURE 3.20: Cost w/o power consumption for Fat-Tree

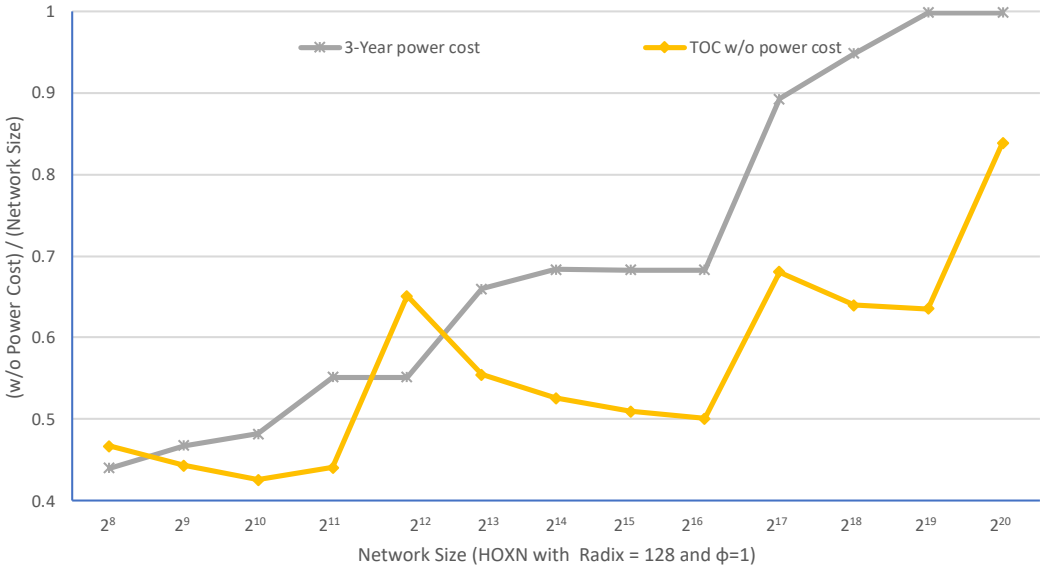


FIGURE 3.21: Cost w/o power consumption for HyperOXN

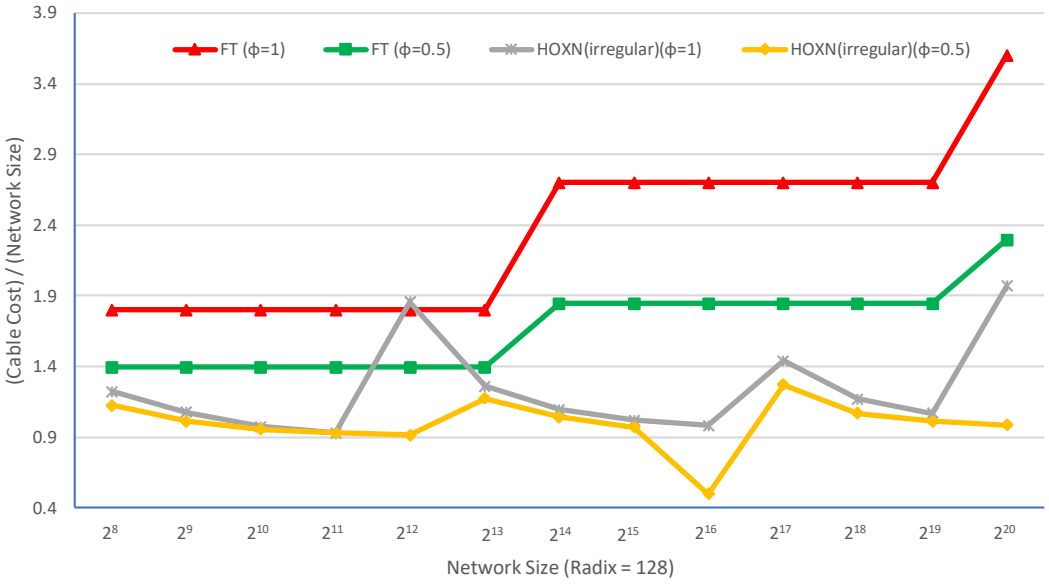


FIGURE 3.22: Cabling cost comparison

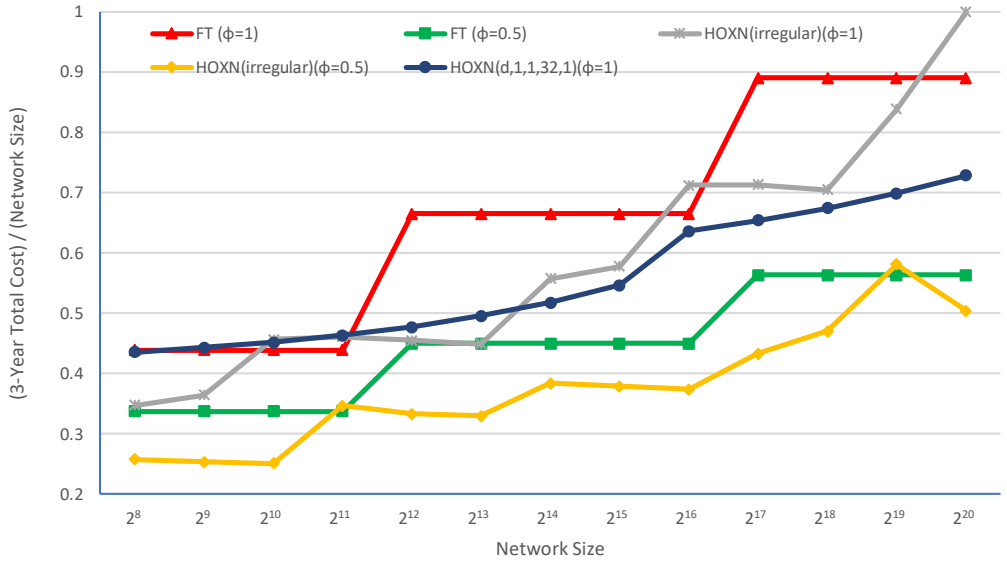


FIGURE 3.23: 3-year overall cost comparison (Radix $r = 64$)

In Figure 3.22, with varied network size, oversubscription ratio ϕ, and fixed switch radix $r = 128$, HyperOXN uses much fewer cables compared with Fat-Tree topologies supporting the same number of ETs. Passive optical DWDM technologies in HyperOXN reduce the cabling complexity and also achieve lower cost.

As shown in 3.24, HyperOXN achieves cost gains over Fat-Trees with higher radix

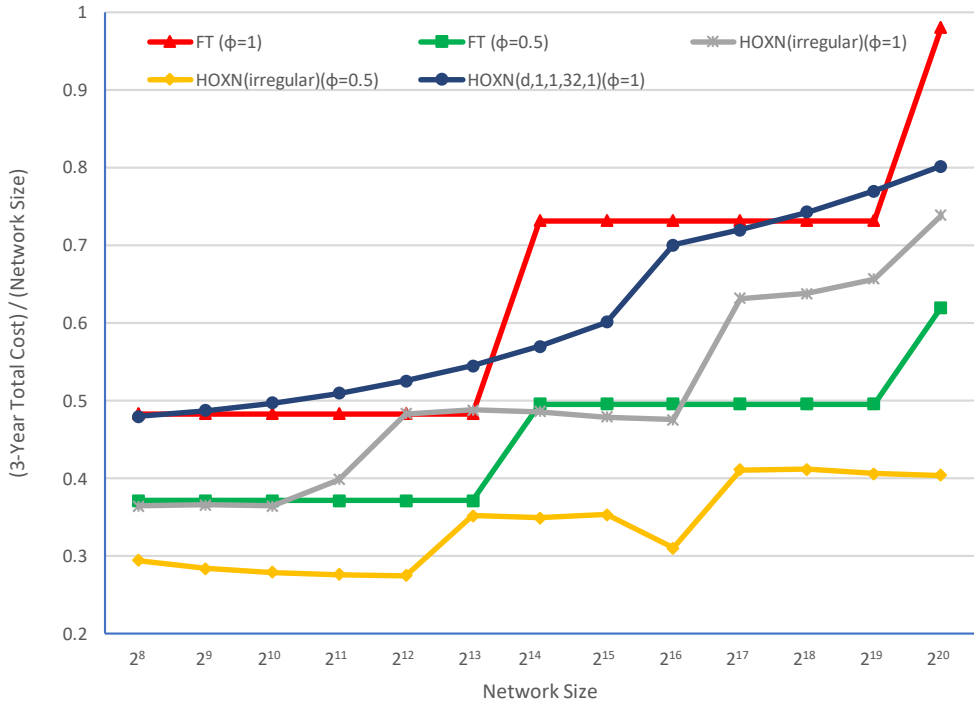


FIGURE 3.24: 3-year overall cost comparison (Radix $r = 128$)

128 while supporting the same number of network size. But the cost of regular HyperOXN$(d, 1, 1, 32, 1)$, which is supports $*$-cast traffic well, is much closer to that of a Fat-Tree under the same oversubscription $\phi = 1$ and radix 128. In a smaller radix 64, HyperOXN exceeds the Fat-Tree's cost just by a small margin. This implies that HyperOXN is better suited for large-scale DCNs at 15-30% lower cost.

3.6.2 Scalability

For the study of typology HyperOXN design, first, it is approached primarily by looking at the number of links, ignoring the number of switches for the moment. The hyperedge needs to consider implementation aspect in more significant details. A hyperedge needs multiple channels for signal transmission. A single electrical link cannot meet the requirement, while numerous electrical links dramatically increase complexity. On the other hand, as optical components are widely adopted

in communications applications, the cost approaches the cost of electrical components. With consideration of the above factors, a POSC or AWGR is chosen for hyperedge implementation. HyperOXN introduces the hyperedge concept, and there is an considerable advantage in the number of links with an order of magnitude less than the others.

Considering the switch factor, the ratio is defined as follows,

$$\gamma = \frac{Number\ of\ Switches}{Number\ of\ Terminals} \tag{3.13}$$

which is measured as an approximate estimation of network cost per ET.

The optimum topologies are constructed respectively for HyperOXN($c = 1$), Fat-Tree and HyperOXN($c = n - 1$) to satisfy three constraints: total ETs T, radix $r = 256$, and $\phi = 1$. As shown in Figure 3.25, HyperOXN($c = n - 1$) is very comparable to Fat-Tree, while the cost for HyperOXN(c=1) is the lowest. Primarily HyperOXN($c = 1$) is more compelling due to its direct topology, which can be integrated into servers such that the cost is reduced.

Scalability is another important metric when measuring the number of ETs a topology can support under a given dimension. Higher scalability allows large-scale Data Centers without impacting other performance metrics. Both Fat-Tree and HyperOXN can meet the requirement for network size growth. As shown in Figure 3.25 and Figure 3.26, HyperOXN can increase more efficient at lower cost and low latency because HyperOXN is more flexible in optimizing the design space, while Fat-Tree has to add another complete layer of switches to increase the number of ETs. In terms of the aforementioned drawbacks of Fat-Trees, the performance degrades for random traffic. The detailed simulation results will be presented in Chapter 4.

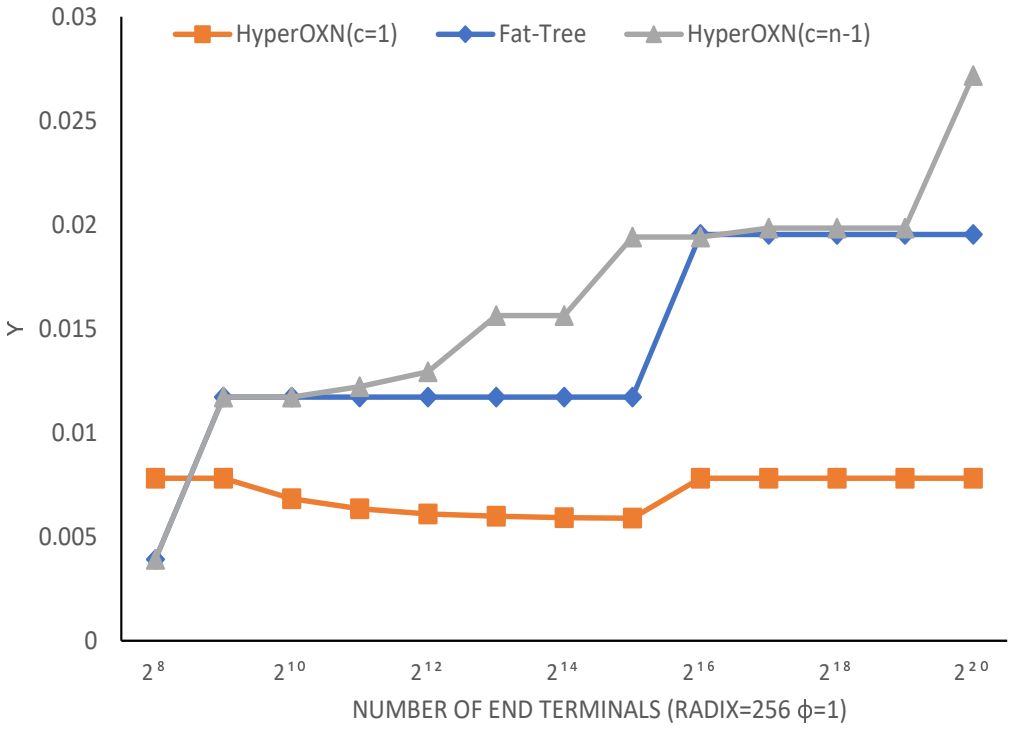


FIGURE 3.25: The number of switches per end-terminal

3.6.3 Latency

The latency of DCNs is primarily determined by the number of links traversed by any routing request. The number of hops are measured for the average packet traversed paths.

As introduced in [153], a n-ary d-level Fat-Tree is an indirect interconnect network based on recursively constructed folded-Clos topology with FBB $\phi = 1$. As such, it is more economical for worst-case traffic, but for random traffic, Fat-Tree may miss half capacity as mentioned in [24]. There are $T = 2{\cdot}n^d$ ETs and $P = (2d{-}1){*}n^{d-1}$ identical SEs, each of which is $n \times n$ switch with radix $2n$. For a n-ary d-level Fat-Tree, the distance between any two communicating ETS should be $2i$ with $1 \leq i \leq d$. From an arbitray ET, there exist $n-1$ ETs with a distance two, $n(n-1)$ ETs with a distance four, ..., and finally $(n-1)n^{d-1}$ ETs at a distance $2d$. Thus, the average distance of a n-ary d-level Fat-Tree can be obtained as

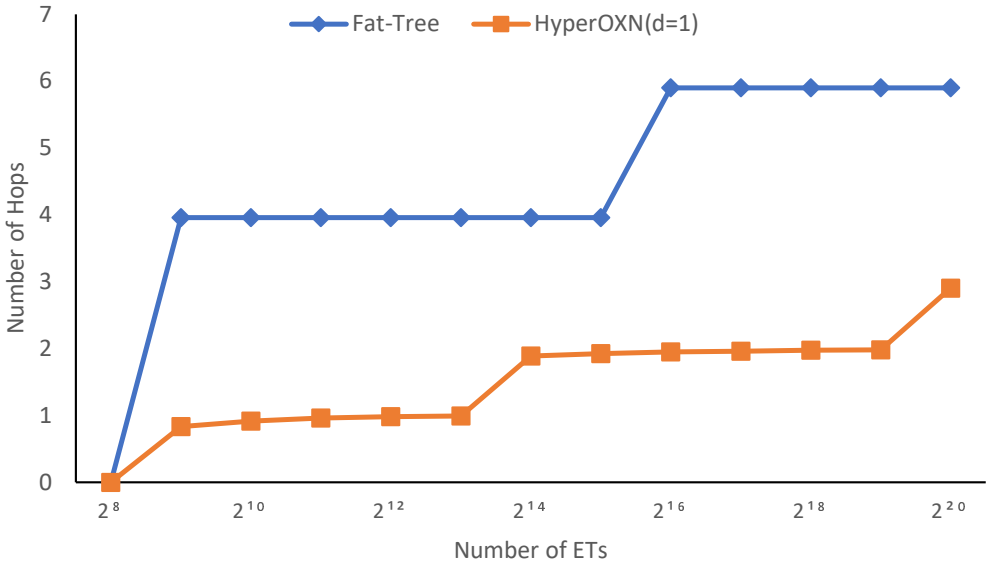


FIGURE 3.26: The Number of hops(Radix $r = 256$, $\phi = 1$)

$$\theta = \frac{\sum_{i=1}^{d} 2i(n-1)n^{i-1}}{T-1} \tag{3.14}$$

For instance, $\theta = 7.125$ for a 4-ary 4-level Fat-Tree. A Fat-Tree can be scaled with same low radix switches to any large networks by increasing the number of levels, but it is not amenable to incremental growth. Besides, the huge amount of cables between the edge and core layers in an Fat-Tree has to be added to the cost for installation and maintenance.

As shown in Figure 3.26, HyperOXN's latency performance is much better than Fat-Tree, in particular, when N gets bigger. From (3.1), it is shown that the average distance in HyperXON is roughly equal to d, the number of dimensions. In the meantime, Fat-Tree's latency is nearly double that of HyperOXN, and Fat-Tree's average distance is very close to the network diameter as in (3.14), *i.e.*, the number of level of the network.

3.7 Summary

In summary, overall, the HyperOXN topology shows significant advantages in latency, scalability, and cost; this makes a compelling case for HyperOXN. Compared to Fat-Tree, it is more suitable for future large-scale Data Center needs.

Chapter 4

HyperOXN Performance Evaluation: Flow-Level Simulation

There are four aspects to evaluating a DCN's performance. These are the topology, routing, congestion control, and load balancing. The topology plays a significant role in data center networks and sets limits on performance metrics on DCNs. To measure a topology intrinsically and neglect any inefficiencies from routing and flow control algorithms, Section 4.1 is focused on the network topology evaluation under flow splitting optimal routing. Each flow can be split into sub-flows by one source and may be routed along multiple paths to a destination. As such, given particular traffic patterns, a multi-flows linear programming optimization is employed to maximize the throughput of the topology. In particular, for the performance of $*$-cast traffic, the information flow model based on network coding is chosen to do the evaluation. The information flow model facilitates achieving the optimal throughput of the topology.

Most switches and routers function as storing and forwarding packets from an input link to an output link in DCNs. According to the network information flow model [154], a route or switch can function as an encoder which receives

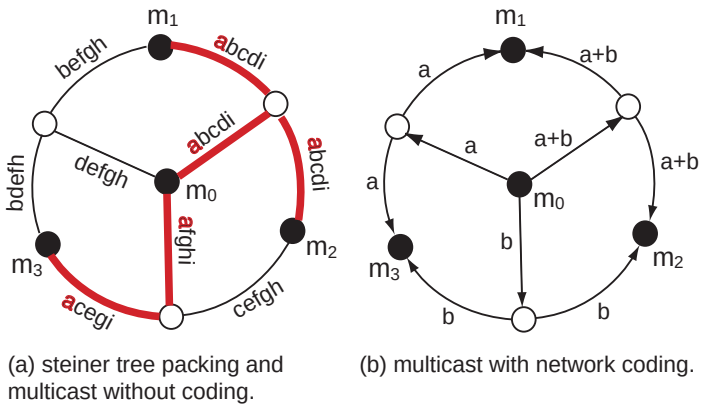


FIGURE 4.1: The achievable optimal throughput is 1.8 without coding, and 2 with coding [17].

information from the input links, and sends *encoded* information to the output links. As shown in Figure 4.2, node 2 can send out packets of the XOR of ones from node 1 and 3. Destination d_1 recovers a packet s_2 by taking the XOR of s_1 and $s_1 \oplus s_2$, and likewise destination d_2 can recover s_1 by taking the XOR of b_1 and $s_1 \oplus s_2$.

Although Steiner Tree Packing [155] has been a state-of-the-art algorithm to compute the optimal throughput of multicast communication sessions, it is an NP-hard problem which cannot be computed efficiently in polynomial time. For the case of information ows in one multicast session, it can be theoretically shown that achieving optimal throughput via multiple multicast trees is equivalent to the problem of Steiner tree packing if coding is not considered. Steiner tree packing seeks to nd the maximum number of pairwise edge-disjoint Steiner trees, in each of which the multicast group remains connected. In Figure 4.1(a), each letter represents a distinct Steiner tree, and nine such Steiner trees (a to i) exist in the shown packing scheme, where the tree corresponding to a is highlighted. Since each link with unit capacity needs to accommodate 5 Steiner trees, the achievable throughput on each tree is 0.2. This leads to a multicast throughput of 1.8, which is optimal without coding.

Network Coding can be used to maximize information flow rates in unicast or multicast sessions. Also, computing the strategies to achieve the maximum throughput

can be performed in polynomial time.

In an un-directed network as shown in Figure 4.2, each link has one unit capacity, and source s sends multicast sessions to destinations d_1 and d_2. The tree in the red dashed line in Figure 4.2 is denoted as $(s, 1, ((d1), (2, 4, ((d1), (d2)))))$. As shown in Table 4.1, the maximum achievable throughput with coding is 2 with network Coding. Without flow splitting, there are only two Steiner trees achieving maximum uncoded throughput 1, each of which has one unit capacity. Assuming half-integral flow splitting, the optimal uncoded throughput 1.5 can be achieved by four Steiner trees, each with a 0.5 unit capacity. With arbitrary flow splitting, one can achieve uncoded throughput of 1.875 using nine Steiner trees: the first six and the last three in red in Table 4.1 each having a capacity of 0.25 and 0.125 respectively. It is worth noting that the maximum throughput the Steiner tree algorithm achieved is much closer to what Network Coding achieves.

Most importantly, as proved in [154], for a multi-cast session, if a rate can be achieved from the sender to each of the multi-cast receivers independently, it can also be achieved for the entire multi-cast session. Li [17] observed that applying Network Coding may not improve the throughput for practical applications, but it can theoretically facilitate the optimal throughput of the topology.

To the best of the authors knowledge, this work is the first to use the network information flow model based on Network Coding to evaluate the throughput of multicast or unicast sessions in different topologies. In this chapter, HyperOXN relying on graph product is compared with Fat-Tree used dominating in today's DCNs, Xpander(based on lifting graph operation), and ProjecTor(dynamic architecture).

The experiments in Section 7.1 (Page 202) use packet-level simulation with routing and flow congestion conrol to target the optimal results in Section 4.1.

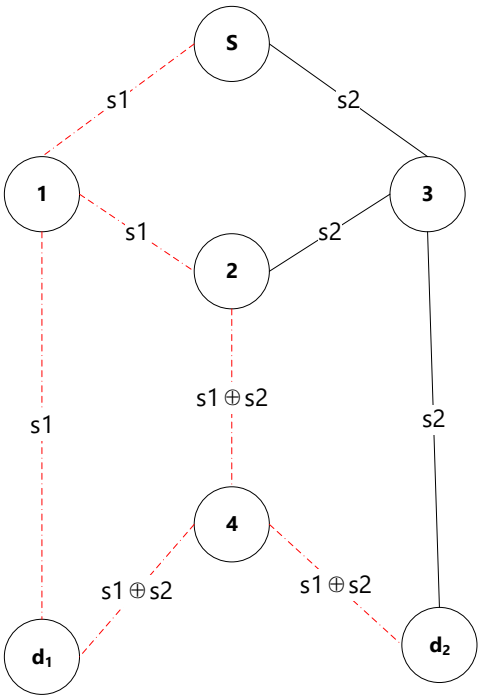


FIGURE 4.2: The maximum achievable multicast throughput with Network Coding

4.1 Flow-level simulation

Inspired by Topobench [19], considering only uni-cast splitting flows based on a fluid model, a simulation platform is developed based on an information model bearing $*$-cast traffic.

Given a directed topology $G(V, E)$ with an input Traffic Matrix (TM) representing the volumes of traffic from sources to destinations in a DCN, the optimal throughput of multicast sessions needs to be considered. The throughput of a network G with a traffic matrix T is the maximum value χ for which $\chi \cdot T$ is feasible in G.

To leverage the power of Network Coding to resolve competition for link capacities, the conceptual flow ($cFlow$) was introduced as in [17] for $*$-cast traffic. The cFlows from sources to sinks are defined as network flows that co-exist in the network but which do not contend for link capacities within the same session.

	Algorithm	Flow Splitting	Trees	Throughput
1	Steiner tree	No flow splitting allowed	$(s, 1, d1)$	1
			$(s, 3, d2)$	
2	Steiner tree	Half-integral flow splitting	$(s, 1, d1, 4, d2)$	1.5
			$(s, 1, d1)$	
			$(s, 3, d2)$	
			$(s, 3, d2, 4, d1)$	
3	Steiner tree	Arbitrary flow splitting	$(s, 1, ((d1), (2, 3, d2)))$	1.875
			$(s, 3, 2, 4, ((d1), (d2)))$	
			$(s, 1, 2, 4, ((d1), (d2)))$	
			$(s, 3, ((2, 1, d1), (d2)))$	
			$(s, 1, ((d1), (2, 4, d2)))$	
			$(s, 3, ((2, 4, d1), (d2)))$	
			$(s, 1, d1, 4, d2)$	
			$(s, 1, d1), (d2)))$	
			$(s, 3, d2, 4, d1)$	
4	Network Coding	Flow XOR coding	$(s, 1, ((d1), (2, 4, ((d1), (d2)))))$	2
			$(s, 3, ((d2), (2, 4, ((d1), (d2)))))$	

TABLE 4.1: Multicast throughput with different algorithms

The maximum χ needs to be solved for which there exists a feasible multicommodity flow that routes flow χ. $F^{s,t}$ through the network from each s to t, subject to the link capacity and the flow conservation constraints. The following linear programming optimization problem in (4.1) is used to get the optimal throughput, the scalar χ. $F^{s,t}$ is a cFlow from source s to each destination t. The below formulation ensures that all flows in the given TM can be satisfied at the desired rates specified by the TM, all scaled by a constant factor. Symbols used in the flow linear programming are listed in Table 4.2.

In the linear programming problem in (4.1), all flows including network flows $f^{s,t}$ and cFlows $F^{s,t}$ shall satisfy the following six constraints:

1. Flow fairness constraints: each flow is reduced equally such that there is proportionality towards the entire TM.

2. Equal rate constraints: all the conceptual flows are identical in a $*$-cast session.

3. The cFlows within the same session do not compete for link capacity, and the effective flow rate within a session mid on link (i, j) is

$$F^{s}_{mid,i,j} = \max_{t \in T_{mid}} (F^{s,t}_{mid,i,j}), \tag{4.1}$$

which can be relaxed to

$$F^{s}_{mid,i,j} \geq F^{s,t}_{mid,i,j} \tag{4.2}$$

Compared to non-linear (4.1), (4.2) can be handled easily by linear programming tools.

4. Link capacity constraints: flow rates shall be upper bounded by the link capacities.

5. Flow conservation constraints at intermediate nodes: all flows entering an intermediate node shall also leave it.

6. Flow conservation constraints at sources and sinks: the incoming flow rate at the source and the outgoing flow rates at the sinks are all zero.

χ	Scalar of the common weighted throughput for all traffic
$TM_{s,t}$	Throughput of the switch level uni-cast traffic matrix(TM) from source s to sink t
TM_{mid}	Throughput of the switch level for a $*$-cast flow (session) with identifier mid
N	Total number of nodes $N = \|V\|$
N_u	Number of uni-cast flows in the TM
N_m	Number of $*$-cast flows in the TM_{mid}
N_{mid}	Number of receivers in a $*$-cast flow with identifier mid
T_{mid}	The set of all receivers in a $*$-cast flow with identifier mid
$f^{s,t}_{uid,i,j}$	Rate for uni-cast flow from s to t with identifier uid going over link (i, j)
$F^{s,t}_{mid,i,j}$	Rate for $*$-cast conceptual flow from s to t with identifier mid going over link (i, j)
$F^{s}_{mid,i,j}$	Maximal rate of all conceptual flows in a $*$-cast session with identifier mid on the link from i to j
$C_{i,j}$	Link (i, j) capacity
$f^{s,t}_{uid}$	Rate for uni-cast flow from s to t with identifier uid
$l^{s,t}_{i,j}$	Rate of flow originating from node s going over edge (i, j) towards destination node t

TABLE 4.2: Symbols used in flow linear program

In this section, the evaluation results are presented for synthetic workloads and near-worst-case TM respectively.

4.2 Fair Flow Linear Programming

Fair flow linear programming (FFLP) tracks individual flows, which is particularly suited for sparse switch-level traffic matrices (e.g., all-to-one). To compute the optimal throughput, χ, the FFLP is formulated as follows.

Maximize: χ

Subject to:

(1) *Flow fairness constraints:* all flows $f^{(s,t)}$ and $F^{(s,t)}$ are reduced equally such that there is proportionality towards the entire TM:

$$\sum_{(s,j)\in E} f^{s,t}_{uid,s,j} \geq \chi \times TM_{s,t},\ \forall uid \in \{1,\cdots,N_u\} \tag{4.3}$$

$$\sum_{(s,j)\in E} F^{s,t}_{mid,s,j} \geq \chi \times TM_{mid},\ \forall mid \in \{1,\cdots,N_m\},\ \forall t \in T_{mid} \tag{4.4}$$

(2) *Equal rate constraints:* all the conceptual flows $F^{(s,t)}$are identical in a $*$-cast session:

$$\sum_{(j,t_1)\in E} F^{s,t_1}_{mid,j,t_1} = \sum_{(j,t_2)\in E} F^{s,t_2}_{mid,j,t_2},\ \ mid \in \{1,\cdots,N_m\},\ \forall t_1,\forall t_2 \in T_{mid} \tag{4.5}$$

(3) *Link capacity constraints:* every link is only capable of handling a certain amount of flow. The sum of rates for all flows along a link (i,j) is equal or less than the link capacity $C_{i,j}$:

$$\sum_{uid=1}^{N_u} f^{s,t}_{uid,i,j} + \sum_{mid=1}^{N_m} F^{s}_{mid,i,j} \leq C_{i,j},\ \forall (i,j) \in E \tag{4.6}$$

(4) *Flow conservation constraints:* at intermediate nodes, the amount of flow going from s to t plus the amount of flow coming into s from all other nodes towards t should be equal to the amount of flow going out of s:

$$\sum_{(j,i)\in E} f^{s,t}_{uid,j,i} = \sum_{(i,j)\in E} f^{s,t}_{uid,i,j},\ \forall uid \in \{1,\cdots,N_u\}, i \neq s, i \neq t \tag{4.7}$$

$$\sum_{(j,i)\in E} F^{s,t}_{mid,j,i} = \sum_{(i,j)\in E} F^{s,t}_{mid,i,j},\ \forall mid \in \{1,\cdots,N_m\}, \forall t \in T_{mid}, i \neq s, i \neq t \tag{4.8}$$

(5) *Flow conservation constraints:* at source s and sink t, the incoming flow rate $f^{s,t}_{uid,i,s}$ and $F^{s,t}_{mid,i,s}$ at the source s and the outgoing flow rates $f^{s,t}_{uid,t,i}$ and $F^{s,t}_{mid,t,i}$ at the sinks t are all zero:

$$\sum_{(i,s)\in E} f^{s,t}_{uid,i,s} = 0,\ \sum_{(t,i)\in E} f^{s,t}_{uid,t,i} = 0,\ \forall uid \in \{1,\cdots,N_u\} \tag{4.9}$$

$$\sum_{(i,s)\in E} F^{s,t}_{mid,i,s} = 0,\ \sum_{(t,i)\in E} F^{s,t}_{mid,t,i} = 0,\ \forall mid \in \{1,\cdots,N_m\},\ \forall t \in T_{mid} \tag{4.10}$$

(6) *All variables in above equations should be equal or great than zero:*

$$f^{s,t}_{uid,i,j} \geq 0,\ \forall (i,j) \in E,\ \forall uid \in \{1,\cdots,N_u\}, \tag{4.11}$$

$$F^{s,t}_{mid,i,j} \geq 0,\ F^{s}_{mid,i,j} \geq 0,\ \forall (i,j) \in E,\ \forall mid \in \{1,\cdots,N_m\}, t \in T_{mid}. \tag{4.12}$$

4.3 Fair Condensed Linear Programming

Fair condensed linear programming (FCLP) does not track how much of each flow goes over a link, but rather condensed flows. This makes it particularly suitable for dense switch-level traffic matrices (e.g., all-to-all), in which case keeping track of the contributions of individual flows would cause an explosion in the number

of variables. With regard to $*$ sessions, the same constraints as (4.2), (4.4), (4.5), (4.8) and (4.10) in the above FFLP should be satisfied. The FCLP is defined as follows.

Maximize: χ

Subject to:

(1) *Flow fairness constraints:* all flows $f^{(s,t)}$ and $F^{(s,t)}$ are reduced equally such that there is proportionality towards the entire TM:

$$f_{uid}^{s,t} \geq \chi \times TM_{s,t},\ \forall uid \in \{1, \cdots, N_u\} \tag{4.13}$$

(2) *Link capacity constraints:* every link is only capable of handling a certain amount of flow. The sum of rates for all flows along a link (i, j) is equal or less than the link capacity $C_{i,j}$:

$$\sum_{k=1}^{N} l_{i,j}^{i,k} + \sum_{mid=1}^{N_m} F_{mid,i,j}^{s} \leq C_{i,j},\ \forall (i,j) \in E \tag{4.14}$$

(3) *Flow conservation constraints for distinct pairs:* the amount of flow going from s to t plus the amount of flow coming into s from all other nodes towards t should be equal to the amount of flow going out of s.

$$f_{uid}^{s,t} + \sum_{(j,s)\in E} l_{j,s}^{j,t} = \sum_{(s,i)\in E} l_{s,i}^{s,t},\quad s \neq t \tag{4.15}$$

(4) *Flow conservation constraints for individual nodes:* the amount of flow going into destionation t should be equal to the amount of flow coming from its edges towards it.

$$\sum_{(s,t)\in V\times V} f_{uid}^{s,t} = \sum_{(j,t)\in E} l_{j,t}^{j,t},\ \forall s \in V,\ s \neq t \tag{4.16}$$

(5) *All variables in above equations should be equal or great than zero:*

$$f_{uid}^{s,t} \geq 0,\ l_{i,j}^{i,t} \geq 0, \forall (i,j) \in E,\ \forall uid \in \{1, \cdots, N_u\}, \tag{4.17}$$

$$F^{s,t}_{mid,i,j} \geq 0,\ F^{s}_{mid,i,j} \geq 0,\ \forall (i,j) \in E,\ \forall mid \in \{1, \cdots, N_m\},\ t \in T_{mid}. \tag{4.18}$$

4.3.1 Traffic Patterns

Similar to [19] but mixing with $*$-cast traffic patterns, four near-worst-case TMs are used to do the evaluation: All to All ($A2A$), Random Matching (RTM), Longest Matching (LTM) and multicast TM as follows:

1. All to All TM ($A2A$): An all-to-all fraction of communicating nodes means that within the fraction of all nodes communicate with each other concurrently. In the all-to-all multicast traffic pattern, each node sends the same message to every other node in the system.

2. Random Matching TM (RTM): Random matching means that there are only one outgoing flow and one incoming flow per server, chosen uniform-randomly among servers. Random Matching tries to simulate a few elephant flows coming from participating servers.

3. Longest Matching TM (LTM): Given a topology with diameter D, all server pairs are chosen to communicate with each other based on the shortest paths. The server pairs with the maximum shortest path length are chosen in the traffic matrix, which is called the longest matching TM approaching the near-worse case traffic.

4. Multicast TM (STM): The multicast group size distribution is set to be uniformly distributed in entire TORs as same in [156] from IBMWebSphere Virtual Enterprise. The amount of groups is fixed at one-fourth of total TORs.

4.3.2 Settings and Methodology

Three topologies including Fat-Tree ($\phi = 1$), Random (Jellyfish), and HyperOXN ($\phi = 1$) with $k = 128$ radix switches are evaluated for comparisons. The state-of-the-art LP solver, Gurobi[157] is used in all simulations. As Fat-Tree and HyperOXN can be built with particular discrete numbers of servers, switches and links which do not match, the methodology in [19] is adopted as follows. The random topologies like Jellyfish and Expander can be constructed for any size and degree distribution. Jellyfish and Expander are verified that both of them achieve identical performance [19]. The approach for evaluating a topology G follows four steps, as below:

1. Compute the topology's throughput using the above linear programming.
2. Construct a Random with the same hardware resources, *i.e.*, the same number of identical switches and servers as in the corresponding original topology.
3. Calculate the throughput of the Random under the same TM.
4. Normalize the topology's throughput with the Random's throughput for comparison against other topologies. This normalized value is defined as *relative throughput*.

4.3.3 Results and Analysis

Overall, experiments show that HyperOXN can achieve good performance for multicast flows over Fat-Tree and Random. For mixed flows combining multicast and unicast, as the network size increases, HyperOXN can match Random's performance. In unicast flows only scenarios, regarding the HyperOXN's 15% lower cost advantage, HyperOXN with the same cost as Random (denoted as HOXN-a in the following figures) can match the performance of Random.

4.3.3.1 Multicast-Only TM

Figure 4.3 shows that HyperOXN's advantage for a multicast-only TM is consistent for large-scale networks. As the network size increases, HyperOXN achieves up to 50% higher relative throughput than Fat-Tree. Random topologies can achieve near-optimal performance compared with all others for unicast flows [66], but it shows disadvantages in multicast scenarios.

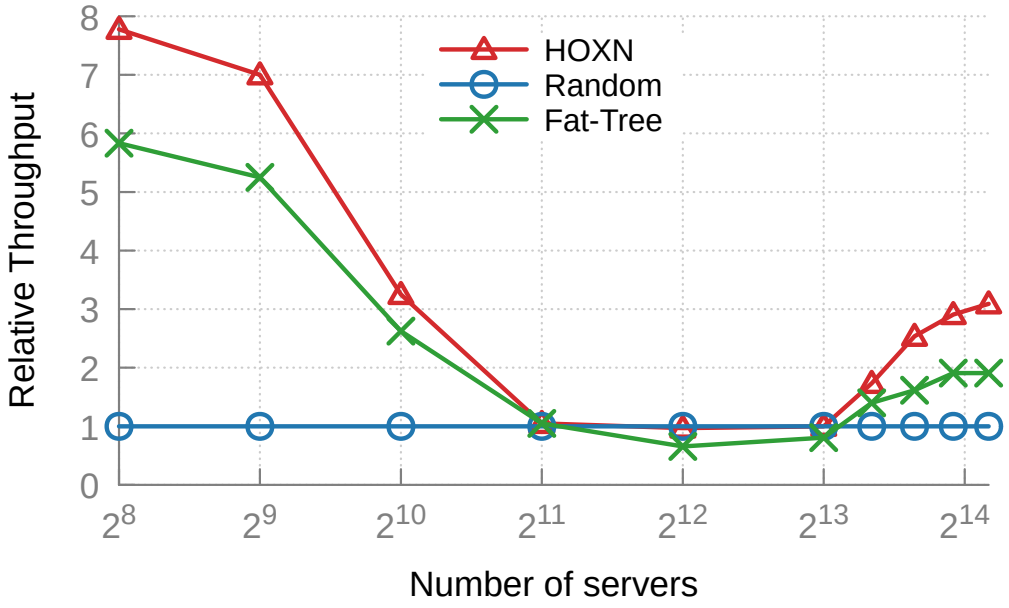


FIGURE 4.3: Multicast-only TM

4.3.3.2 RTM

As shown in Figure 4.4 and Figure 4.5, for an RTM with unicast flows, Random is the best, ignoring the cost advantage of HyperOXN. For a large-scale network with mixed traffic, specifically for multicast flows, HyperOXN demonstrates an inherent advantage over the other topologies.

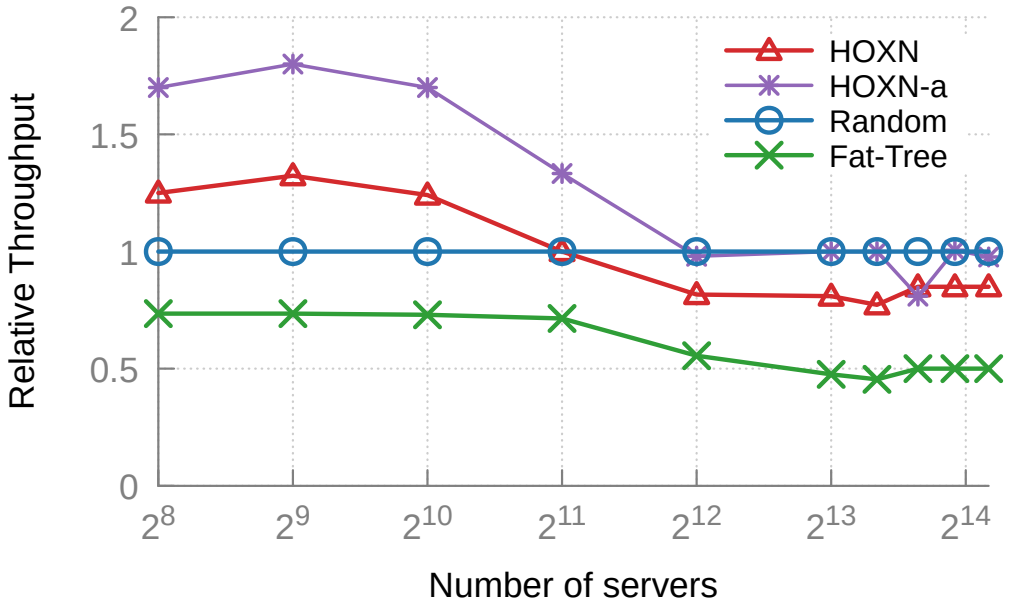


FIGURE 4.4: Random Matching TM for unicast flows

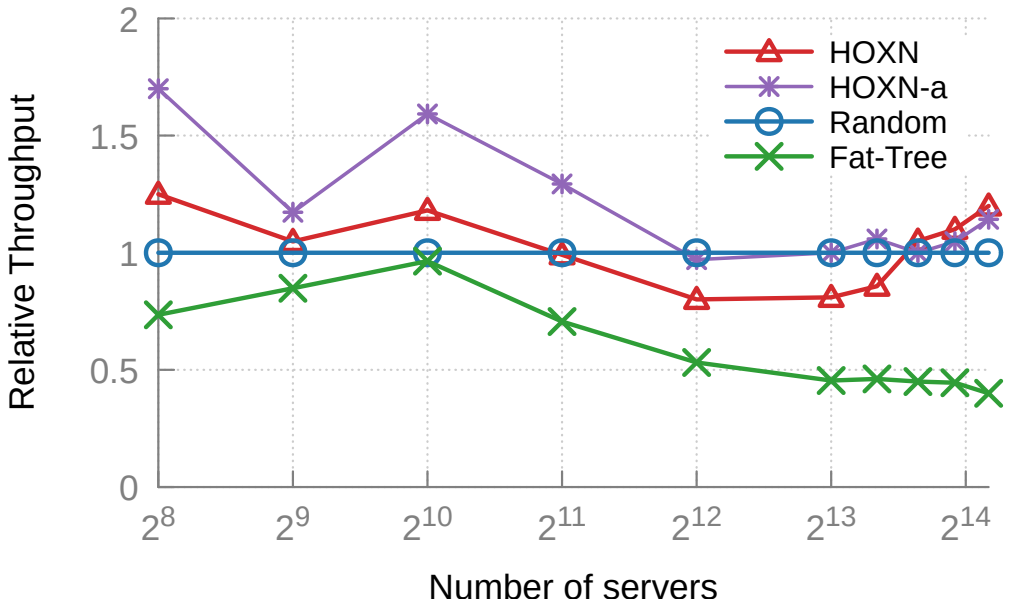


FIGURE 4.5: Random Matching TM for mixed flows

4.3.3.3 A2A

In a full-bisection bandwidth Fat-Tree, throughput under A2A is the lowest in unicast-only traffic and mixed traffic as shown in Figure 4.6 and 4.7. Although

Random achieves the best performance for unicast flows, HOXN can match Random's performance under mixed traffic.

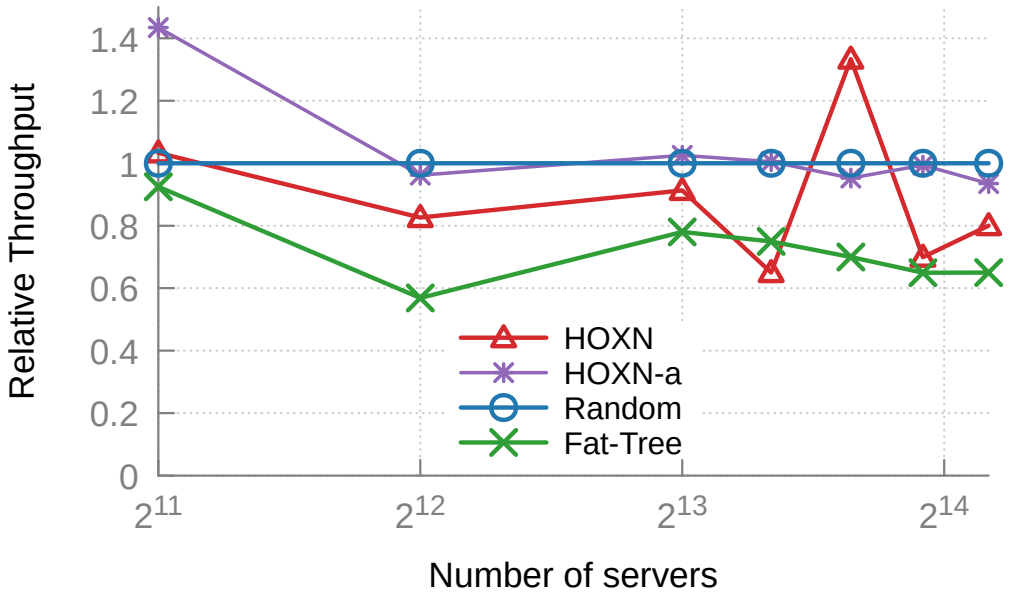


FIGURE 4.6: All to All TM for unicast flows

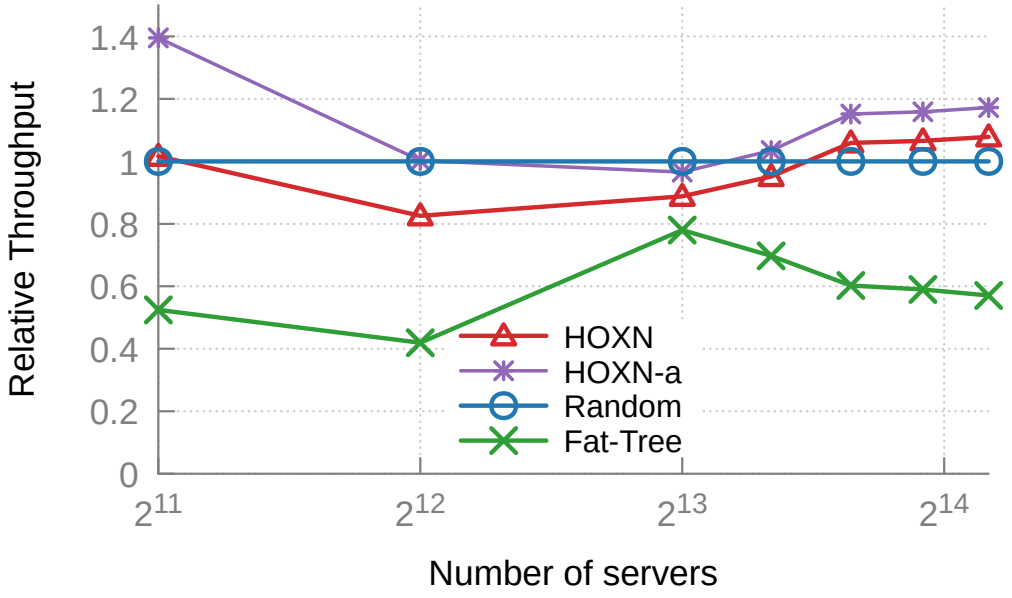


FIGURE 4.7: All to All TM for mixed flows

4.3.3.4 LTM

The analogous set of results for LTM is similar to the ones for A2A. Figure 4.8 shows HOXN can match Random's performance for unicast traffic if regarding the 15% lower cost advantage. Under mixed flows, HOXN exhibits better performance than Random and Fat-Tree shown in Figure 4.9, as the network size grows.

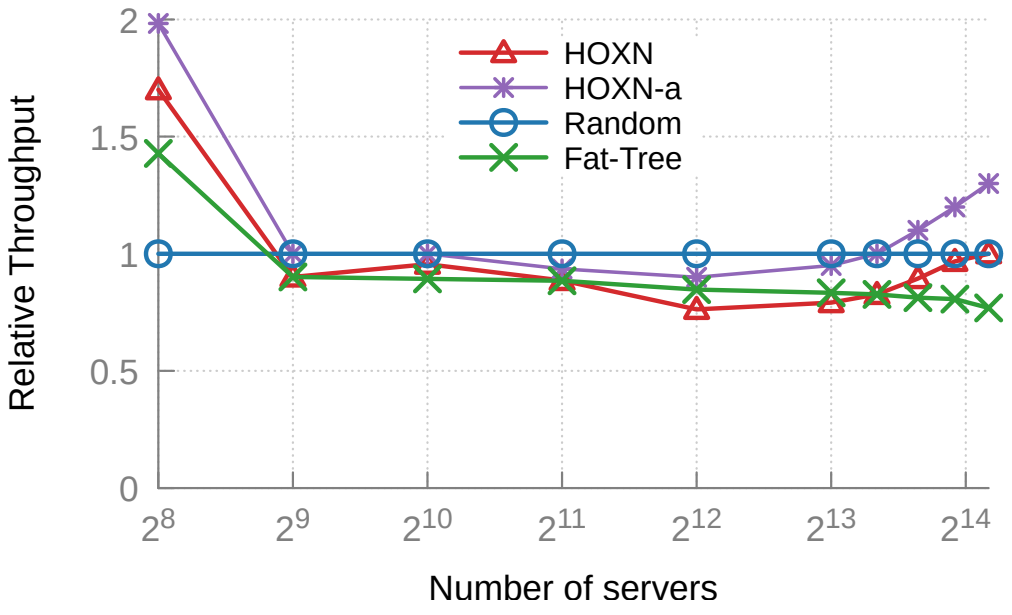


FIGURE 4.8: Longest Matching for unicast flows

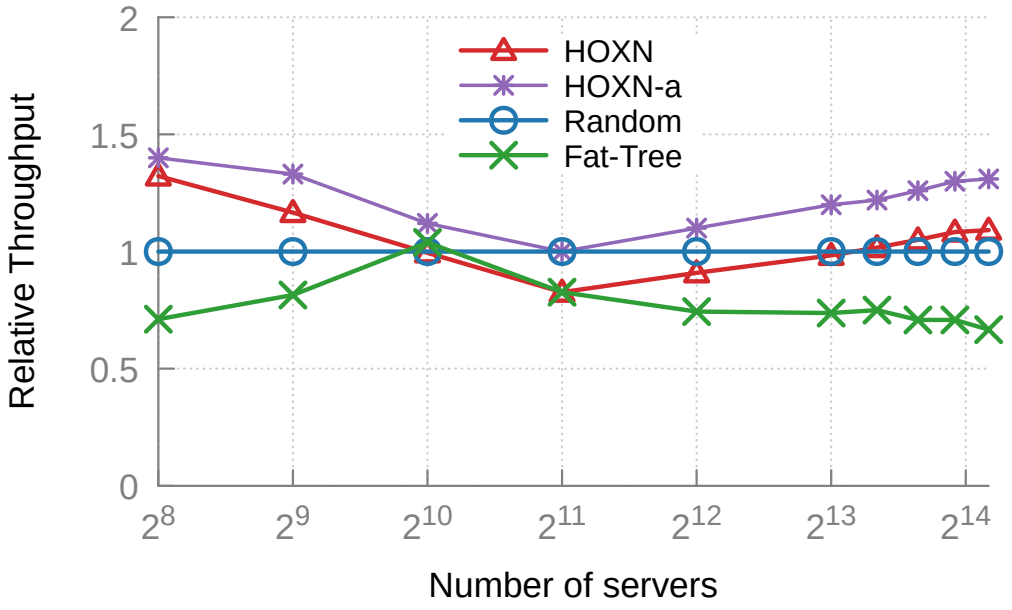


FIGURE 4.9: Longest Matching TM for mixed flows

4.4 Summary

HyperOXN takes advantage of high-radix switch components leveraging state-of-the-art colourless wavelength division multiplexing technologies, effectively supports multicast traffic, and at the same time meets the demands for high throughput. Flow-level simulations demonstrate that HyperOXN can achieve good performance for multicast flows over Fat-Tree and Random. As the network size increases, HyperOXN can match Random's performance for mixed flows.

Chapter 5

HOLMES$^+$: Holistic Mice-Elephants Stochastic Scheduling Architecture in DCNs

Scheduling between latency-sensitive small data flows (*aka* mice) and throughput-oriented large ones (*aka* elephants) has become more challenging with the proliferation of cloud based applications. In light of this mounting problem, this work proposes a novel flow control scheme, HOLMES (HOListic Mice-Elephants Stochastic) [41] [42], which offers a holistic view of global congestion awareness as well as a scheduler of mixed mice-elephants data flows based on the stochastic DCN model.

HOLMES$^+$ is based on the HOLMES[41] [42] architecture, but it enhances load balance and end-to-end path congestion detection. HOLMES$^+$ can leverage local network information when global network status is lacking. As such, most of this chapter borrows heavily from HOLMES [42].

In this chapter, firstly, the necessity for partitioning DCN paths into sub-networks using a stochastic queue model is theoretically proven. Secondly the HOLMES$^+$ architecture is proposed, which adaptively partitions the available DCN paths into low-latency and high-throughput sub-networks via a global congestion-aware

scheduling mechanism. Based on the stochastic power-of-two-choices policy (P2C) [125], the HOLMES$^+$ scheduling mechanism acquires only a subset of the global congestion information, while it achieves close to optimal load balancing on each of the end-to-end DCN paths. The stability of HOLMES$^+$ flow scheduling algorithm is also proven. Finally, extensive simulations in Setction 5.4 validate the effectiveness and stability of HOLMES$^+$ with select DCN topologies for **Fat-Trees**. The performance of HOLMES$^+$ used in HyperOXN is evaluated in Chapter 7.

This chapter is organized as follows. Section 5.1 presents an overview of HOLMES$^+$, and Section 5.2 and 5.3 discuss in detail the HOLMES$^+$ architecture and its load balancing mechanisms, respectively. Lastly, Section 5.4 evaluates HOLMES$^+$ with extensive simulation experiments.

5.1 HOLMES$^+$ Overview

HOLMES$^+$ aims to satisfy the competing performance requirements of elephants and mice by partitioning a DCN into high-throughput and low-latency sub-networks. HOLMES$^+$ proposes a stochastic and global congestion-aware mechanism to schedule the elephant and mouse flows in these separate sub-networks. Also, the necessity of network partition and the stability of the HOLMES$^+$ scheduling algorithm are theoretically proven.

5.1.1 Necessity of Network Partition

In order to lay out a solid foundation for HOLMES$^+$, there is a need to theoretically prove the limitations of the unified scheduling schemes when handling hybrid DC traffic. The proposed analytic model promotes the need for deploying network partition solutions in the architecture of flow scheduling and congestion control schemes.

The main challenge that any unified load-balancing scheme needs to address is the interference between elephant and mouse flows. A typical scenario in which mouse

W_i	TCP windows size for round i; $1 \leq W_i \geq W_{max}$
$E[W]$	Expected average TCP window size
$W(t)$	Window size of a flow at time t
b	Number of packets that are acknowledged by a received ACK
L_i	Number of packets lost in the $(i-1)^{st}$ round, $1 \leq L_i < W_{i-1}$
p	Probability that a packet in a flow is lost
$Q(t)$	Queue length of an end-to-end path at time t
P_{SS}	Steady-state probability of the slow start period
P_{CA}	Steady-state probability of the congestion avoidance period
$B(p)$	Steady-state send rate of a TCP connection with probability of packet loss p
C	Bottleneck end-to-end link capacity of an end-to-end path
$Tred$	Marking threshold
T_0	Acknowledgment (ACK) timeout interval (ATO)
T_C	Duration of one TCP congestion avoidance period
T_S	Duration of one TCP slow start period
Q_{max}	Maximum queue size of an end-to-end path
N	Total amount of DC flows
A_w	Amplitude of oscillation in window size of a single flow
A_q	Amplitude of queue oscillations of an end-to-end path
P_{M-E}	Probability that mice flows affect elephants flows
P_{E-M}	Probability that elephants flows affect mice flows

TABLE 5.1: Symbols used in HOLMES$^+$

flows affect elephant flows is when a burst of mouse flows triggers a congestion signal, e.g., Quantized Congestion Notification (QCN) [102], which in turn reduces the transmission rates of elephant flows over a subsequent long period of time. The measurement results in [35] [158] [159] show that the short-lived flows in large-scale clusters exhibit significant traffic bursts. Benson [35] [37] further explained that the traffic bursts in DCNs are caused by the huge amount of aggregated edge links as well as the concurrency of parallelized computing schemes, e.g., the shuffle period in a MapReduce process. Experimental results also show that a momentary traffic burst can lead to short-lived congestion in a DCN [37]. On the other hand, the effect of elephant flows on mouse flows is that elephant flows occupy the vast majority of the network buffers, and hence the mouse flows will result in long queuing delays and flow completion times (FCTs), a.k.a. the bufferbloat problem [160], where the existence of excessively large and frequently full buffers inside the network, causes unnecessary latency and poor system performance.

The probability that the above two scenarios would occur is calculated by the

unified flow scheduling solutions and also how the two flows would affect network performance. Some key notations and definitions are given in Tables 5.1. Then it is proven that fundamental limitations exist in the unified scheduling schemes when handling hybrid DC traffic. Therefore, network partitioning is a better solution that eliminates the interference by separating the mouse and elephant flows.

The proof of the necessity for DCN traffic partitioning is based on the performance modeling results of TCP Reno [161]. By analyzing the interactions between the mouse and elephant TCP flows and studying how the interactions of the elephant and mouse flows affect the network performance, it is proven that network partitioning is a better solution that eliminates the interference by separating the mouse and elephant flows. Please see details of the proof in Section 5.1.2.

5.1.2 Proof of The Necessity for DCN Partitioning

This section analyzes the interactions between the mouse and elephant TCP flows. Based on the performance modeling results of TCP Reno proposed by Padhye [161], a stochastic TCP model is deployed to analyze the steady-state probability of two TCP periods: the slow start period and the congestion avoidance period. A two-state Markov Chain can be used to model the two TCP states: the slow start state and the congestion avoidance state. Hence, study the probability that interactions of elephant and mouse flows occur, and how the interactions affect the network performance.

5.1.2.1 Model for Two TCP States

When a packet loss occurs, the TCP transmission will enter a slow start period or a congestion avoidance period. Define P_{CA-SS} as the probability that TCP enters the slow start period from the congestion avoidance period, and define P_{SS-CA} as the probability of entering the congestion avoidance period from the slow start period. According to [161], $P_{ack}(w,k)$ is defined as the probability that the first

k packets are ACKed in a round of w packets, given there is a sequence of one or more losses in the round. Then

$$P_{ack}(w,k) = \frac{(1-p)^k p}{1-(1-p)^w}$$

and let $P_c(n,m)$ be denoted as the probability that m packets are ACKed in sequence in the last round and the rest of the packets in the round are lost. Then

$$P_c(n,m) = \begin{cases} (1-p)^m p, & m \leq n-1 \\ (1-p)^n, & m = n \end{cases}$$

Then, the probability for a TO loss in a window of size w of the congestion avoidance phase, denoted as $Lto(w)$, is given by

$$Lto(w) = \begin{cases} 1, & w \leq 3 \\ \sum_{k=0}^{2} P_{ack}(w,k) + \sum_{k=3}^{w} P_{ack}(w,k) \sum_{m=0}^{2} P_c(k,m), & otherwise. \end{cases}$$

After algebraic manipulations, $Lto(w)$ can be gotten as the following:

$$\begin{aligned} Lto(w) &= min\left(1, \frac{(1-(1-p)^3)(1+(1-p)^3(1-(1-p)^{w-3}))}{1-(1-p)^w}\right) \\ &\approx min\left(1, \frac{3}{w}\right) \end{aligned} \tag{5.1}$$

Thus,

$$\begin{aligned} P_{CA-SS} &= \sum_{w=1}^{\infty} Lto(w) P\{W=w\} \\ &= \sum_{m=1}^{\infty} min(1, \frac{(1-(1-p)^3)(1+(1-p)^3(1-(1-p)^{w-3}))}{1-(1-p)^w}) P\{W=w\} \\ &= min(1, \frac{(1-(1-p)^3)(1+(1-p)^3(1-(1-p)^{E[W]-3}))}{1-(1-p)^{E[W]}}) \\ &\approx min(1, \frac{3}{E[W]}) \end{aligned} \tag{5.2}$$

According to the derivations in [161], the expected average window size of a single TCP flow is:

$$E[W] \approx \frac{2+b}{3b} + \sqrt{\frac{8(1-p)}{3bp} + \left(\frac{2+b}{3b}\right)^2} \approx \sqrt{\frac{8}{3bp}} \tag{5.3}$$

and the relationship between the packet sending rate $B(p)$ and the packet loss probability p is derived as:

$$B(p) \approx \frac{1}{RTT\sqrt{\frac{2bp}{3}} + T_0 \cdot min\left(1, 3\sqrt{\frac{3bp}{8}}\right) p(1+32p^2)} \tag{5.4}$$

Based on the above intermediate modeling results, derive the probability P_{CA-SS} as:

$$P_{CA-SS} \approx min\{1, 3\sqrt{\frac{3bp}{8}}\} \tag{5.5}$$

During a slow start period, the window size of a TCP flow increases exponentially, until it reaches a specific threshold. Then, the TCP transmission will enter the congestion avoidance period, which means there are no packet losses in the slow start phase. As such, the probability that the transmission enters a congestion avoidance period from a slow start period can be formulated according to [161]:

$$\begin{aligned} P_{SS-CA} &= \sum_{w=1}^{\infty} \left(\prod_{i=1}^{\sqrt{w/2}} (1-p)^{i^2 b} \right) P\{W=w\} \approx \sum_{i=1}^{\sqrt{E[W]/2}} (1-p)^{i^2 b} \\ &\approx (1-p)^{\frac{b}{6\sqrt[4]{bp}}} \end{aligned} \tag{5.6}$$

The two states (slow start state and congestion avoidance state) then form a two-state Markov Chain. Based on the intermediate results of the regenerative process [162], the steady-state probabilities of the slow start and congestion avoidance

periods can be expressed by:

$$P_{CA} = \frac{P_{SS-CA}}{P_{SS-CA} + P_{CA-SS}} = \frac{(1-p)^{\frac{b}{6\sqrt[4]{bp}}}}{min\{1, 3\sqrt{\frac{3bp}{8}}\} + (1-p)^{\frac{b}{6\sqrt[4]{bp}}}} \tag{5.7}$$

$$P_{SS} = \frac{P_{CA-SS}}{P_{SS-CA} + P_{CA-SS}} = \frac{min\{1, 3\sqrt{\frac{3bp}{8}}\}}{min\{1, 3\sqrt{\frac{3bp}{8}}\} + (1-p)^{\frac{b}{6\sqrt[4]{bp}}}} \tag{5.8}$$

Thus, the probability that interactions between the elephant and mice flows occur needs to be modeled. During the congestion avoidance period, the window size of a flow increases linearly. $Pt(W_1, W_2)$ is defined as the overall amount of packets transmitted during the period that the window size grows from $W1$ to $W2$. it follows that:

$$Pt(W_1, W_2) = (W_2 - W_1) \cdot (W_2 + W_1)/2 \tag{5.9}$$

Observe that:

$$Q(t) = NW(t) - C \cdot RTT \tag{5.10}$$

The relationship between the maximum window size of a TCP flow and the threshold for triggering a congestion signal can be derived as:

$$W_{max} = (C \cdot RTT + T_{red})/N \tag{5.11}$$

The amplitude of the oscillation in the window size of a single flow can be formulated as:

$$\begin{aligned} A_w &= (W_{max} + 1) - (W_{max} + 1)/2 \\ &= (W_{max} + 1)/2 \\ &= \frac{1}{2N}(C \cdot RTT + T_{red}) + \frac{1}{2} \end{aligned} \tag{5.12}$$

The amplitude of the queue oscillations of an end-to-end path can be further calculated as:

$$A_q = N \cdot A_w = \frac{C \cdot RTT + T_{red} + N}{2} \tag{5.13}$$

During the congestion avoidance period, the window size increases one unit at one time period. Thus, the duration of this period is:

$$T_c = A_w = \frac{1}{2N}(C \cdot RTT + T_{red}) + \frac{1}{2} \tag{5.14}$$

The maximum queue length of an end-to-end DC path is calculated as:

$$Q_{max} = N(W_{max} + 1) - C \cdot RTT = N + T_{red} \tag{5.15}$$

By substituting (5.11), then:

$$Q_{max} = N + T_{red} \tag{5.16}$$

5.1.2.2 How Interactions Affect Network Performance

Based on the above results, two steps are given as follows to analyze the probability that interactions of elephant and mouse flows occur and study how the interactions affect the network performance. The two steps are as follows.

(I) Probabilities that Hybrid Flow Interactions Occur

In large-scale data centers, there exists huge amount of traffic, and thus N is large enough. Assume that the window sizes of the N flows are independently and identically distributed. During the convergence avoidance period, the window size of a flow increases linearly. As such, the queue length distribution of an end-to-end path shared by the N flows can be calculated

The expectation μ and variance σ^2 of a single flow's window size are:

$$\begin{aligned}\mu &= \frac{1}{2}\frac{(W_{max}+1)}{2} + \frac{1}{2}(W_{max}+1) \\ &= \frac{3W_{max}+3}{4}\end{aligned}$$

and

$$\sigma^2 \approx \frac{(W_{max}+1)^2}{48}$$

According to the Central Limit Theorem [163], the overall window size of the aggregated flow in an end-to-end path satisfies a Gaussian distribution. Based on (5.10), derive the distribution of the end-to-end queue length $Q(t)$ as:

$$\frac{Q(t) + C \cdot RTT - \frac{3NW_{max}+3N}{4}}{4\sqrt{3N}} \sim N(0,1) \tag{5.17}$$

where the cumulative probability function (CDF) of the standard normal distribution $N(0,1)$ [164] is:

$$\Phi(x) = \frac{1}{\sqrt{2\pi}}\int_{-\infty}^{x} e^{-t^2/2}dt = p\{X \leq x\} \tag{5.18}$$

Consider the scenario that a burst of mouse flows triggers the congestion signals and affects the transmission of elephant flows. Obliviously, the probability that this scenario occurs is proportional to the probability P_{tcs} of triggering a congestion signal.

In an end-to-end DC path, the congestion signal is triggered when the queue length exceeds the threshold T_{red}. According to (5.17) and (5.18), the probability of triggering a congestion signal in the congestion avoidance period can be calculated

as:

$$\begin{aligned} P\{Q(t) > T_{red}\} &= 1 - P\{Q(t) \leq T_{red}\} \\ &= 1 - p\{X \leq \frac{4T_{red} + 4C \cdot RTT - 3NW_{max} - 3N}{16\sqrt{3N}}\} \\ &= 1 - \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\frac{4T_{red}+4C\cdot RTT-3NW_{max}-3N}{16\sqrt{3N}}} e^{-t^2/2} dt \end{aligned} \tag{5.19}$$

Combine the above expression (5.19) and (5.7), can derive the expression of P_{tcs}:

$$\begin{aligned} P_{tcs} &= P_{CA} \cdot P\{Q(t) > T_{red}\} \\ &= P_{CA} \cdot \left(1 - \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\frac{4T_{red}+4C\cdot RTT-3NW_m-3N}{16\sqrt{3N}}} e^{-t^2/2} dt\right) \\ &= \frac{(1-p)^{\frac{b}{6\sqrt[4]{bp}}}}{min\{1, 3\sqrt{\frac{3bp}{8}}\} + (1-p)^{\frac{b}{6\sqrt[4]{bp}}}} \cdot (1 - \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\frac{T_{red}+C\cdot RTT-3N}{16\sqrt{3N}}} e^{-t^2/2} dt) \end{aligned} \tag{5.20}$$

It is obtained from the expression P_{tcs} that the probability of triggering a congestion signal is proportional to $-T_{red}$. Suppose the distribution of mouse and elephant flows is fixed. Then, the probability that the mouse flows affect the elephant flows is proportional to P_{tcs}, thus:

$$P_{M-E} \propto P_{tcs} \propto -T_{red} \tag{5.21}$$

The other scenario for interactions between hybrid flows is that the long-lived large flows occupy the vast majority of buffers, while the short-lived small flows will suffer from long queuing delays. Obviously, the probability that this scenario occurs is *proportional* ($\propto$) to the threshold of the queue length, hence:

$$P_{E-M} \propto T_{red} \tag{5.22}$$

Comparing (5.21) and (5.22), it is found that conflicts exist for avoiding the two types of mixed flow interactions. To avoid the interference of mouse flows on

elephant flows, the optimal solution should maximize the queue size threshold to hold more burst traffic; however, when avoiding the effects of elephant flows on mouse flows, the optimal solution should minimize the threshold in order to reduce the queuing delay of the mouse flows. The optimal unified scheduling scheme may find the best trade-off between the two types of interactions but cannot avoid either of the interactions. Therefore, network partitioning is a better solution that eliminates the interactions by separating the mouse and elephant flows.

II. Effect of Mixed Flow Interactions on DCN Performance

The effect of mixed flow interactions on network performance needs to be analyzed. Also, the performance of the same end-to-end path under two different types of interactions, *i.e.*, interference between a mouse flow and an elephant flow, and interaction of two elephant flows, has to be compared.

A time point is considered at which a congestion signal has just been triggered and the window size of each flow has just been reduced. The link utilization condition of the end-to-end path during the period from this time point to the next congestion point ($W(t) = W_{max}$), under the two types of interactions, needs to be evaluated.

1) One mouse flow and one elephant flow

Assume that the overall transmission latency of a mouse flow is less than the duration of a slow start period, that is, the a mouse flow only enters a slow start period. Then, the window size of the mouse flow will not exceed $(W_{max} + 1)/2$, and the maximum transmission time of the mouse flow is:

$$T_s = \log_2 (W_{max} + 1)/2 \tag{5.23}$$

According to (5.14), the duration of the elephant flow's congestion avoidance period (from a congestion point to the next) is T_C, which is equal to $(W_m + 1)/2$.

The overall amount of packets transmitted by the elephant flow is:

$$S_{large} = [(W_{max} + 1) + \frac{1}{2} \cdot (W_{max} + 1)] \cdot \frac{T_c}{2} \tag{5.24}$$

and the overall amount of packets transmitted by the mouse flow is:

$$S_{small} = 1 + 2 + 4 + \cdots + \frac{W_{max} + 1}{2} \approx W_{max} \tag{5.25}$$

The average link utilization during this time period is defined as β_1, and then,

$$\begin{aligned} \beta_1 &= \frac{(S_{small} + S_{large} + 0 \cdot (T_c - T_s)) \cdot (1 - P_{tcs})^{\frac{W_{max}+1}{2}}}{C \cdot T_c} \\ &= \frac{1}{C \cdot T_c}(\frac{3W_{max}^2 + 6W_{max} + 3}{8} + W_{max}) \cdot (1 - P_{tcs})^{\frac{W_{max}+1}{2}} \end{aligned} \tag{5.26}$$

2) Two elephant flows

When the two flows are both elephant flows, each flow's window size will increase from $(W_{max} + 1)/2$ to $W_{max} + 1$. β_2 is denoted as the average link capacity in this scenario, and then,

$$\begin{aligned} \beta_2 &= \frac{2S_{large} \cdot (1 - P_{tcs})^{\frac{W_{max}+1}{2}}}{C \cdot T_s} \\ &= \frac{1}{C \cdot T_s} \cdot \frac{3W_{max}^2 + 6W_{max} + 3}{4} \cdot (1 - P_{tcs})^{\frac{W_{max}+1}{2}} \end{aligned} \tag{5.27}$$

Comparing the link utilization of the end-to-end path under the two types of interactions in (5.26) and (5.27):

$$\beta_2 - \beta_1 > \frac{1}{C \cdot T_s} \cdot (S_{large} - S_{small}) \cdot (1 - P_{tcs})^{\frac{W_{max}+1}{2}}$$

Thus,

$$\beta_2 - \beta_1 > \frac{1}{C \cdot T_s} \cdot \frac{3W_{max}^2 - 2W_{max} + 3}{8} \cdot (1 - P_{tcs})^{\frac{W_{max}+1}{2}} > 0 \tag{5.28}$$

Therefore, compared with the two elephant flows scenario, the interactions of mixed flows further degrade the average link utilization after congestion occurs. In other words, network partitioning is a better solution that eliminates the interference by separating the mouse and elephant flows.

5.1.3 HOLMES$^+$ Architecture with Network Partition

Wang [98] and W. Cheng [99] applied the network partition solutions to separate the large and small flows in DCN paths, and to avoid the interference between the two.

Similarly, HOLMES$^+$ is based on the network partition solution, which partitions and separates all the end-to-end DCN paths into two logical sub-networks in order to isolate the transmission of elephant and mouse flows. A centralized controller is deployed to map each link to either the high throughput or the low latency sub-network. The traffic transmitted between each pair of edge switches may contain both elephant and mouse flows. Therefore, the HOLMES$^+$ architecture needs to ensure that at least one of the multiple paths between each two edge switches belongs to the high throughput sub-network, and at least another one belongs to the low latency sub-network. The scales of the two sub-networks can be determined either statically or dynamically according to the architecture of the DC as well as the traffic patterns. For example, Wang [98] proposed a dynamic path partitioning algorithm to dynamically adjust the number of low latency and high throughput paths according to the traffic load.

In addition to network partitioning, network statistics and analysis functions are provided by the HOLMES$^+$ AI module. The detailed architecture will be discussed in Section 5.2.

5.1.4 Global Congestion-aware Load Balancing Algorithm based on P2C Model

Another important component in HOLMES$^+$ is its load balancing algorithm. Since local congestion-aware load balancing algorithms have been proven to react slowly to link failures [43] and are prone to forming congestion trees [95], the global congestion-aware load balancing is deployed in HOLMES$^+$. Inspired by the P2C model [125], a stochastic load-sensitive flow scheduling algorithm is further designed which reduces the overhead for congestion information maintenance and improves the scheduling efficiency.

M. Mitzenmacher and S. Ghorbani showed that tagging a single unit of memory with random sampling and identifing the shortest output queue from the previous time slots guarantees stability [94] [125]. A distributed implementation of this approach is extended in Section 5.3: when a packet arrives at an input port of an edge ToR switch, that ToR switch resolves the source and destination addresses of the packet. It then assigns the packet to the least-loaded of d randomly chosen end-to-end paths from all N ($d << N$) paths by the packet forwarding module. The choice of the previous time slot is saved for the scheduling of the next time slot.

Section 5.3 will discuss the detailed process and analyze the stability of this load balancing algorithm.

5.2 HOLMES$^+$ Architecture

Figure 5.1 shows the HOLMES$^+$ architecture in three layers: the AI layer, control layer, and infrastructure layer. Based on the real-time monitoring information, HOLMES$^+$ AI module first analyzes the status or tendency of the DCN using Machine Learning models, and generates the learning results such as networking tendency predictions. Next, according to the analysis results, the HOLMES$^+$ control layer designs a network partition policy and a corresponding random flow

scheduling policy, and the policies are generated in the SDN controllers. Finally, the network partition as well as the flow scheduling operations will be executed on the DCN switches or hosts under the controllers.

The HOLMES$^+$ AI layer contains a computing cluster with Machine Learning to implement the software AI module. The AI module collects the detailed DCN information from the DCN monitoring modules, and applies the Machine Learning like prediction algorithms to generate some analysis results, e.g. networking tendency predictions and network outlier locations. These learning results generated by the AI module provide a more comprehensive view of DCN behaviors. They will then be used for network partitioning and flow scheduling operations.

The HOLMES$^+$ control layer for policy management is responsible for generating the network partition, congestion control, and local balancing policies, based on the monitoring information as well as the learning results generated by the AI module. The policies generated in the controllers are decomposed into a series of fine-grained partition and scheduling orders, which are transmitted to the DCN switches and end hosts for execution. Without deploying the HOLMES$^+$ AI module, the functions provided by the control layer are the same as the SDN controllers.

The HOLMES$^+$ infrastructure layer executes specific network partitions as well as the flow scheduling operations. It is responsible for storing, forwarding and processing data packets. The detailed operation orders are transmitted and configured on the DCN switches. The DCN switches first map each link to the high throughput network or the low latency sub-network, according to the network partition policies. When the elephant and mouse flows arrive at a DCN switch, their packets are separately scheduled to the pre-partitioned paths, which are managed by the HOLMES$^+$ control layer.

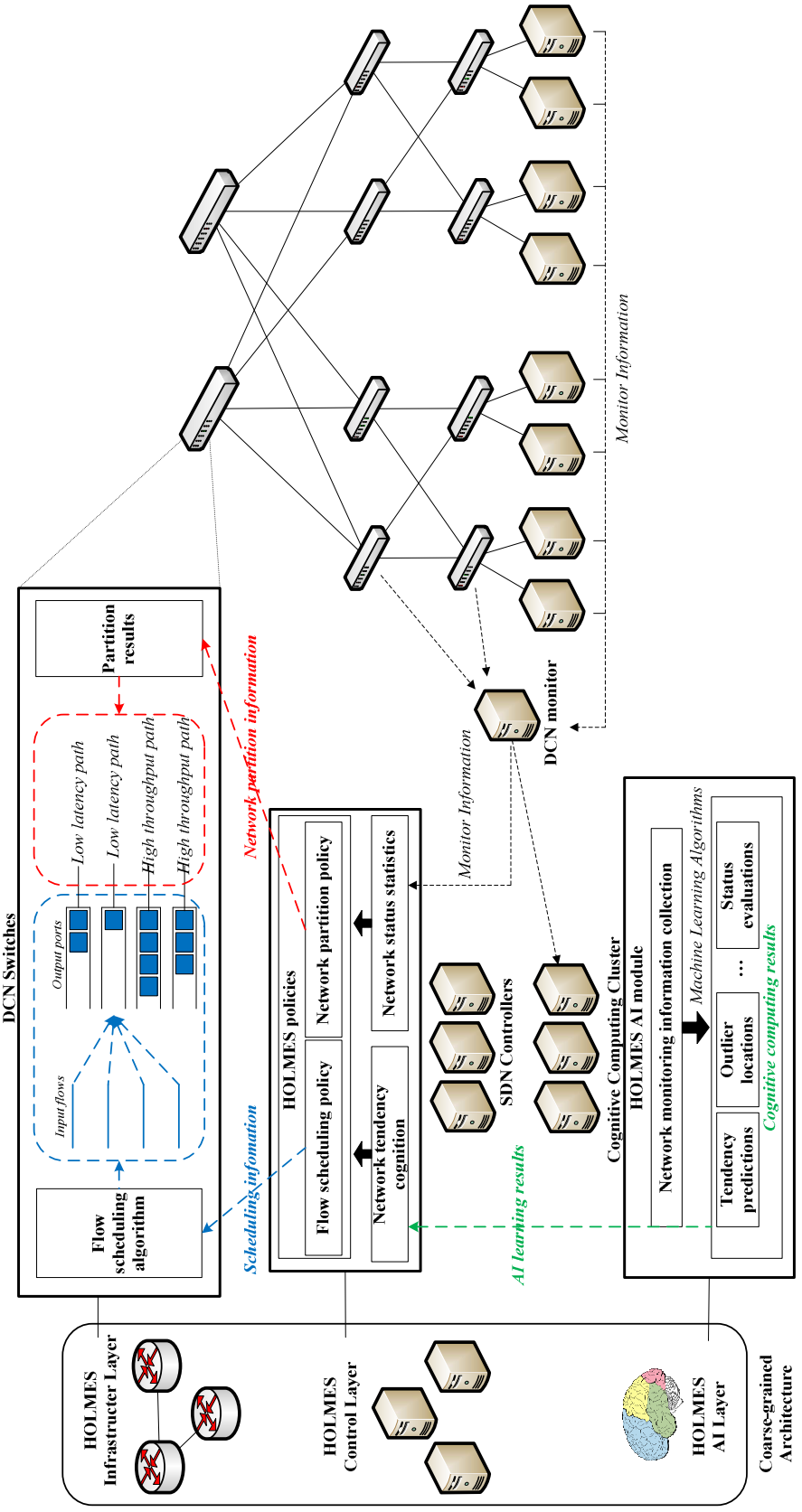


FIGURE 5.1: HOLMES$^+$ architecture [12]

5.2.1 HOLMES$^+$ Network Partition

Both static and dynamic network partition mechanisms are enabled by HOLMES$^+$. When the scale of the DCN is small, HOLMES$^+$ provides static network partition policies to DCN switches to avoid the frequent scale changes of the two subnetworks. On the other hand, when the scale of the DCN is large, HOLMES$^+$ will deploy a dynamic network partition solution that dynamically adapts the scales of the two types of sub-networks to the traffic load variants. Moreover, the dynamic network partition solution utilizes the network resources more adaptively, especially when the arrival rate of DC traffic is slow.

Both static and dynamic network partition mechanisms have been studied in-depth [98] [99]. One can either integrate these mechanisms in HOLMES$^+$ or implement the appropriate network partition mechanism according to the scale of the DCN as well as the traffic patterns. HOLMES$^+$ focuses on the flow scheduling algorithm.

5.2.2 Application Scenarios for HOLMES$^+$ Architecture

Compared with the commonly used SDN architectures, a prominent feature of HOLMES$^+$ is the implementation of an AI module and its Machine Learning algorithms. Machine Learning methods have been widely used in network management [165] and DC scheduling policies for the generation of operations [166]. Those continuing learning and analyzing results provide a comprehensive understanding of network features and behaviors, which benefits the designing of corresponding network partition and flow scheduling policies. Therefore, the deep analysis and accurate AI prediction provided by the AI module enable the HOLMES$^+$ architecture to perform more complex and intelligent operations.

One typical application scenario for the HOLMES$^+$ architecture is the deployment of application driven networks (ADN) [130], where a physical network is sliced into logically isolated sub-networks to serve different types of applications. Each network slice in ADN can deploy its own architecture and corresponding protocols,

to satisfy the requirements of the applications it serves. The key operations when implementing ADN are:

(1) Constructing an application abstraction model to formulate the resource requirements of the applications; and

(2) Mapping the distinct properties of applications to particular network resources.

It has been shown that the complexity and performance of these operations can be improved when some application features are pre-known [90]. Hence, the HOLMES^{+} AI module shown in Figure 5.1 benefits the analysis of application features as well as the prediction of resource requirements, which further alleviates the complexity of application abstractions and mapping operations. Moreover, the design and implementation of network slicing mechanisms can also be realized by cooperation of the control layer and the infrastructure layer.

Similarly, the HOLMES^{+} architecture is also applicable for some other intelligent or complex application scenarios that demand a deep understanding of network or application features, such as co-flow scheduling [167] and some other network architectures based on network-slicing or network partitions. With the proliferation of complex and diverse cloud-based applications, such architectures are expected to be the development trend in the future.

5.2.3 Out-of-order vs. Efficiency

While elephant-flows contribute to the majority volume of DCN traffic, mouse traffic accounts for 80% of flows [35] [38]. Out-of-order scheduling done by an NIC driver on the host side may sacrifice some CPU efficiency. In addition, compatibility with legacy hardware has to be ensured. Therefore, one may still consider deploying conventional, sometimes oblivious, ECMP scheduling for the in-order scheduling of mice.

5.3 HOLMES$^+$ Scheduling Algorithms

Once the hybrid DCN traffic is separated into different sub-networks, scheduling algorithms affect the performance of each sub-network. In this section, the global congestion-aware scheduling algorithm is discussed and the stability of its stochastic policy is proven.

5.3.1 Necessity of Deploying Stochastic Scheduling Algorithms

Compared with other state-aware flow scheduling algorithms, stochastic flow scheduling algorithms are more applicable for large-scale Data Centers for the following reasons.

5.3.1.1 Simplification of computing complexity

One of the key factors that degrade the performance of the traditional ECMP mechanism is a lack of global congestion information. To overcome this limitation, a group of studies [43] [94] [96] [34] [97] designed new flow scheduling polices based on a global "macroscopic" view of the entire DCN, e.g., CONGA [43]. However, in large-scale and heavily loaded Data Centers, the global macroscopic load balancing algorithms introduce unacceptable computing complexity [43] [96] in dealing with the massive information, and the scheduling control loops in these scenarios are much slower than the duration of the majority of congestion incidents in Data Centers [94]. Therefore, deploying stochastic algorithms like DRILL (Distributed Randomized In-network Localized Load-balancing) in [94] to achieve micro load balancing is a more viable solution. The micro load balancing solutions require only limited congestion information, which simplifies the computing complexity and enables instant reactions to load variations in large-scale Data Centers.

5.3.1.2 Optimization of storage complexity

In Data Centers, 8- and 16-way multipathing are common, and there is growing interest in as many as 32 or even 64 multipaths. Specifically, for example, with 40 servers in a rack, there will be 40 uplinks. Each flow can use a different subset of the 40 links, leading to $\mathbf{2^{40}}$ possible subsets. Maintaining the state of each path in this scenario requires unacceptable amounts of storage resources, which is difficult to implement. As illustrated in Figure 5.2, a leaf switch L_1 has 40 outlinks to 40 spine switches, and spine switch $S1$ has a failed downlink to L_k. Consequently, flows from L_1 to L_k cannot be switched by the output link to $S1$ but the other flows can. As such, the flow from L_1 to L_k has to choose a subset of paths. Similar patterns of failure may happen and make other flows choose different subsets.

In contrast, stochastic scheduling algorithms are effective for coping with storage complexity optimization as well as small numbers of register reads. Edsall [114] deployed the stochastic P2C hashing solution for balancing the loads of DC routers. The storage complexity of such a stochastic solution is logarithmically reduced, which is discussed in Section 5.4.4.2, page 161.

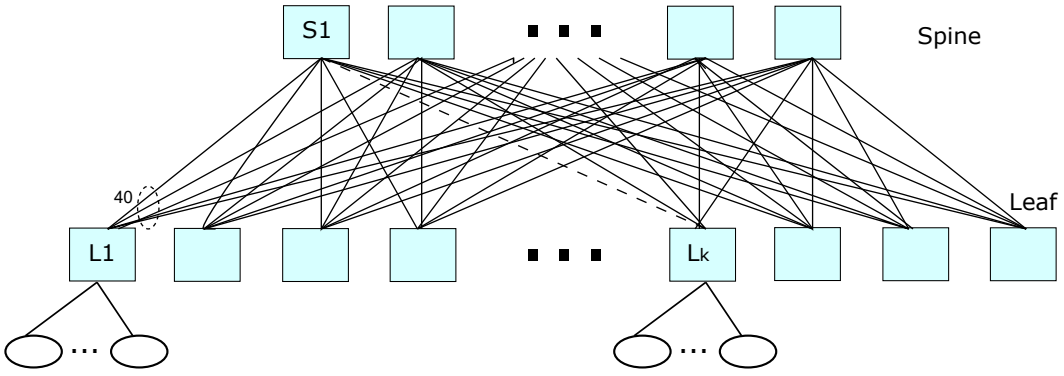


FIGURE 5.2: Exponential numbers of potential ECMP path sets

5.3.1.3 Better adaptability for heterogeneous DCNs

A typical flow scheduling method in multi-rooted DCNs is equal traffic splitting based on hashing as used in the traditional ECMP approach which is viewed as

a *de facto* configuration for switches and routers. However, the uniform hashing approach cannot achieve optimal load balancing without the assumption of symmetric and fault-free topology [88] [43] [35], which is not generally true in heterogeneous Data Centers. To provide better adaptability for heterogeneous DCNs, weighted traffic distribution methods [92] have been widely adopted in global macro load balancing solutions [92]. In order to correct the imbalance caused by the even distribution approach and to enable fair resource allocation, the weighted approaches distribute the traffic among available paths in proportion to the available link capacity of each path. The global weighted traffic distribution solutions have shown good adaptability to dynamically changing network topologies, however, these solutions still need real-time state information for all the paths. This introduces additional computing and storage complexity.

Stochastic flow scheduling algorithms can reduce the computing and storage overhead for weighted traffic distribution mechanisms while maintaining adaptability to heterogeneous DCN topologies. Consider the stochastic P2C: the algorithm only needs to record the states of two randomly chosen paths; therefore, the storage and computing complexity are dramatically reduced. Moreover, the algorithm compares the load conditions of these two chosen paths and selects the better one, hence performing a weighted operation in another form. Stochastic load balancing solutions have also been proven to be applicable for heterogeneous DCNs [94] [114].

Based on these justifications, the flow scheduling algorithms is extended to our HOLMES$^+$ algorithm.

5.3.2 Flow Scheduling Algorithm in HOLMES$^+$

A stochastic scheduling policy, HOLMES$^+(d, m)$, is introduced. The input port of TORs chooses d random end-to-end paths out of all the possible paths. It finds the path with the minimum occupancy among all the d samples and m least loaded

Algorithm 1: Flow scheduling policy in HOLMES^{+}

Input: Load condition of each end-to-end path at time t:
$\{load(1), load(2), load(3), \cdots\}$
Path number of the selected path at time $t-1$: $s^{(t-1)}$
Output ports of the ToR and aggregate switches on each path:
$\{\{P^A_{(1)}, P^T_{(1)}\}, \{P^A_{(2)}, P^T_{(2)}\}, \cdots\}$

Output: Path number of the selected path at time $t-1$: s
Output ports ToR and aggregate ports on the selected path:
$\{P^A_{(s)}, P^T_{(s)}\}$

1 Initialize: $m = 2, d = 1$;
2 Initialize: $load_{OPT} = load(s^{(t-1)})$;
3 Random select m end-to-end paths $\{Path(1), Path(2), ..., Path(m)\}$;
4 Construct candidate set: $P' \leftarrow \{Path(1), Path(2), ..., Path(m)\} \cup \{s^{(t-1)}\}$;
5 **for** each path $i (1 \leq i \leq m+1)$ in the candidate set P' **do**
6 **if** $load(P'(i)) \leq load_{OPT}$ **then**
7 $load_{OPT} = load(P'(i))$;
8 **end for**
9 Assign value: $s =$ Path number of the path with load $load_{OPT}$;
10 **return** $\{P^A_{(S)}, P^T_{(S)}\}$ and s.

samples from the previous scheduling time slot. It then schedules the input packet to the selected end-to-end path.

Increasing the value of d and m to $\gg 2$ and $\gg 1$ respectively will degrade the performance, since a large number of random samples makes it more likely to cause a burst of packet arrivals on the same path [94]. As a result, let $m = 1$ and $d = 2$ in our scheduling model. The detailed flow scheduling procedure is shown in Algorithm 1 and Figure 5.3.

5.3.3 Stability Analysis of HOLMES^{+}'s Scheduling Algorithm

It's proven that the stability of this stochastic global congestion-aware scheduling algorithm in a two-tier Clos DCN topology as follows.

An end-to-end path is abstracted in a Clos network (Figure 5.4A) as a serial queuing system which consists of a series of queues (Figure 5.4B). Thus, the whole

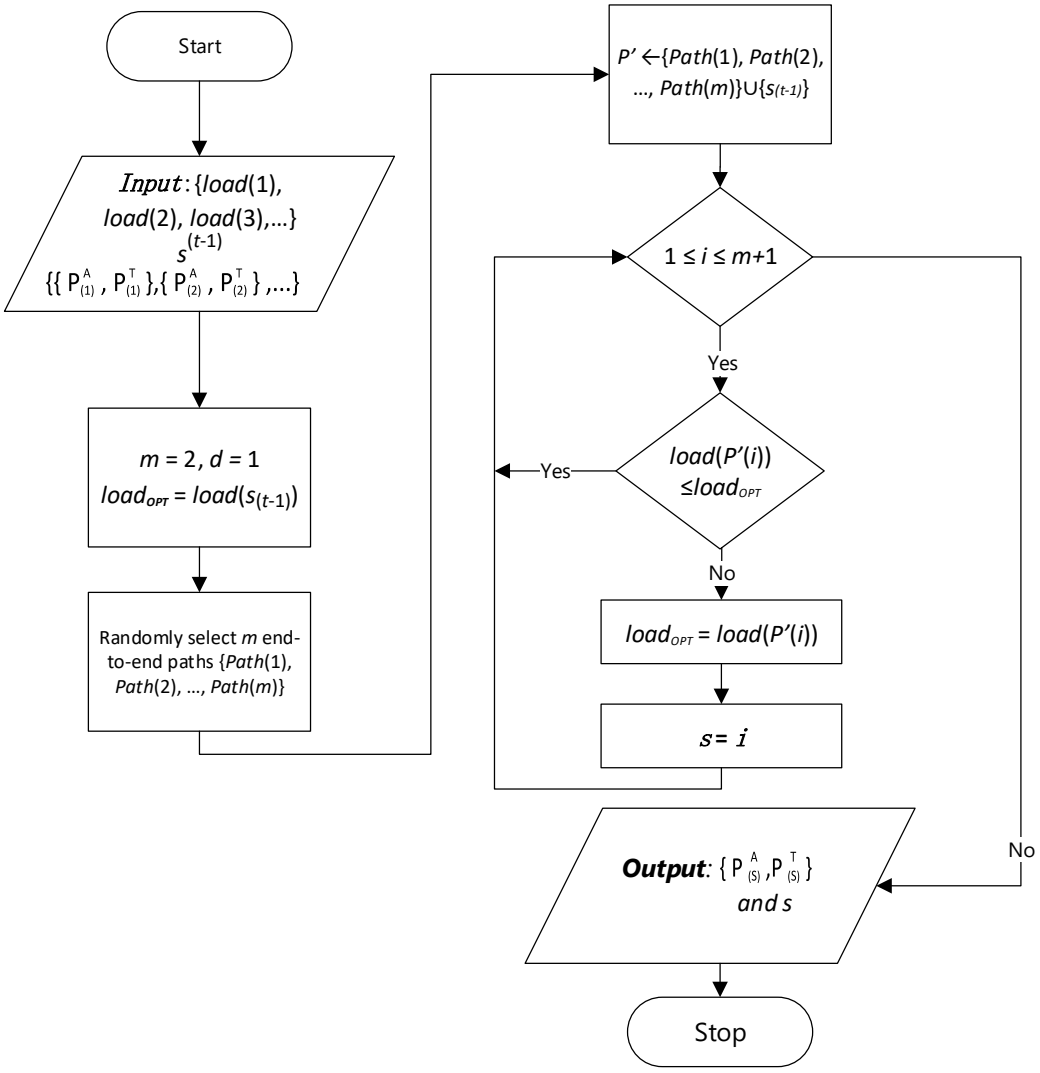


FIGURE 5.3: Flowchart for the scheduling algorithm in HOLMES$^+$

Clos DCN topology is abstracted as a queuing network. Then the performance of a specific leaf-to-spine DCN path is evaluated using a stochastic queuing network model.

Analyzing the stability of the scheduling process is focused on (a) when a packet arrives at a ToR switch and (b) when the packet reaches the spine switch. The packet experiences two queuing processes, at the ToR and the aggregate switch ports, respectively. The entire path from the ToR node to the spine node can also be modeled as a large queue.

Similar to [94], it is proven below that HOLMES$^+$'s scheduling algorithm is stable

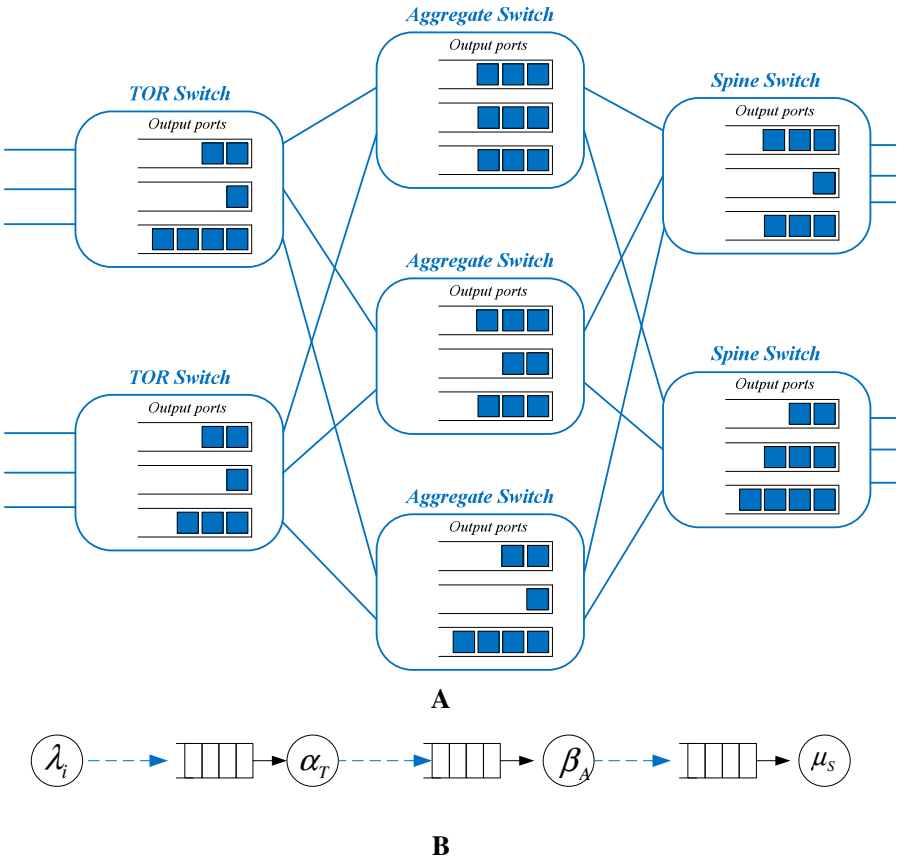


FIGURE 5.4: Abstraction of a leaf-to-spine path in a Clos network (A) to a serial queuing system (B) [12]

for all uniform and nonuniform independent packet arrivals. Some key notations and definitions used in the scheduling model are given in Table 5.2.

It is proven that the global HOLMES$^{+}(1,1)$ policy is stable for all admissible parallel arrival process.

Based on the result in [168] and Theorem 5.4, to prove the algorithm is stable, there should exist a negative expected single-step drift in the Lyapnuov function, *i.e.*, the expectation of $(L(t+1) - L(t))$ under the condition $L(t)$, denoted as $E[L(t+1) - L(t)|L(t)]$, satisfies

$$E[L(t+1) - L(t)|L(t)] \leq -\epsilon, \tag{5.29}$$

for some $\epsilon > 0$.

λ_i	Average data arrival rate in the ith leaf-to-spine path
α_i	Average data processing rate of the ToR port in the ith leaf-to-spine path
w_i	Average data arrival rate of the aggregate port in the ith leaf-to-spine path
β_i	Average data processing rate of the aggregate port in the ith leaf-to-spine path
N	The number of the ToR and aggregate switches in a pod
N_A	The number of spine switches connected by each aggregate switch
$Q_k(t)$	The number of accumulated packets in the buffer of the kth leaf-to-spine path at time t
$\hat{Q}_i(t)$	The number of accumulated packets in the buffer of the end-to-end path chosen by the ith input port using HOLMES$^+$ at time t
$Q^*(t)$	The number of the accumulated packets in the global least loaded leaf-to-spine path at time t
$Q_{iT}(t)$	The number of the accumulated packets on the ToR port of the ith leaf-to-spine path, at time t
$Q_{iA}(t)$	The number of the accumulated packets on the aggregate port of the ith leaf-to-spine path, at time t
$Q_r^*(t)$	The number of the accumulated packets on the ToR port of the global least loaded leaf-to-spine path at time t
$Q_A^*(t)$	The number of the accumulated packets on the aggregate port of the global least loaded leaf-to-spine path at time t

TABLE 5.2: Summary of key notations and definitions in scheduling model

A Lyapunov function L can be constructed as follows:

$$L(t) = \sum_{i=1}^{N \cdot N_A} (\hat{Q}_i(t) - Q^*(t))^2 + \sum_{i=1}^{N \cdot N_A} Q_i^2(t) \tag{5.30}$$

Then, it consists of two sub functions as:

$$L_1(t) = \sum_{i=1}^{N \cdot N_A} (\hat{Q}_i(t) - Q^*(t))^2 \qquad L_2(t) = \sum_{i=1}^{N \cdot N_A} Q_i^2(t) \tag{5.31}$$

Based on the above formulas, it can be proven that there exists a negative expected single-step drift in the Lyapunov function in each possible case. Therefore, the global HOLMES$^+(1,1)$ policy is stable. Based on the HOLMES$^+(d,m)$ policy, HOLMES$^+$'s scheduling algorithm is also stable. Please see details of the proof in Section 5.3.3.1.

5.3.3.1 Proof of the Stability

Definition 5.1 ([169, Chapter 13]). A system is called asymptotically stable around its equilibrium point at the origin if it satisfies the following two conditions:

1. Given any $\epsilon > 0$, $\exists \beta_1 > 0$ such that if $\|x(t_0)\| < \beta_1$, then $\|x(t)\| < \epsilon$, $\forall t > t_0$.
2. $\exists \beta_2 > 0$ such that if $\|x(t_0)\| < \beta_2$, then $x(t) \to 0$ as $t \to \infty$.

The first condition requires that the state trajectory can be confined to an arbitrarily small "ball" centered at the equilibrium point and of radius ϵ, when released from can arbitrary initial condition in a ball of sufficiently small (but positive) radius β_1. This is called stability in the sense of *Lyapunov*.

Definition 5.2 ([169, Chapter 13]). Let V be a continuous map from $\mathbb{R}^n$ to $\mathbb{R}$. $V(x)$ is called a locally positive definite (**lpd**) function around $x = 0$ if

1. $V(0) = 0$.
2. $V(x) > 0, 0 < \|x\| < r$ for some r.

Similarly, the function is called locally positive semidefinite (**lpsd**) if the strict inequality on the function in the second condition is replace by $V(x) \geq 0$. The function $V(x)$ is locally negative semidefinite (**lnsd**) if $-V(x)$ is **lpsd**.

Consider the continuous-time system

$$\dot{x}(t) = f(x(t)) \tag{5.32}$$

with an equilibrium point at $x = 0$.

Definition 5.3 ([169, Chapter 13]). Let V be an **lpd** function (a "candidate *Lyapunov* function"), and Let $\dot{V}$ be its derivative along trajectories of the system (5.32). If $\dot{V}$ is **lnsd**, then V is called a *Lyapunov function* of the system (5.32).

Theorem 5.4 ([169, Chapter 13]). *If there exists a Lyapunov function of the system (5.32), then $x = 0$ is a stable equilibrium point in the sense of Lyapunov. If in addition $\dot{V}(x) < 0, 0 < \|x\| < r_1$ for some r_1, i.e. if $\dot{V}$ is lnd, then $x = 0$ is an asymptotically stable equilibrium point.*

For a more detailed exposition about definitions 5.1, 5.2 and 5.3 and Theorem 5.4, refer to the lecture by Frazzoli [169].

Next, the *Lyapunov* Theorem 5.4 for local stability will be used to prove Theorem 5.5 as follows.

Theorem 5.5. *HOLMES$^+$(1, 1) policy is stable for all admissible packet arrival processes.*

PROOF. The key notations and definitions used in the proof are listed in Table 5.2. Consider the function $L_1(t)$ in (5.31); its upper bound can be derived as:

$$\begin{aligned} L_1(t) &= \sum_{i=1}^{N \cdot N_A} (\hat{Q}_i(t) - Q^*(t))^2 \\ &\leq 2 \times \left(\sum_{i=1}^{N \cdot N_A} (\hat{Q}_{iT}(t) - Q^*_{iT}(t))^2 + \sum_{i=1}^{N \cdot N_A} (\hat{Q}_{iA}(t) - Q^*_{iA}(t))^2 \right) \end{aligned}$$

Then, the upper bound for $E[L_1(t+1) - L_1(t)|L_1(t)]$ can be further derived as:

$$\begin{aligned} & E[L_1(t+1) - L_1(t)|L_1(t) \\ & \leq 2 \cdot \left(- \frac{1}{\sum_{i=1}^{N} \lambda_i + \sum_{j=1}^{N} \alpha_j} \cdot \frac{1}{NN_A} \cdot \sum_{i=1}^{N} \lambda_i (\hat{Q}_{iT}(t) - Q^*_{iT}(t))^2 \right. \\ & \quad \left. + \sum_{i=1}^{N} [2\sqrt{(\hat{Q}_{iT}(t) - Q^*_{iT}(t))^2}] + NN_AK + NN_A \right) \\ & \quad + 2 \cdot \left(- \frac{1}{\sum_{i=1}^{N_A} w_i + \sum_{j=1}^{N_A} \beta_j} \cdot \frac{1}{NN_A} \cdot \sum_{i=1}^{N_A} w_i (\hat{Q}_{iA}(t) - Q^*_{iA}(t))^2 \right. \\ & \quad \left. + \sum_{i=1}^{N_A} [2\sqrt{(\hat{Q}_{iT}(t) - Q^*_{iT}(t))^2}] + NN_AK + NN_A \right) \end{aligned}$$

Next, consider $L_2(t)$; the upper bound of the Lyapunov drift function $L_2(t)$ can be limited as follows:

$$
\begin{aligned}
&E[L_2(t+1) - L_2(t) | L_2(t)] \\
&\leq 2 \cdot \left(\sum_{i=1}^{N} \frac{\lambda_i}{\sum_{i=1}^{N} \lambda_i + \sum_{j=1}^{N} \alpha_j} \cdot (Q_{iT}(t+1) - Q_{iT}(t))^2 + \sum_{i=1}^{N} \frac{\alpha_i}{\sum_{i=1}^{N} \lambda_i + \sum_{j=1}^{N} \alpha_j} \cdot (Q_{iT}(t+1) - Q_{iT}(t))^2 \right) \\
&+ 2 \cdot \left(\sum_{i=1}^{N_A} \frac{w_i}{\sum_{i=1}^{N_A} w_i + \sum_{j=1}^{N_A} \beta_j} \cdot (Q_{iA}(t+1) - Q_{iA}(t))^2 + \sum_{i=1}^{N_A} \frac{\beta_i}{\sum_{i=1}^{N_A} w_i + \sum_{j=1}^{N_A} \beta_j} \cdot (Q_{iA}(t+1) - Q_{iA}(t))^2 \right) \\
&\leq 2 \cdot \left(\frac{1}{\sum_{i=1}^{N} \lambda_i + \sum_{j=1}^{N} \alpha_j} \sum_{i=1}^{N} \lambda_i \cdot (2\hat{Q}_{iT}(t) + 1) + \frac{1}{\sum_{i=1}^{N} \lambda_i + \sum_{j=1}^{N} \alpha_j} \sum_{i=1}^{N} \alpha_i \cdot (-2Q_{iT}(t) + 1) \right) \\
&+ 2 \cdot \left(\frac{1}{\sum_{i=1}^{N_A} w_i + \sum_{j=1}^{N_A} \beta_j} \sum_{i=1}^{N_A} w_i \cdot (2\hat{Q}_{iA}(t) + 1) + \frac{1}{\sum_{i=1}^{N_A} w_i + \sum_{j=1}^{N_A} \beta_j} \sum_{i=1}^{N_A} \beta_i \cdot (-2Q_{iA}(t) + 1) \right) \\
&= \frac{2}{\sum_{i=1}^{N} \lambda_i + \sum_{j=1}^{N} \alpha_j} \cdot \left(\sum_{i=1}^{N} \lambda_i \cdot (2Q^*_{iT}(t) + 2\sqrt{(\hat{Q}_{iT}(t) - Q^*_{iT}(t))^2} + 1) + \sum_{i=1}^{N} \alpha_i - 2 \cdot \sum_{i=1}^{N} \alpha_i Q_{iT}(t) \right) \\
&+ \frac{2}{\sum_{i=1}^{N_A} w_i + \sum_{j=1}^{N_A} \beta_j} \cdot \left(\sum_{i=1}^{N_A} w_i \cdot (2Q^*_{iA}(t) + 2\sqrt{(\hat{Q}_{iA}(t) - Q^*_{iA}(t))^2} + 1) + \sum_{i=1}^{N_A} \beta_i - 2 \cdot \sum_{i=1}^{N_A} \beta_i Q_{iA}(t) \right) \\
&= \frac{2}{\sum_{i=1}^{N} \lambda_i + \sum_{j=1}^{N} \alpha_j} \left(\sum_{i=1}^{N} \lambda_i \cdot (2\sqrt{(\hat{Q}_{iT}(t) - Q^*_{iT}(t))^2} + 1) \right. \\
&\qquad \left. + 2Q^*_{iT}(t)(\sum_{i=1}^{N} \lambda_i - \sum_{j=1}^{N} \alpha_j) + 2\sum_{i=1}^{N} \alpha_i (Q^*_{iT}(t) - Q_{iT}(t)) \right) \\
&+ \frac{2}{\sum_{i=1}^{N_A} w_i + \sum_{j=1}^{N_A} \beta_j} \left(\sum_{i=1}^{N_A} w_i \cdot (2\sqrt{(\hat{Q}_{iA}(t) - Q^*_{iA}(t))^2} + 1) \right. \\
&\qquad \left. + 2Q^*_{iA}(t)(\sum_{i=1}^{N_A} \beta_i - \sum_{j=1}^{N_A} w_j) + 2\sum_{i=1}^{N_A} \beta_i (Q^*_{iA}(t) - Q_{iA}(t)) \right)
\end{aligned}
$$

Thus, the upper bound of the Lyapunov drift function can be limited as:

$$
\begin{aligned}
&E[L_2(t+1) - L_2(t)|L_2(t)] \\
&\leq \frac{2\sum_{i=1}^{N}\lambda_i}{\sum_{i=1}^{N}\lambda_i + \sum_{j=1}^{N}\alpha_j} + \frac{4\sum_{i=1}^{N}\lambda_i\sqrt{(\hat{Q}_{iT}(t) - Q^*_{iT}(t))^2}}{\sum_{i=1}^{N}\lambda_i + \sum_{j=1}^{N}\alpha_j} \\
&\quad + \frac{4Q^*_{iT}(t)(\sum_{i=1}^{N}\lambda_i - \sum_{j=1}^{N}\alpha_j)}{\sum_{i=1}^{N}\lambda_i + \sum_{j=1}^{N}\alpha_j} + \frac{4\sum_{i=1}^{N}\alpha_i(Q^*_{iT}(t) - Q_{iT}(t))}{\sum_{i=1}^{N}\lambda_i + \sum_{j=1}^{N}\alpha_j} \\
&\quad + \frac{2\sum_{i=1}^{N_A} w_i}{\sum_{i=1}^{N_A} w_i + \sum_{j=1}^{N_A}\beta_j} + \frac{4\sum_{i=1}^{N_A} w_i\sqrt{(\hat{Q}_{iA}(t) - Q^*_{iA}(t))^2}}{\sum_{i=1}^{N_A} w_i + \sum_{j=1}^{N_A}\beta_j} \\
&\quad + \frac{4Q^*_{iA}(t)(\sum_{i=1}^{N_A} w_i - \sum_{j=1}^{N_A} w_j)}{\sum_{i=1}^{N_A} w_i + \sum_{j=1}^{N_A}\beta_j} + \frac{4\sum_{i=1}^{N_A}\beta_i(Q^*_{iA}(t) - Q_{iA}(t))}{\sum_{i=1}^{N_A} w_i + \sum_{j=1}^{N_A}\beta_j}
\end{aligned}
$$

For analytic tractability, assume that $N = N_A$. Combining the expression of the two Lyapunov drift functions, finally the expression of $E[L(t+1) - L(t)|L(t)]$ can

be gotten as follows:

$$
\begin{aligned}
&E[L(t+1)-L(t)|L(t)] \\
&\leq \sum_{i=1}^{N} \frac{-2(\sum_{i=1}^{N}\lambda_i+\sum_{j=1}^{N}\alpha_j)}{\lambda_i}\cdot\left(\frac{\lambda_i\sqrt{(\hat{Q}_{iT}(t)-Q_T^*(t))^2}}{N(\sum_{i=1}^{N}\lambda_i+\sum_{j=1}^{N}\alpha_j)}-(\frac{N\lambda_i}{\sum_{i=1}^{N}\lambda_i+\sum_{j=1}^{N}\alpha_j}+N)\right)^2 \\
&+\sum_{i=1}^{N}\frac{2N^2(\sum_{i=1}^{N}\lambda_i+\sum_{j=1}^{N}\alpha_j)}{\lambda_i}+\sum_{i=1}^{N}\frac{2N^2\lambda_i}{\sum_{i=1}^{N}\lambda_i+\sum_{j=1}^{N}\alpha_j}+\frac{2\sum_{i=1}^{N}\lambda_i}{\sum_{i=1}^{N}\lambda_i+\sum_{j=1}^{N}\alpha_j}+2N^2K+6N^2 \\
&+\sum_{i=1}^{N}\frac{-2(\sum_{i=1}^{N}w_i+\sum_{j=1}^{N}\beta_j)}{w_i}\cdot\left(\frac{w_i\sqrt{(\hat{Q}_{iA}(t)-Q_A^*(t))^2}}{N(\sum_{i=1}^{N}w_i+\sum_{j=1}^{N}\beta_j)}-(\frac{Nw_i}{\sum_{i=1}^{N}w_i+\sum_{j=1}^{N}\beta_j}+N)\right)^2 \\
&+\sum_{i=1}^{N}\frac{2N^2(\sum_{i=1}^{N}w_i+\sum_{j=1}^{N}\beta_j)}{w_i}+\sum_{i=1}^{N}\frac{2N^2w_i}{\sum_{i=1}^{N}w_i+\sum_{j=1}^{N}\beta_j}+\frac{2\sum_{i=1}^{N_A}w_i}{\sum_{i=1}^{N_A}w_i+\sum_{j=1}^{N_A}\beta_j}+2N^2K+6N^2 \\
&+\frac{4Q_{iT}^*(t)(\sum_{i=1}^{N}\lambda_i-\sum_{j=1}^{N}\alpha_j)}{\sum_{i=1}^{N}\lambda_i+\sum_{j=1}^{N}\alpha_j}+\frac{4\sum_{i=1}^{N}\alpha_i(Q_{iT}^*(t)-Q_{iT}(t))}{\sum_{i=1}^{N}\lambda_i+\sum_{j=1}^{N}\alpha_j} \\
&+\frac{4Q_{iA}^*(t)(\sum_{i=1}^{N_A}w_i-\sum_{j=1}^{N_A}\beta_j)}{\sum_{i=1}^{N_A}w_i+\sum_{j=1}^{N_A}\beta_j}+\frac{4\sum_{i=1}^{N_A}\beta_i(Q_{iA}^*(t)-Q_{iA}(t))}{\sum_{i=1}^{N_A}w_i+\sum_{j=1}^{N_A}\beta_j}
\end{aligned}
$$

Define the following parameters:

$$
A_{1i}=\frac{\lambda_i}{\sum_{i=1}^{N}\lambda_i+\sum_{j=1}^{N}\alpha_j},\ A_{2i}=\frac{w_i}{\sum_{i=1}^{N}w_i+\sum_{j=1}^{N}\beta_j}
$$

$$
B_1=2N^2K+6N^2+\sum_{i=1}^{N}\frac{2N^2(\sum_{i=1}^{N}\lambda_i+\sum_{j=1}^{N}\alpha_j)}{\lambda_i}+\sum_{i=1}^{N}\frac{2N^2\lambda_i}{\sum_{i=1}^{N}\lambda_i+\sum_{j=1}^{N}\alpha_j}+\frac{2\sum_{i=1}^{N}\lambda_i}{\sum_{i=1}^{N}\lambda_i+\sum_{j=1}^{N}\alpha_j}
$$

$$
B_2=2N^2K+6N^2+\sum_{i=1}^{N}\frac{2N^2(\sum_{i=1}^{N}w_i+\sum_{j=1}^{N}\beta_j)}{w_i}+\sum_{i=1}^{N}\frac{2N^2w_i}{\sum_{i=1}^{N}w_i+\sum_{j=1}^{N}\beta_j}+\frac{2\sum_{i=1}^{N}w_i}{\sum_{i=1}^{N}w_i+\sum_{j=1}^{N}\beta_j}
$$

Therefore:

$$
\begin{aligned}
&E[L(t+1) - L(t)|L(t)] \\
&\leq \sum_{i=1}^{N} -\frac{2}{A_{1i}} \left(\frac{A_{1i}}{N} \sqrt{(\hat{Q}_{iT}(t) - Q_T^*(t))^2} - (NA_{1i} + N) \right)^2 + B1 \\
&+ \sum_{i=1}^{N} -\frac{2}{A_{2i}} \left(\frac{A_{2i}}{N} \sqrt{(\hat{Q}_{iA}(t) - Q_A^*(t))^2} - (NA_{2i} + N) \right)^2 + B2 \\
&+ \frac{4Q_{iT}^*(t)(\sum_{i=1}^{N} \lambda_i + \sum_{j=1}^{N} \alpha_j)}{\sum_{i=1}^{N} \lambda_i + \sum_{j=1}^{N} \alpha_j} + \frac{4\sum_{i=1}^{N} \alpha_i (Q_{iT}^*(t) - Q_{iT}(t))}{\sum_{i=1}^{N} \lambda_i + \sum_{j=1}^{N} \alpha_j} \\
&+ \frac{4Q_{iA}^*(t)(\sum_{i=1}^{N_A} w_i + \sum_{j=1}^{N_A} \beta_j)}{\sum_{i=1}^{N_A} w_i + \sum_{j=1}^{N_A} \beta_j} + \frac{4\sum_{i=1}^{N_A} \beta_i (Q_{iA}^*(t) - Q_{iA}(t))}{\sum_{i=1}^{N_A} w_i + \sum_{j=1}^{N_A} \beta_j}
\end{aligned} \tag{5.33}
$$

The distribution of flows should satisfy the basic stable conditions for a queuing system; thus:

$$
\sum_{i=1}^{N} \lambda_i < \sum_{j=1}^{N} \alpha_j, \ \sum_{i=1}^{N_A} w_i < \sum_{j=1}^{N_A} \beta_j
$$

Consequently,

$$
\frac{4Q_{iT}^*(t)(\sum_{i=1}^{N} \lambda_i - \sum_{j=1}^{N} \alpha_j)}{\sum_{i=1}^{N} \lambda_i + \sum_{j=1}^{N} \alpha_j} + \frac{4Q_{iA}^*(t)(\sum_{i=1}^{N_A} w_i - \sum_{j=1}^{N_A} \beta_j)}{\sum_{i=1}^{N_A} w_i + \sum_{j=1}^{N_A} \beta_j} < 0 \tag{5.34}
$$

Suppose the queuing system does not converge to the global JSQ [170] [171] policy; there must exist a value t, so that the value of $L(t)$ is very large. Since $L(t) = L_1(t) + L_2(t)$, then at least one of the $L_1(t)$ and $L_2(t)$ is very large.

Case 1: $L_1(t)$ is very large and $L_2(t)$ is convergent

In this case:

$$\exists C_1, C_2,\ s.t.$$
$$[Q_{iT}(t) + Q_{iA}(t)] - [Q^*_{iT}(t) + Q^*_{iA}(t)] \leq C_1, \quad Q_{iT}(t) + Q_{iA}(t) \leq C_2$$

Then:

$$\begin{aligned}
&\frac{4\sum_{i=1}^{N}\alpha_i(Q^*_{iT}(t) - Q_{iT}(t))}{\sum_{i=1}^{N}\lambda_i + \sum_{j=1}^{N}\alpha_j} + \frac{4\sum_{i=1}^{N_A}\beta_i(Q^*_{iA}(t) - Q_{iA}(t))}{\sum_{i=1}^{N_A} w_i + \sum_{j=1}^{N_A}\beta_j} \\
&\leq Max\left\{\frac{\sum_{j=1}^{N}\alpha_j}{\sum_{i=1}^{N}\lambda_i + \sum_{j=1}^{N}\alpha_j}, \frac{\sum_{j=1}^{N_A}\beta_j}{\sum_{i=1}^{N_A} w_i + \sum_{j=1}^{N_A}\beta_j}\right\} \cdot C_2 \\
&\qquad - Min\left\{\frac{\sum_{j=1}^{N}\alpha_j}{\sum_{i=1}^{N}\lambda_i + \sum_{j=1}^{N}\alpha_j}, \frac{\sum_{j=1}^{N_A}\beta_j}{\sum_{i=1}^{N_A} w_i + \sum_{j=1}^{N_A}\beta_j}\right\} \cdot (C2 - C1) \\
&= C
\end{aligned} \tag{5.35}$$

Since:

$$L_1(t) \leq 2 \cdot \left(\sum_{i=1}^{NN_A}(\hat{Q}_{iT}(t) - Q^*_{iT}(t))^2 + \sum_{i=1}^{NN_A}(\hat{Q}_{iA}(t) - Q^*_{iA}(t))^2\right)$$

indicating that at least one of the local optimal solutions on the aggregate and the TOR switches are not convergent. Suppose the local solution on the TOR switch is not convergence, while the local solution on the aggregate switch is convergent. Then, there exists a constant D_1, satisfying:

$$|\hat{Q}_{iA}(t) - Q^*_A(t)| \leq D_1$$

Further:

$$\sum_{i=1}^{N} -\frac{2}{A_{2i}} \left(\frac{A_{2i}}{N} \sqrt{(\hat{Q}_{iA}(t) - Q_A^*(t))^2} - NA_{2i} + N) \right)^2$$
$$\leq \sum_{i=1}^{N} -\frac{2}{A_{2i}} \left(\frac{A_{2i}}{N} D_1 - (NA_{2i} + N) \right)^2 = D \tag{5.36}$$

Since the local solution on the TOR switch is not convergent, a constant t_1 can be found, satisfying:

$$\forall t \geq t_1, \sum_{i=1}^{N} \frac{2}{A_{1i}} \left(\frac{A_{1i}}{N} \sqrt{(\hat{Q}_{iT}(t) - Q_T^*(t))^2} - NA_{1i} + N) \right)^2 > B_1 + C + D \tag{5.37}$$

Substitute (5.34), (5.35), (5.36) and (5.37) into (5.33),

thus:

$$\forall t \geq t_1,\ E[L(t+1) - L(t)|L(t)] < 0$$

In other words:

$$\exists \epsilon > 0,\ s.t.\ E[L(t+1) - L(t)|L(t)] < -\epsilon$$

Hence, according to Theorem 5.4, a negative expected single-step drift has been found in the Lyapnuov function, which contradicts the hypothesis. Therefore the hypothesis is wrong and the local solution on the TOR switch is convergent. The stability of the local solution on the aggregate switch can also be proved; however, due to space limitations and brevity, the detailed proof is omitted for here. Therefore, $L_1(t)$ is convergent in all possible cases.

Case 2: $L_1(t)$ is convergent and $L_2(t)$ is very large

In this case, a value t_2 can be found, satisfying $Q_i(t)$ is very large when $t > t_2$. The global JSQ solution has proven to be convergence, and assume that the value Q*(t) is not overly large. Then, the difference of $Q_i(t)$ and $Q^*(t)$ should be large enough:

$$Q_i(t) - Q^*(t) = [Q_{iT}(t) + Q_{iA}(t)] - [Q_{iT}^*(t) + Q_{iA}^*(t)]$$

According to the above equation, at least one of $Q_{iT}(t)-Q^*_{iT}(t)$ and $Q_{iA}(t)-Q^*_{iA}(t)$ is large enough. $L_1(t)$ is convergence in this scenario, thus the constants E_1 and E_2 can be found, satisfying:

$$\sum_{i=1}^{N} -\frac{2}{A_{1i}}\left(\frac{A_{1i}}{N}\sqrt{(\hat{Q}_{iT}(t)-Q^*_T(t))^2}-(NA_{1i}+N)\right)^2+B_1\le E_1 \tag{5.38}$$

$$\sum_{i=1}^{N} -\frac{2}{A_{2i}}\left(\frac{A_{2i}}{N}\sqrt{(\hat{Q}_{iA}(t)-Q^*_T(t))^2}-(NA_{2i}+N)\right)^2+B_2\le E_2 \tag{5.39}$$

Suppose the local solutions on the TOR switches and aggregate switches are both divergent, then:

$$Q_{iT}(t)>Q^*_{iT}(t),\ Q_{iA}(t)>Q^*_{iA}(t)$$

$$\exists t\ge t_2,\ \frac{4\sum_{i=1}^{N}\alpha_i(Q^*_{iT}(t)-Q_{iT}(t))}{\sum_{i=1}^{N}\lambda_i+\sum_{j=1}^{N}\alpha_j}+\frac{4\sum_{j=1}^{N_A}\beta_j(Q^*_{iA}(t)-Q_{iA}(t))}{\sum_{i=1}^{N_A}w_i+\sum_{j=1}^{N_A}\beta_j}\le-(E_1+E_2) \tag{5.40}$$

Substitute (5.38), (5.39), (5.40) into (5.33),

$$\forall t\ge t_2,\ E[L(t+1)-L(t)|L(t)]<0$$

In other words,

$$\exists\epsilon>0,\ s.t.\ E[L(t+1)-L(t)|L(t)]<-\epsilon$$

Thus, according to Theorem 5.4, a negative expected single-step drift has been found in the Lyapnuov function, which contradicts the hypothesis. Suppose only the local solution on the TOR switch is convergent; then, set an upper bound F_1, satisfying:

$$Q^*_{iA}(t)\le F_1$$

In this case,

$$\frac{4\sum_{j=1}^{N_A}\beta_j(Q^*_{iA}(t)-Q_{iA}(t))}{\sum_{i=1}^{N_A}w_i+\sum_{j=1}^{N_A}\beta_j}\le\frac{4F_1\sum_{j=1}^{N_A}\beta_j}{\sum_{i=1}^{N_A}w_i+\sum_{j=1}^{N_A}\beta_j}=F \tag{5.41}$$

Further:

$$\exists t \geq t_2,\ \frac{4\sum_{i=1}^{N}\alpha_i(Q_{iT}^{*}(t)-Q_{iT}(t))}{\sum_{i=1}^{N}\lambda_i+\sum_{j=1}^{N}\alpha_j} \leq -(E_1+E_2+F) \tag{5.42}$$

Substitute (5.38), (5.39), (5.41), (5.42) into (5.33):

$$\forall t \geq t_2,\ E[L(t+1)-L(t)|L(t)] < 0$$

In other words:

$$\exists \epsilon > 0,\ s.t.\ E[L(t+1)-L(t)|L(t)] < -\epsilon$$

Thus, according to Theorem 5.4, a negative expected single-step drift has been found in the Lyapnuov function, which contradicts the hypothesis. Therefore the hypothesis is wrong and $L_2(t)$ is convergent in all possible cases.

Based on the above modeling results, a negative expected single-step drift has been found in the Lyapnuov function in all possible cases. Therefore, according to Theorem 5.4, the HOLMES$^+(1,1)$ policy is stable, and based on the HOLMES$^+(d,m)$ policy, the HOLMES$^+$'s scheduling algorithm is also stable. □

5.4 HOLMES$^+$ Performance Evaluation

HOLMES$^+$ is evaluated by simulation based on OMNET++. A test-bed simulation platform is constructed to simulate the data transmission process in symmetric and asymmetric Fat-Tree DCNs. Similar to [43], a heavy-tailed distribution is used to generate DC flow of different sizes. The hosts of the DCN run TCP applications, and the flow request rate of each TCP connection satisfies the Poisson process.

By comparing the performance of the same DCN under different load balancing mechanisms, HOLMES$^+$ is validated and analyzed by the influence of the network partition solution and end-to-end scheduling schemes on DCN performance (see

Section 5.4.1, page 150). Moreover, the adaptability of HOLMES$^+$'s load balancing algorithm for heterogeneous architectures is evaluated (see Section 5.4.3, Page 156) and discussed regarding some technical issues in the hardware implementation of HOLMES$^+$ in Section 5.4.4.

5.4.1 Evaluation of HOLMES$^+$ Network Partition

The HOLMES$^+$ network partition policy is evaluated in a scenario where hybrid elephant and mouse flows are scheduled in the same DCN with different scheduling schemes. The DCN topology deployed in the test-bed is a Clos network with two and four leaf and spine switches respectively, as shown in Figure 5.10. The elephant and mouse flows are generated to leaf switches with average message sizes of 100KB and 1KB, respectively. The queue lengths of the switch ports are used as performance indicators.

Since the scale of the DCN in the simulation is not very large, HOLMES$^+$ deploys a static policy that partitions the two subnetworks in advance. The buffer sizes of all the switch ports are set to be the same. When the buffer of a switch port is full, all the upcoming input packets to that port will be dropped. HOLMES$^+$ is compared against two state-of-the-art unified load balancing schemes: CONGA [43] and queue length gradient based scheduling, TIMELY [96]. CONGA proposes the spraying of DC traffic among multi-rooted networks based on the congestion state of each end-to-end DC path, while TIMELY [96] uses a delay gradient as the congestion signal and proposes a gradient-based algorithm to jointly optimize the latency and throughput of a DCN in different time periods. Similar to the delay gradient based congestion control policy used in [96], the queue length gradient is deployed as the indicator and schedules an arriving packet to the port with the minimum length gradient.

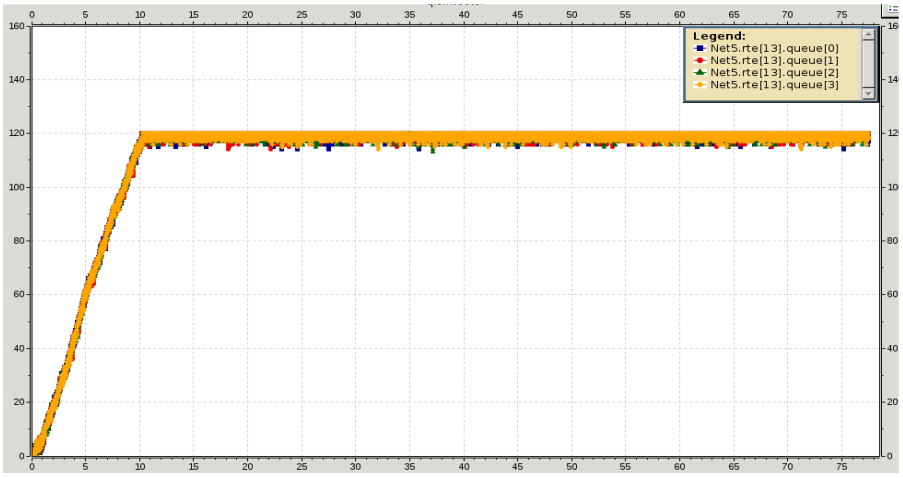
Figures 5.5A-C show the queue length variation of the four ports of a spine switch under the three scheduling schemes. The X-axis indicates the time period and the Y-axis denotes the queue length. It is shown in Figure 5.5A that using CONGA,

the buffers of all the four ports are full after a period of time, indicating that the throughput of the switch has been fully utilized, which benefits the transmission of the elephant flows. However, when a mouse flow arrives, all the packets in that flow have a long queuing time since all the output ports are heavily loaded. Consequently, the latency of the mouse flow will increase and degrade the overall performance of the hybrid DC flows.

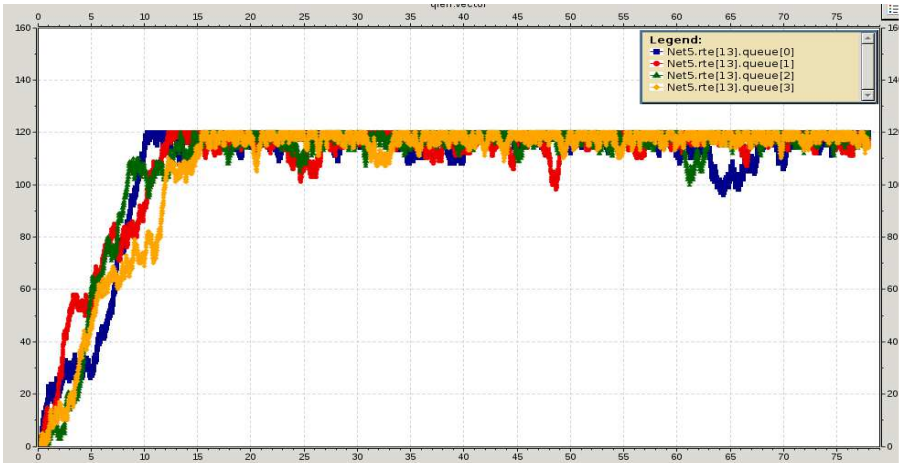
Similarly, as shown in Figure 5.5B, the buffers of the four output ports are also almost full after a period of time when using the length gradient based policy. The results indicate that the length gradient based load balancing policy still suffers from interference between the elephant flows and the mouse flows.

Comparing Figure 5.5A and Figure 5.5B, it is shown that the load balancing condition of the gradient-based scheme is a little worse than that of CONGA [43]. The reason is that the gradient-based scheme schedules the DC flows according to the changing trend but not the current state of the DCN. Finally, Figure 5.5C shows that HOLMES^{+} has successfully isolated the elephant and mouse flows. Two of the ports have been partitioned to the low latency sub-network and used for transmitting the mouse flows. Figure 5.5C shows that the buffers of the two ports are almost empty during the entire transmission procedure. Thus, packets in the mouse flows do not need additional queuing delays, and the low latency of the mouse flow is ensured. Moreover, the buffers of the other high throughput ports are also full-filled, which satisfies the throughput requirements of the elephant flows. Hence, by isolating the mixed traffic, HOLMES^{+} network partition policy successfully eliminates the interference of the elephant flows with the mouse flows.

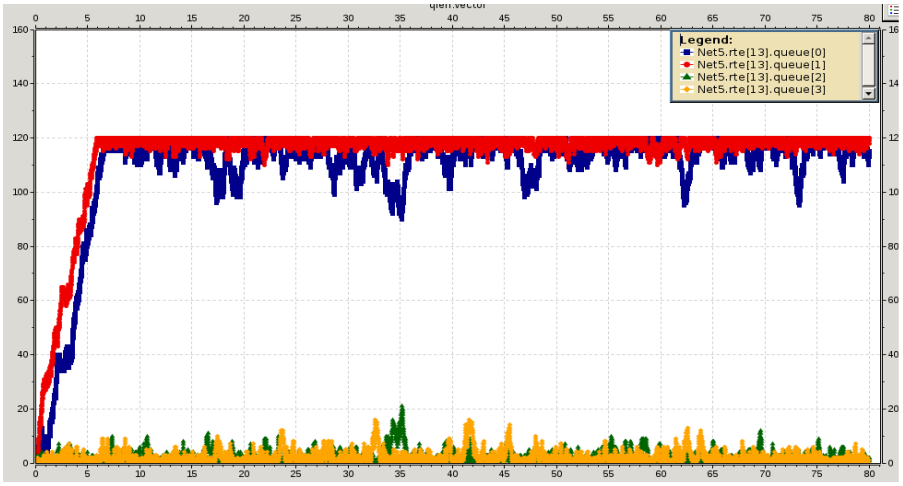
The main shortcoming of the network partition solution is the inefficient use of network resources. Although isolation of the hybrid traffic avoids the interactions of the elephant and mouse flows, the spare network resource in the low latency paths has not been entirely used since the buffers of these paths are almost empty. An effective solution is to improve the buffer allocation by limiting the buffer size of the low latency subnetwork and assigning the spare buffers to the high-throughput sub-network. This policy was implemented in [99].



A



B



C

FIGURE 5.5: Queue length change trends of a DCN spine node's ports under CONGA (A), length-gradient-based policy (B) and HOLMES^{+} (C) [12].

5.4.2 Stability Validation of HOLMES$^+$ Scheduling Algorithm

The stability of the HOLMES$^+$ flow scheduling algorithm is evaluated, and the scenario that DC traffic is scheduled in a DC with asymmetric network topology is simulated. Two different sized Clos networks are combined, and an asymmetric DCN architecture is constructed. One of the Clos networks consists of two leaf switches and four spine switches like the topology in Figure 5.10. The other is a Clos network with four leaf and five spine switches, ten hosts are attached to each leaf switch. Validating the stability of HOLMES$^+$ flow scheduling algorithm is concentrated on in this scenario, rather than the network partition mechanism. Thus, only the HOLMES$^+$ flow scheduling algorithm is executed in the experiment instead of deploying HOLMES$^+$ network partition mechanism.

CONGA [43] and DRILL [94] are two load balancing solutions proven to be stable. Therefore, they are chosen to be compared against HOLMES$^+$. All DC traffic is scheduled with the granularity of packet, and focus on analyzing the stability of the three scheduling algorithms. Uniformly set $d = 2$ and $m = 1$ when using the P2C selections. Similarly to the previous experiments, deploy queue length as the load balance indicator to evaluate the overall load balancing condition of the DCN.

Figures 5.6, 5.7 and 5.8 show the queue length changing trend of a specific leaf switch's ports, under the load balancing policies CONGA, DRILL and HOLMES$^+$. It is shown in Figure 5.6 that the queue length change trends of all the ports in a leaf switch almost overlap under CONGA, indicating that the queue lengths of all the switch ports are almost the same at each time unit. Therefore, the load balancing condition under CONGA is optimal among all the three mechanisms, since CONGA makes each scheduling decision based on the global congestion information. Without considering the time used for obtaining the congestion information, CONGA obtains the global optimal load balancing result. Figure 5.7 shows the queue length change trends of the same leaf switch ports under DRILL. Different from the former results, there are fluctuations in the queue length change curve.

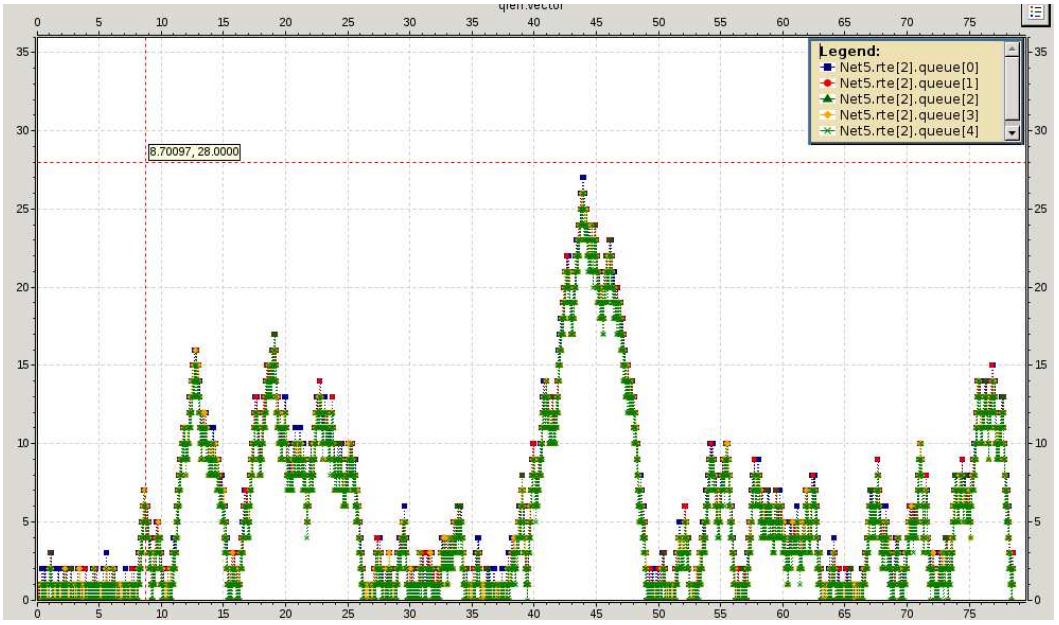


FIGURE 5.6: Queue length changing trends of a DCN leaf node's ports under load balancing policy CONGA [12]

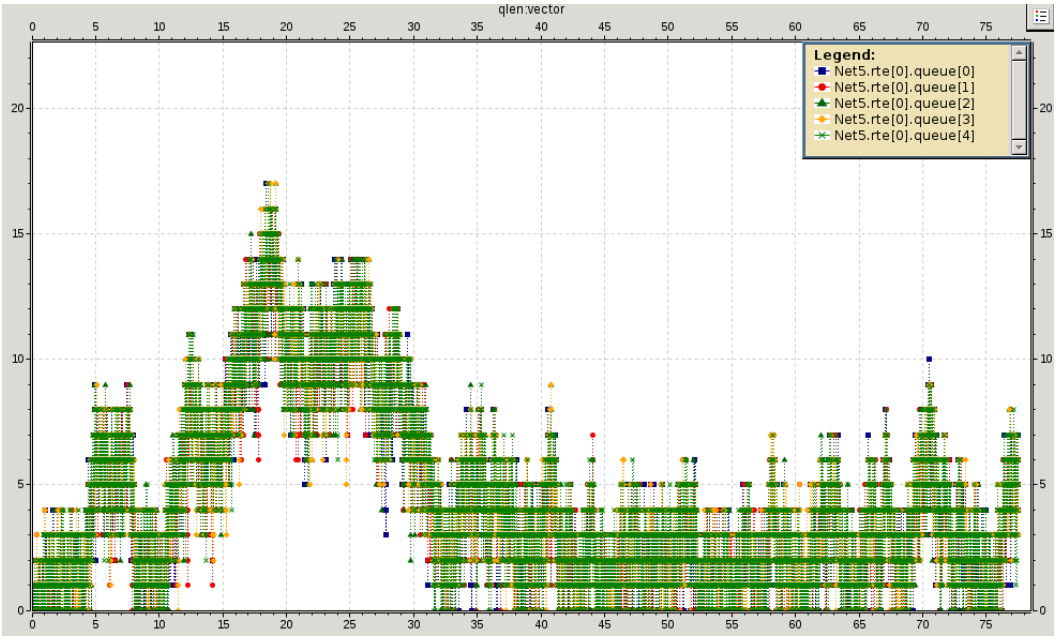


FIGURE 5.7: Queue length changing trends of a DCN leaf node's ports under load balancing policy DRILL [12]

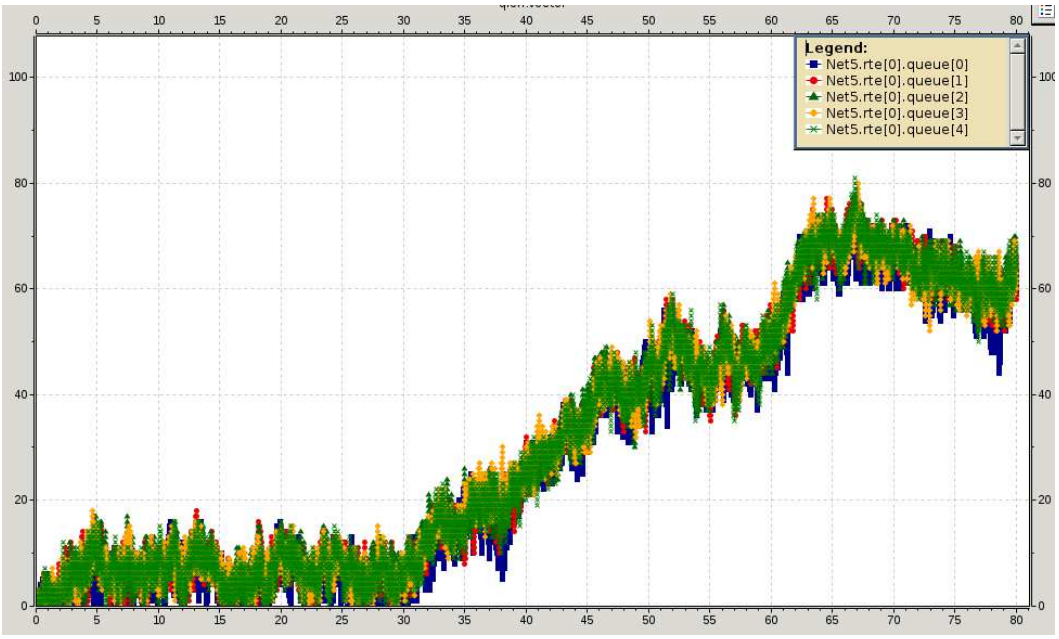


FIGURE 5.8: Queue length changing trends of a DCN leaf node's ports under load balancing policy HOLMES$^+$ [12]

In other words, the difference in length between the longest queue and the shortest queue is clear. The reason is that the use of the (d, m) policy in DRILL [94] reduces the scale of the solution space, and the local optimal solutions affect the load balancing condition of the DCN.

Figure 5.8 shows the queue length change trends of the same leaf switch ports under the HOLMES$^+$ scheduling algorithm. Fluctuations also exist in the graph in Figure 5.8, and the amplitude of the fluctuation is more obvious. This phenomenon is also caused by the use of the HOLMES$^+(d, m)$ policy. Compared with DRILL, the global HOLMES$^+(d, m)$ policy used in HOLMES$^+$ further limits the solution space, and exacerbates the fluctuations. However, although the fluctuations are more obvious when executing HOLMES$^+$ flow scheduling algorithm, an upper bound (about 10 packets) for the fluctuation amplitude can also be observed, indicating that the difference in length of the shortest and the longest queues are not infinite in HOLMES$^+$. Thus, our HOLMES$^+$ flow scheduling algorithm is stable during the whole scheduling period. Moreover, limiting the solution exploration space reduces the time used to obtain the congestion information, making HOLMES$^+$ more efficient and applicable for large-scale Data Centers.

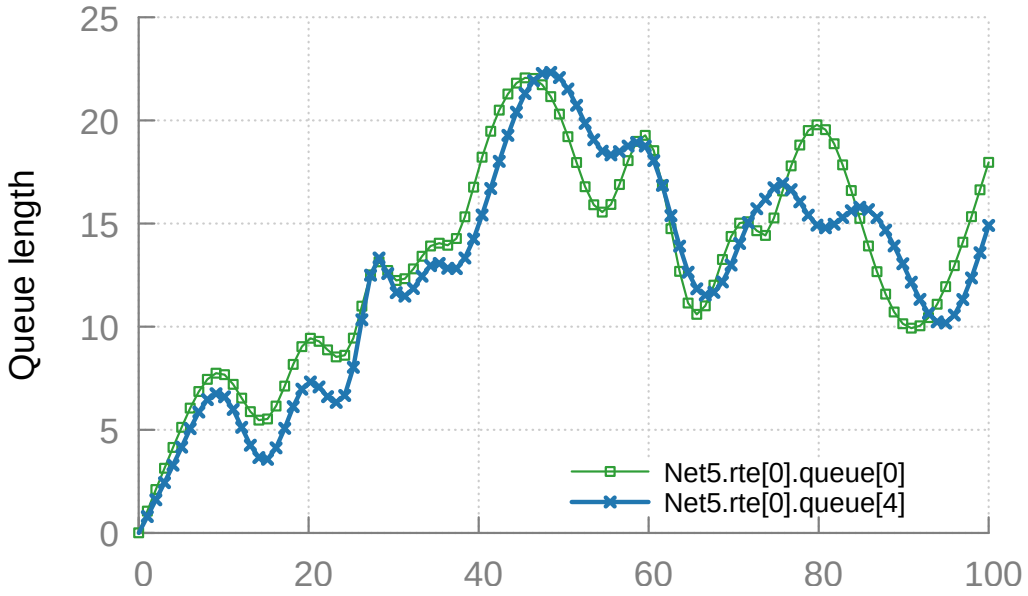


FIGURE 5.9: Queue length changing trends of a DCN leaf node's ports under load balancing policy HOLMES$^{+}(d, m+1)$

The HOLMES$^{+}(d, m+1)$ policy maintains the congestion states of d new randomly chosen paths, m congestion-optimal paths, and one local optimal path at the latest time unit for each switch. As shown in Figure 5.9, the amplitude of the fluctuation is much reduced. Using the HOLMES$^{+}(d, m+1)$ policy can balance the local and global network statuses and make quick decisions at microsecond time scales.

5.4.3 Adaptability for Heterogeneous

The case of DCNs that are asymmetric and consist of switches with disparate capabilities (*e.g.*, having a different number of ports as well some variations in linespeed) introduces even greater complexity. The adaptability of a scheduling algorithm allows the ability to adjust to heterogeneous DCNs and keep the same performance. The evaluation of the adaptability of the stochastic flow scheduling algorithm and of the weighted traffic splitting solutions [92] is done as follows.

A theoretical comparison of the approximate adaptability of the two solutions is done. Consider a simple leaf-spine DCN topology with N paths available between

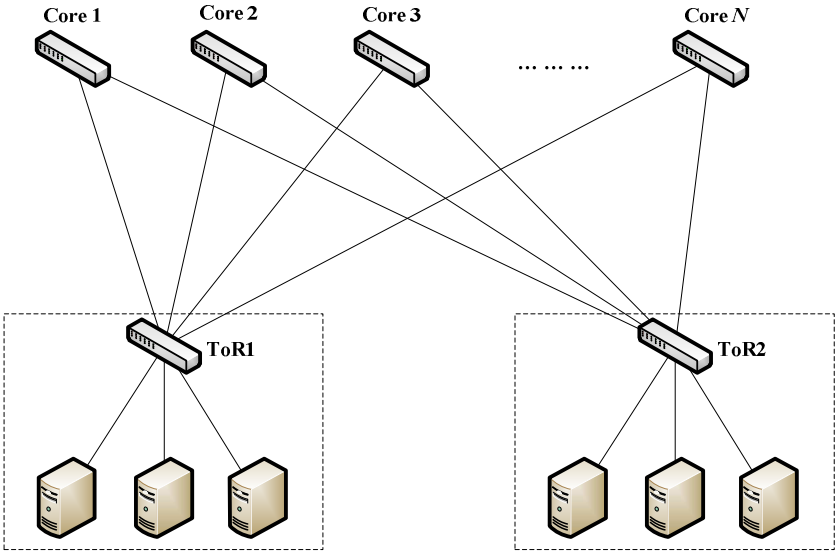


FIGURE 5.10: Simple leaf-spine topology for adaptability evaluations [12]

two racks as shown in Figure 5.10. The loads of the N paths (number of packets accumulated in the buffers) are unevenly distributed. A comparison of the adaptability of the two mechanisms under the linear distribution of path loads is done.

For analytical tractability, assume that the number of packets accumulated in the buffer of each path are: 0, 1, 2, ..., $N-1$ and that the buffer size of each path is also set to N. Next, calculate the probabilities that one arriving packet is scheduled to the least loaded path under the HOLMES$^{+}(2,1)$ policy as well as the weighted traffic splitting policy.

Given the probability $P1$ that a packet is scheduled to the least loaded path under the HOLMES$^{+}(d,m)$ policy, HOLMES$^{+}(d=2,m=1)$ can be calculated as:

$$P_1 = C_{N-1}^{d-1}/C_N^d = \frac{d}{N} = \frac{2}{N} \quad (d=2)$$

When deploying the weighted traffic splitting solution, the probability that one packet is assigned to a specific path is proportional to the available capacity of

that path. Then, the probability that the packet chooses the least loaded path is:

$$P_2 = \frac{N}{N + (N-1) + (N-2) + \cdots + 1} = N/C_{N+1}^{2} = \frac{2}{N+1}$$

When the DCN scale is large, the value of N is sufficiently large, and then the value of $P1$ and $P2$ are almost equal. Thus, the adaptability of the HOLMES$^{+}(2,1)$ policy and the weighted traffic splitting policy for heterogeneous DCNs are almost the same when the path loads satisfy linear distributions. However, compared with the weighted solution that maintains the load status of all the N paths, the memory complexity is dramatically reduced when using stochastic flow scheduling mechanisms.

Next, consider another scenario, in which the load distribution of the DCN paths shows stronger heterogeneity. Suppose the path loads satisfy an exponential distribution and that the numbers of spare buffers in N paths are:

$$N, \frac{N}{2}, \frac{N}{4}, \frac{N}{2^3}, \cdots, \frac{N}{2^{N-1}}$$

Using the weighted traffic splitting policy, the probability that a packet chooses the least loaded path is:

$$P_2 = \frac{N}{N + \frac{N}{2} + \frac{N}{4} + \frac{N}{2^3} + \cdots + \frac{N}{2^{N-1}}} = \frac{N}{2N - \frac{N}{2^{N-1}}} = \frac{1}{2 - \frac{1}{2^{N-1}}}$$

On the other hand, when deploying the HOLMES$^{+}(d,m)$ policy, the probability becomes:

$$P1 = C_{N-1}^{d-1}/C_N^d = \frac{d}{N}$$

When $d = N/2$, the below can be gotten:

$$P_2 \approx P_1 = \frac{1}{2}$$

Therefore, when the load conditions of the DC paths are heavily heterogeneous, the HOLMES$^{+}(d, m)$ policy also needs to maintain plenty of load status information to keep its adaptability as equal to that of the weighted traffic splitting solutions. The stochastic scheduling mechanism does not show obvious advantages in this scenario.

Evaluations of the two approaches have been also made by some other researchers. Key *et al.* [172] proposed a coordinated solution that periodically samples for new paths via randomly selecting routers at the higher level, shifts the traffic load to the paths with the lowest loss rates (or lowest ECN rates, largest delays), then equates the marginal utility of its aggregate data rate to the loss rate on these "best" paths at the lower level. This approach can be considered as an improvement over the traditional weighted traffic splitting solutions, since it considers the real-time changes in the load states. Moreover, Key *et al.* [172] showed that the approach outperforms the P2C policy, and that it does almost as well as if each user could see the global list of resources.

Similar to the experiment in [172], simulate the execution process of the coordinate approach as well as the P2C algorithm on the same switch. Figure 5.11 shows the change trend of the overall switch load as the modeling factor (d) increases.

The load distribution of the switch ports is initialized exponentially in this experiment. Figure 5.11 shows that, as the value of d increases, the load condition of the switch is improved under the P2C policy ($HOLMES^{+}(d, m)$ policy); since a larger value of d increases the probability of choosing the lightest loaded output port. When increasing the value of d from 2 to 5, the P2C policy performs almost as well as the theoretical load optimal policy with global information ($d = 10$), which validates our previous modeling results in Section 5.4.2. In contrast, when using the coordinated approach, the switch attains optimal performance when the value of d is small ($d = 2$) and the load state of the switch is almost as good as the theoretical optimal load policy. This simulation result is in accordance with the analysis in [172]. However, as the value of d increases, the load state of the switch becomes worse; and when assigning $d = 10$, the load balancing condition of

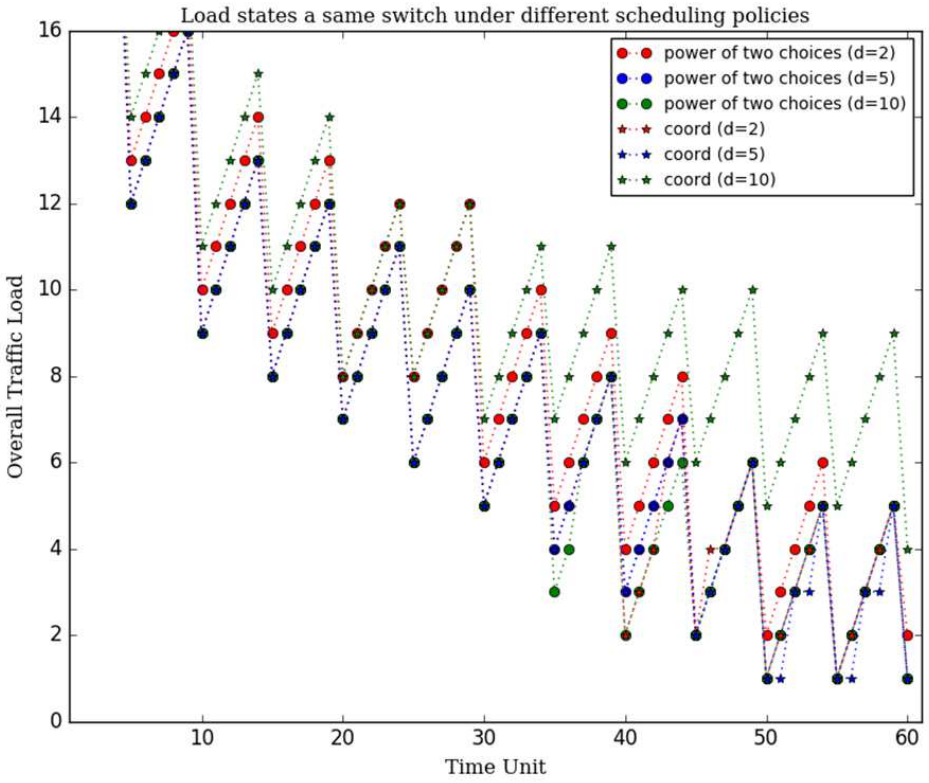


FIGURE 5.11: Changing trend of a switch's overall traffic load under different policies [12]

the switch under the coordinated policy is even worse than the HOLMES$^+(2,1)$ policy.

Although stochastic flow scheduling outperforms the weighted traffic splitting solution [92] in most cases, it still has some limitations. The weighted traffic splitting solution maintains the load status of all the paths, and dynamically adjusts the value of each weight according to the current load status of each path (the static weight configuration was proven to not be applicable in [43]). However, when deploying the HOLMES$^+(d,m)$ policy, the values of d and m are constant after the initialization. Thus, when the values are not appropriately assigned, the HOLMES$^+(d,m)$ policy will not perform as well as the weighted traffic-splitting solution. Hence, the HOLMES$^+$ AI module is responsible for analyzing the overall degree of heterogeneity of a DCN and guiding the flow scheduling algorithm to set appropriate values for the algorithm factors (d and m). The detailed design and implementation of the HOLMES$^+$ AI module is intended for our future work.

5.4.4 Technical Challenges in Hardware Implementations

Some technical challenges need to be considered in order to implement HOLMES$^+$ in real-world Data Centers. Now these challenges are summarized in hardware implementations.

5.4.4.1 Handling the packet reordering

The flow scheduling algorithm in HOLMES$^+$ can be implemented with different data granularities: per packet scheduling, per-flow scheduling or some intermediate data, e.g., flowcell [116], flowlet [43], etc. Using the TCP and implementing per-packet (or flowcell) scheduling, Dixit *et al.* [115] showed that fine-grained traffic-splitting techniques can cause packet reordering and lead to severe TCP throughput degradation. Therefore, the packet reordering problem needs to be considered when implementing the fine-grained HOLMES$^+$ traffic scheduling algorithm. A viable solution is to deploy the JUGGLER [173] network stack in Data Center traffic transmissions. JUGGLER exploits the small packet delays in data-center networks and the inherent traffic bursts to eliminate the negative effects of packet reordering, while keeping the state for only a small number of flows at any given time. This practical reordering network stack is lightweight and can handle packet reordering efficiently.

5.4.4.2 Design of DCN forwarding and state tables

The design of the forwarding and state tables is also a noteworthy challenge. An appropriate approach should cope with a small time budget as well as small register usage. A viable design is proposed to implement the per-flow scheduling algorithms of HOLMES$^+$ in the following.

As shown in Figure 5.12, a ToR switch maintains a flow table and a state table. A new flow table entry is set up by applying the HOLMES$^+$(2, 1) policy in the state table. The period timer of each table is triggered every time period T1 and

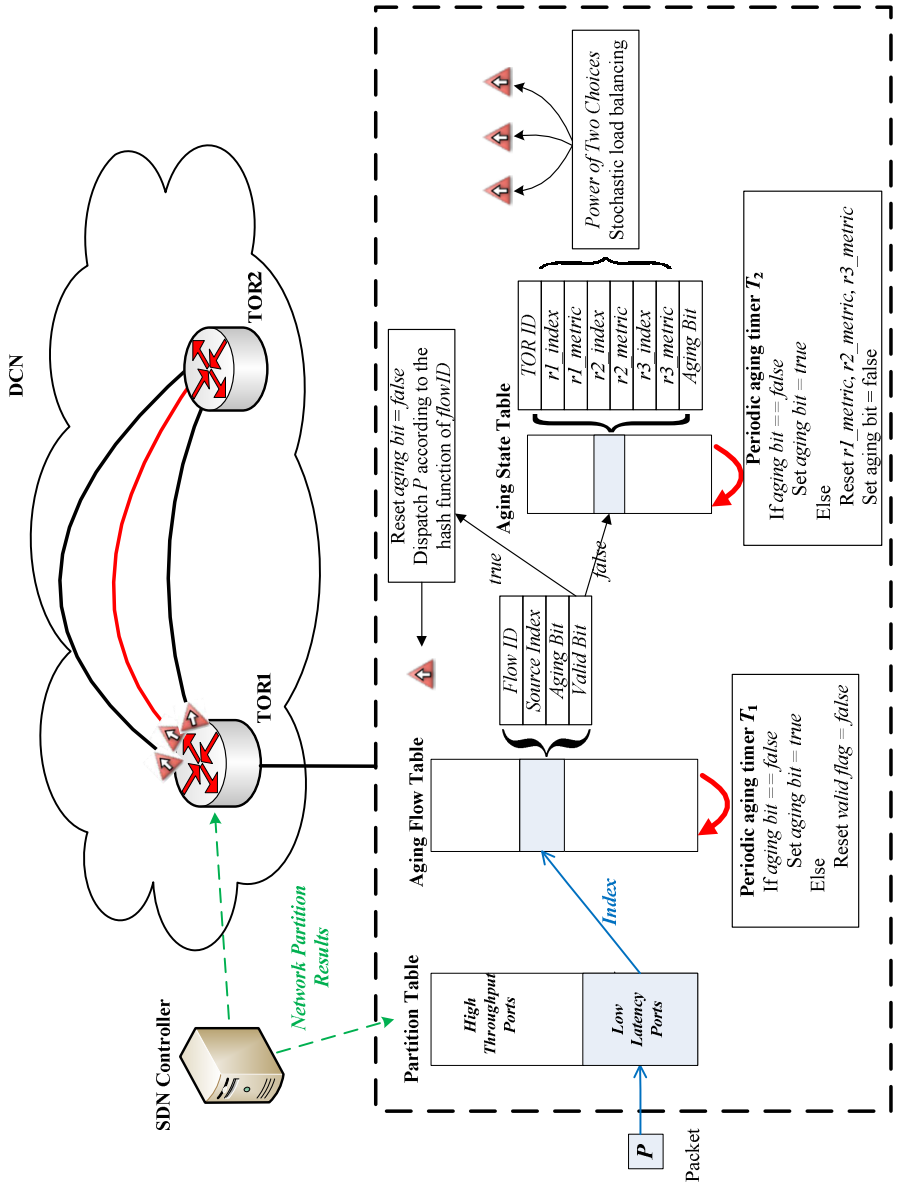


FIGURE 5.12: Overview of HOLMES$^+$ forwarding and state tables [12]

T2 in order to age out inactive flows and update the load status of each candidate end-to-end path. The two tables work together to execute the load balancing policy from the SDN controller. Specifically, following a flowchart, as shown in Figure 5.13, when a packet arrives, its flowID is hashed to map the packet to a flow table entry. If the table entry is valid, the packet is dispatched to the path indicated by the stored hash applied to the packet's flow ID. On the contrary, if the packet's flowID is not maintained in the flow table, the ToR switch will look up the destination ToR ID in the state table. After that, the HOLMES$^+(d, m)$ policy is applied to compare the load states of the three candidate end-to-end paths to the

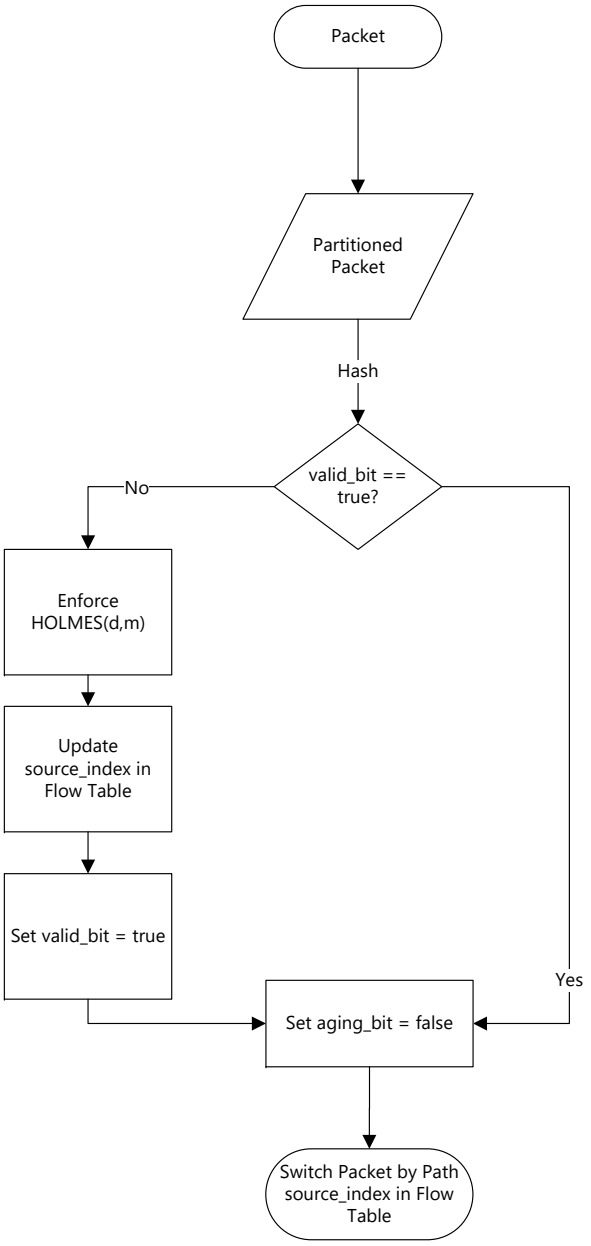


FIGURE 5.13: The flowchart of forwarding a packet in HOLMES$^+$

destination ToR (r1_metric, r2_metric and r3_metric as depicted in Figure 5.12), and assign the packet to the optimal path. Two of the three end-to-end paths are randomly selected, and the third one is the optimal path from the last selection. Finally, the flow ID and the hash of the chosen path will be inserted into the flow table.

The information in each table needs to be updated periodically in order to keep the real-time status of the traffic and paths. Thus, associate an aging bit with each

table entry. The aging bit of the flow table is responsible for marking inactive or idle flows; when a packet's flow ID maps the information in the forwarding table, the aging bit is cleared to indicate that the flow entry is active. A hardware timer visits every table entry every aging timeout $T1$. When the timer process visits a table entry, it either turns on the aging bit or invalidates the entry if the aging bit is already on. In other words, $T1$ is the timeout threshold to age out inactive flows, which is proportional to the size of the scheduling unit, e.g., per-flow, per-flowcell, etc. If the packet's flow ID is not maintained in the flow table, the ToR switch will execute the HOLMES$^{+}(d, m)$ policy on the state table. Thus, $T1$ can also be considered as the time period at which to trigger the execution of the HOLMES^{+} scheduling algorithm. On the other hand, another timer process runs together with the aging bit of the state table to update the load status of each candidate end-to-end path. The timeout threshold to age out the old status information in the state table is set to $T2$. To ensure that the latest load state information is used when executing the HOLMES$^{+}(d, m)$ policy, the value of $T1$ and $T2$ should satisfy: $T1 \geq T2$. Moreover, in most cases, the global congestion control signals deployed in the flow scheduling algorithms are the feedback signals from the receivers of the end-to-end paths. Thus, and further: $T1 \geq T2 \geq RTT$. Key *et al*. [174] suggested that the policy that periodically samples a random path and retains the best paths may perform well.

Periodic sampling of the path congestion states in the state table causes the real-time collection of status information to become a technical challenge, since the state collection operations should not introduce obvious transmission overheads and performance penalties. Especially in the TCP incast scenarios [175] where multiple source nodes transmit traffic at the same time to a common destination node, the state collection operations introduce additional traffic and are prone to causing DCN throughput collapse. A viable solution for collecting real-time congestion statuses is deploying RepSYN [120] as a signal to detect the load conditions of the multiple paths: before transmitting data among multi-rooted paths, multiple TCP connections are established; however, traffic is only transmitted using the first established connection and the other connections are ended immediately. The

delay experienced by a SYN packet reflects the latest congestion condition of the corresponding path, and thus the congestion states can be collected. Moreover, this solution [120] only replicates SYN packets to probe the network, which does not aggravate the TCP incast in a DCN.

Two types of dedicated probe packets are adopted in HOLMES^{+}. A *Probe-Request* packet can be sent along a chosen path to the destination ToR switch. The *Probe-Request* packet collects related congestion information along the path, and finally the destination ToR switch sends back a *Probe-Ack* packet to the source ToR switch. In every period $T2$, the state table needs to maintain the congestion states of d new randomly chosen paths and m congestion-optimal paths in the latest time unit. The period aging timer is triggered every time interval $T2$ to send a *Probe-Request* packets along each of another set of randomly chosen d paths. The source ToR switch updates the state table when it receives the *Probe-Request* packets, and finally, chooses an optimal one from $(d+m)$ paths to replace the source index in the flow table as depicted in Figure 5.12. The source index indicates which path the flow will be switched along to the destination.

Compared with some other congestion-aware solutions, e.g., CONGA [43], the storage complexity has been dramatically reduced, as only $(d+m)$-paths' information needs to be stored per a pair of ToR switches. Besides, an aging mechanism can be employed in the state table as depicted in Figure 5.12 in order to further save memory space. In order to make the scheduling results of the HOLMES$^{+}(d, m)$ policy more effective, choose the disjoint end-to-end paths (paths with different intermediate aggregate or spine switches) to avoid the scenario that the same local congestion is shared by multiple end-to-end paths. This implementation is applicable for more complex multi-tier Leaf-Spine topologies(two-layer Fat-Trees) or asymmetric DCN topologies.

5.4.4.3 Dealing with the stale information

When implementing the HOLMES^{+} scheduling algorithm at a packet granularity, the transmission latency of a packet is so small that the information refresh rate

in the state table cannot keep up. Correspondingly, the load balancing algorithm has to use stale information to make the scheduling decisions. Mitzenmacher *et al*. [125] pointed out that the delayed information leads to a herd behavior of the scheduling results: data will herd together toward a previously light loaded path for a much longer time than it takes to fulfill the path. Thus, another technical challenge is how to deal with the stale information used in the load balancing algorithms.

To further assess the effects of stale information on the execution results of different scheduling policies, design another experiment that evaluates the load balancing status of the same ToR switch under different state table timers. A scenario is simulated that traffic arrives at a ToR switch with a fixed rate (1 packet per time unit). The ToR switch contains 10 output ports with a fixed data transmission rate (1 packet per 8 time units). Two different timers are deployed to periodically update the information in the ToR state table (state information is updated every 5 or 20 time units respectively). The focus is on evaluating the load balancing status of the same ToR switch under three different flow scheduling approaches in this experiment: the HOLMES$^+(2,1)$ policy, HOLMES$^+(5,1)$ policy, and the deterministic policy ($d = N = 10$ in the definition of the HOLMES$^+(d,m)$ policy).

Figures 5.14, 5.16 and 5.18 show the change trends of the ToR load balancing states under the deterministic policy, such as HOLMES$^+(2,1)$ and HOLMES$^+$-$(5,1)$, when the state table is updated every 5 time units. For comparisons, Figures 5.15, 5.17, and 5.19 show the performance of the three policies with the state table updated every 20 time units. Next, analyze the effects of the stale information aging on different scheduling policies.

Figures 5.14 and 5.15 show the execution results of the deterministic scheduling policy when the refresh periods of the state table are set to 5 or 20 time units respectively. It is found from the two figures that although the deterministic policy provides optimal performance using the perfect run-time state information, it suffers from the effects of the stale information. It is shown in Figure 5.14 that even if the update period is set short (5 time units), the load states of the

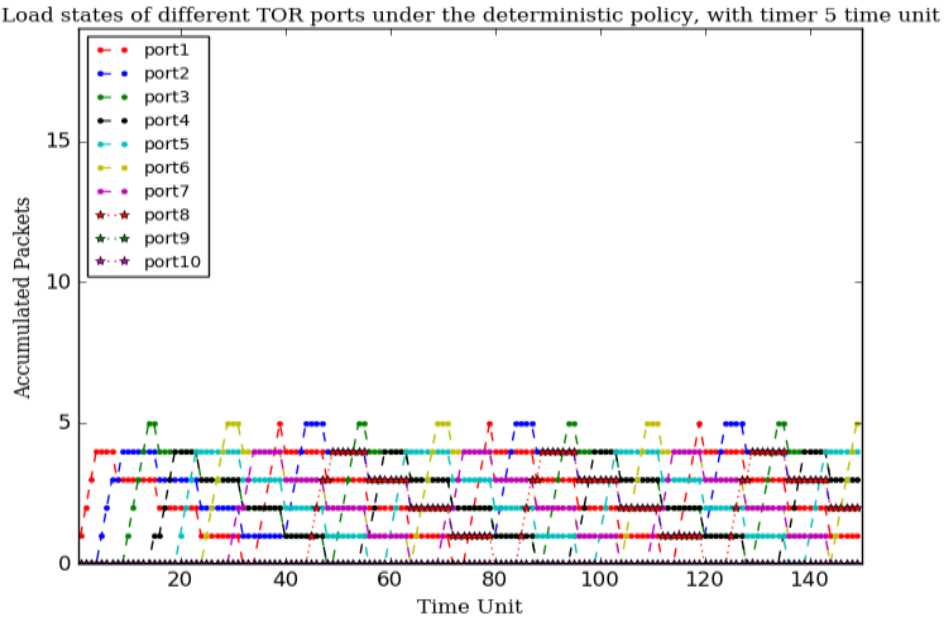


FIGURE 5.14: Load states changing trend of 10 ToR ports under deterministic policy with different state table timer ($T2 = 5$) [12]

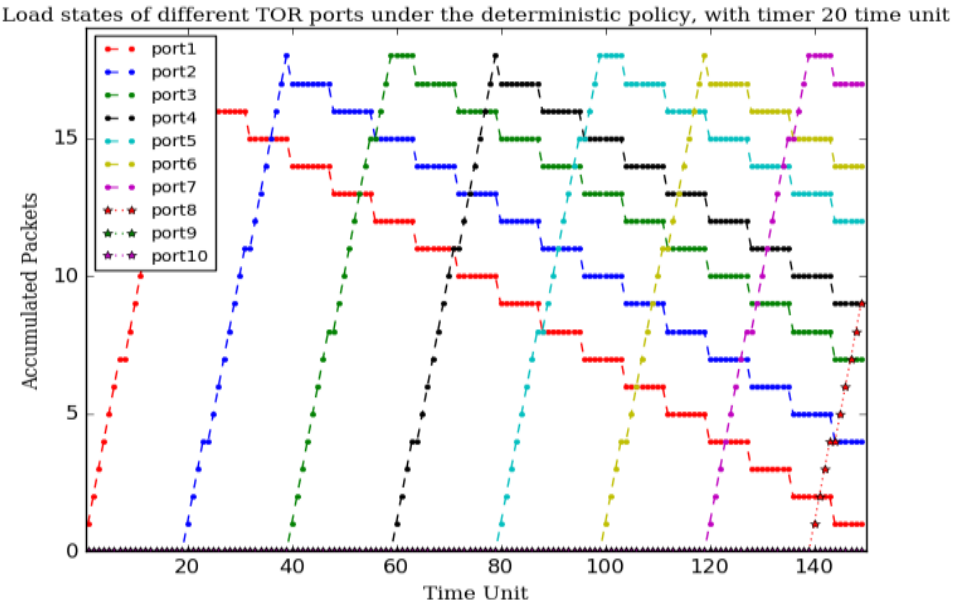


FIGURE 5.15: Load states changing trend of 10 ToR ports under deterministic policy with different state table timer ($T2 = 20$) [12]

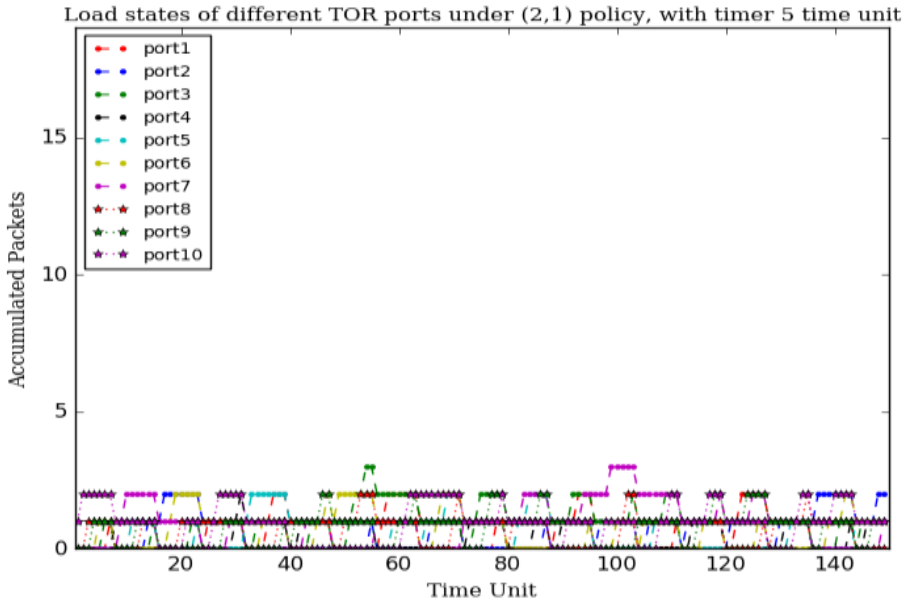


FIGURE 5.16: Load states changing trend of 10 ToR ports under deterministic policy HOLMES^{+}(2, 1) ($T2 = 5$) [12]

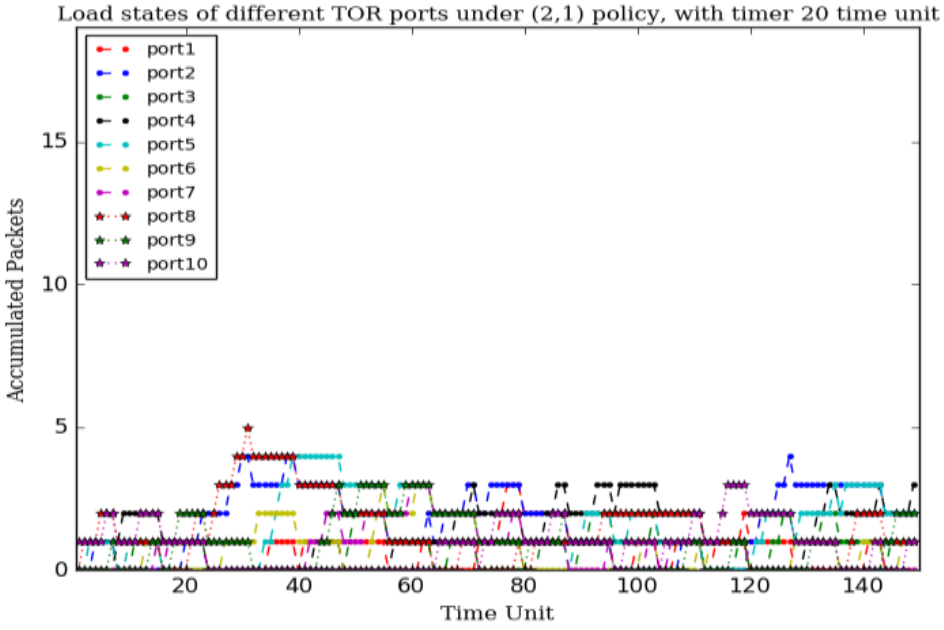


FIGURE 5.17: Load states changing trend of 10 ToR ports under deterministic policy HOLMES^{+}(2, 1) ($T2 = 20$) [12]

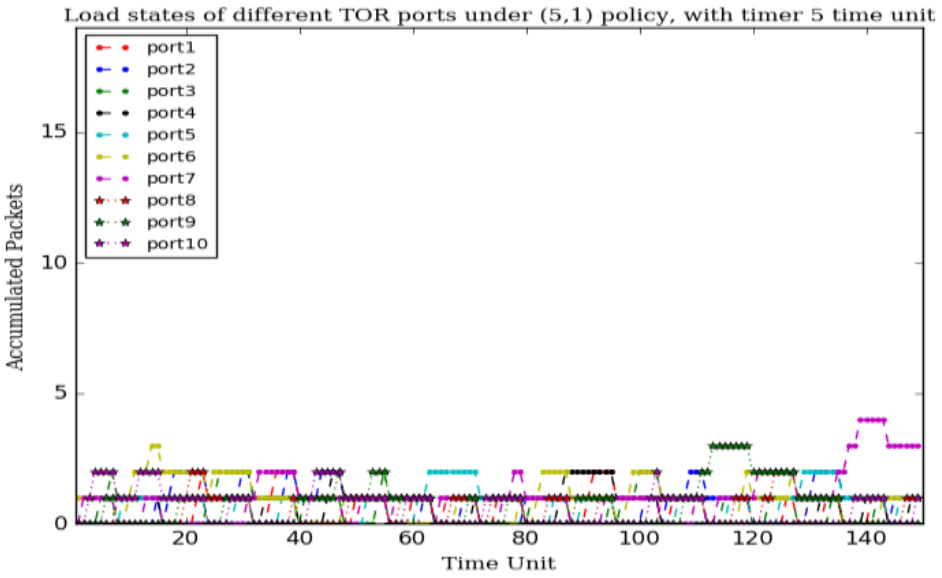


FIGURE 5.18: Load states changing trend of 10 ToR ports under deterministic policy HOLMES^{+}(5, 1) ($T2 = 5$) [12]

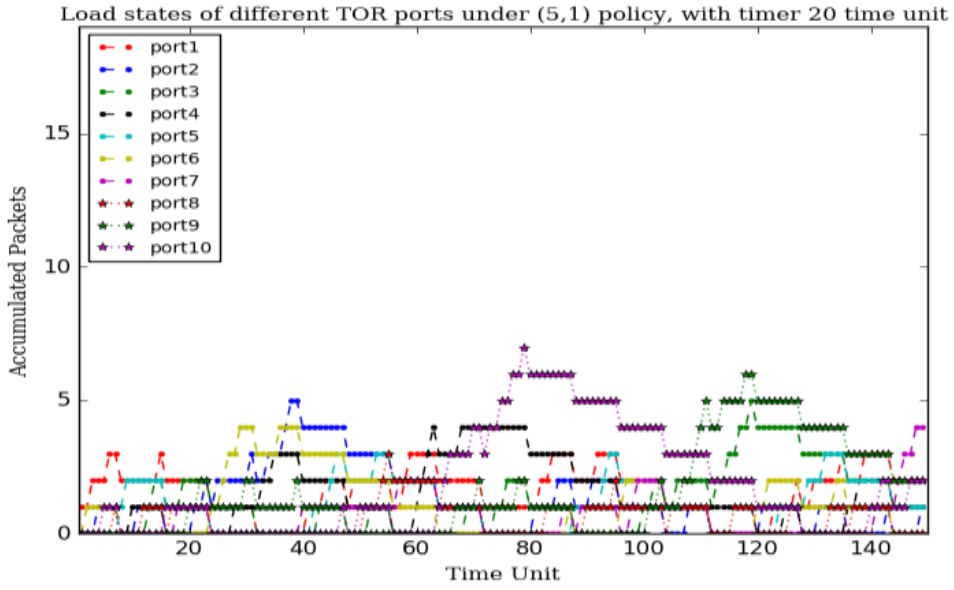


FIGURE 5.19: Load states changing trend of 10 ToR ports under deterministic policy HOLMES^{+}(5, 1) ($T2 = 20$) [12]

ToR ports show obvious herd phenomena: traffic will herd together toward a previously light loaded port first; the next refresh of the state table will put the port high up the load list and the port will become lightly loaded again; then the path will be heavily loaded again after several time periods as the next herd sees that it is lightly loaded. From Figure 5.14 it is seen that each ToR port's load condition shows the obvious periodically changing trend (heavily loaded, lightly loaded, heavily loaded, and so on), and this herd phenomenon is more obvious as the age of the stale state information increases. As shown in Figure 5.15, the amplitude of the fluctuation of each port's load curve is obvious compared with Figure 5.14: the maximum load of a port has increased from 5 packets to more than 15 packets. Compared with the other two approaches (shown in Figures 5.17 and 5.19), the load balancing condition under the deterministic policy becomes the worst as the age of the stale information increases. Thus, the adaptability of the deterministic policy to the stale information is poor.

Theoretically, when deploying the perfect run-time state information, the load balancing condition of the ToR switch under HOLMES$^+(5,1)$ policy is in second place among all the three approaches. Clearly, deploying a larger value of d in the HOLMES$^+(d,m)$ policy increases the probability of choosing a lightly loaded output port. However, comparing Figure 5.18 with Figure 5.16, it is found that the ToR's load balancing condition under the HOLMES$^+(5,1)$ policy (shown in Figure 5.18 does not outperforms the HOLMES$^+(2,1)$ policy (shown in Figure 5.16 when using the short-aged stale information. Moreover, compared with Figures 5.17 and 5.19, note that as the age of the stale information increases, the HOLMES$^+(2,1)$ policy gradually outperforms the HOLMES$^+(5,1)$ policy. The experimental results reflect the issue that, compared with the probability of choosing the optimal path, the adaptability for herd behavior becomes the dominant factor for performance optimization.

Comparing Figure 5.16 with 5.17, it is seen that increasing the age of the stale information does not affect the execution results of the HOLMES$^+(2,1)$ policy: the maximum load of a ToR port is still maintained at less than 5 packets when the information update period increases from 5 time units to 20 time units. Comparing

FIGURE 5.20: Load states changing trend of 10 ToR ports under deterministic policy HOLMES^{+}(5, 1+1) ($T2 = 20$)

Figures 5.15, 5.17, and 5.19, also note that the HOLMES$^{+}(2, 1)$ policy performs best among all the three approaches when using long-aged, stale information. The experimental results indicate that the HOLMES$^{+}(2, 1)$ policy provides optimal adaptability for HOLMES^{+} with the stale information, especially when the age of the information is great. The main reason is that the HOLMES$^{+}(2, 1)$ policy combines the best advantages of both worlds: It deploys real load information to schedule traffic, yet rejects the herd behavior much more strongly than the other two approaches. When implementing the HOLMES^{+} scheduling algorithm based on stale information, the optimal performance approach is to assign $d = 2$.

The HOLMES^{+} scheduling algorithm can use local load information when there is not enough global network status information. The HOLMES$^{+}(d, m+1)$ policy maintain the congestion states of d new randomly chosen paths, m congestion-optimal paths and one local optimal path for the latest time unit for each switch. It enables the network fabric to make load balancing decisions at microsecond time scales based on traffic information as DRILL [94] while missing the global network status information. Compared with Figure 5.19, as shown in 5.20, the HOLMES^{+} scheduling algorithm can still achieve much better performance than HOLMES [12].

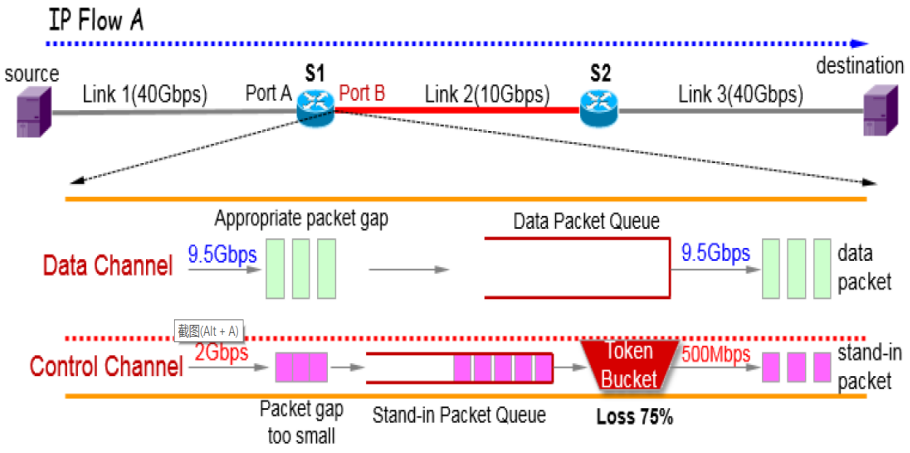


FIGURE 5.21: Overview of SCP dual channel architecture [18]

Besides adjusting the value of modeling factors, *e.g.*, d and m, another applicable solution is to deploy Machine Learning methods to predict the current load state based on stale historical information. Fox [176] used queue length as the metric for load balancing and noting rapid oscillations in queue lengths caused by stale information. To overcome this limitation, they further changed the manager stub to keep a running estimate of the change in queue lengths between successive reports, and showed that these estimations are sufficient to eliminate the oscillations. Dahlin [177] also estimates the job arrival rates and the age of stale information, and proposed corresponding algorithms to balance the traffic load in a distributed system. Similar approaches can also be deployed in HOLMES$^+$ that estimate the current load states of different paths by analyzing successive historical state information. This function will be realized in the HOLMES$^+$ AI module, and will be addressed in future work.

5.5 Combine HOLMES$^+$ with SCP

SCP (Stand-in Control Protocol) is introduced in [18]. As shown in Figure 5.21, SCP configures and logically partitions a physical link into the control and data channels. The data channel is used to carry the actual data packets, while the

control channel carries the stand-in packets. The stand-in packets act as probes to detect the available bandwidth for each flow. It also follows the network partition principle. Before sending actual data packets in the data channel, SCP sends stand-in packets in the control channel to detect the path condition. Based on the preview results of the control channel detections, SCP ensures that the data channel is never congested but still approaches maximum throughput; thus, zero packet loss, full utilization and zero buffer occupancy are simultaneously satisfied in the data channel. In particular, SCP can be used to detect d paths global congestion information for the HOLMES$^{+}(d, m)$ policy. Also, the stochastic power-of-two-choices approach (P2C) of HOLMES^{+} can significantly improve SCP performance, which is an excellent potential topic for future work.

5.6 Summary

Overall, the simulation results validate the modeling results in Section 5.4, and show that the HOLMES^{+} load-balancing algorithm is stable and adaptable in heterogeneous DCNs. The HOLMES^{+} policy uses not only global network status, but also local switch load information.

Chapter 6

Traffic and Network Control Algorithms for Data Centres

One assists nowadays to the rapid technological evolution of Data Centres (DCs), as their new hardware and networking technologies are at the forefront of computer communications. Recently, DCs are being more and more controlled and managed via specialized software platforms that implement Software-Defined Data Centre (SDDC) applications. This new concept brings better agility to DCs, allowing enterprises to maintain business advantages and strength during difficult times. A SDDC allows the virtualization of DC hardware (CPU, storage, and networking edge devices) together with new software concepts such as serverless or microserver-based applications. DC infrastructure management (DCIM) tools can give an organization a comprehensive view of key elements such as power, cooling and computing burden, helping identify inefficiencies and spot potential problems in time. DC packet traffic management is key to designing and implementing the hardware-based DC functionality among these parameters. For this to happen, the traffic has to be structured in classes, more recently in small packets but in great numbers (mice traffic) and large packets in limited numbers (elephants traffic).

Chapter 6 will introduce a new formalism for traffic control based on traffic classification by taking into account the number and size of packets, splitting the DC

traffic into two major traffic classes, *i.e.*, low and high traffic, called mouse and elephant traffic, respectively. Therefore, it is enough to measure the packets regarding their size and then implement a ratio control method to separate the DC network usage. These concepts will be used in the design of DC hardware such as optical switches in general, and for the HyperOXN architecture (proposed in Chapter 3), new optical network elements in particular. This chapter will present the results obtained through simulations using a combination of OMNET++ [178] and Wireshark [179].

6.1 Introduction

Data center network topology plays a significant role in the "last mile" of traffic, its transmission through the Internet and its Quality of Service (QoS) as measured by evaluating congestion conditions. The traffic always originates from a sender's computing device and ends up in the receiver's server, encountering on its path above all the hardware and software libraries as specified in the RFCs of the Open System Interconnect (OSI) Model. In the following, it is assumed that a stream of Layer 6 and 5 is transmitted through to the receiver according to Transmission Control Protocol (TCP) prescriptions. The traffic is only supervised by the Application and Session Layers of the sender's computer, starting with the TCP Layer of that computer. As such, the traffic units' (segments at the TCP Layer, packets at the IP layer, and frames at the last layer of transmission) travel behaves dynamically and affects the performance of all the network elements encountered on its path, thus determining the level of failure resiliency, ease of incremental expansion, communication bandwidth, and latency. To improve traffic reception at the receiver point in the network, robust network topologies with low latencies (tens of microseconds) and high bandwidth across servers (up to 40 Gbps) have been proposed and implemented. Some topologies scaling into large networks have been designed and implemented by connecting many inexpensive switches to achieve large aggregate capacities. Lately, optical switches have been proposed, connected and tested for building the last leg of DC networks. The use of optical

switches allows DC ITs to reconfigure a DC network's topology dynamically as needed.

6.1.1 Classification of TCP Traffic

In the following, the dynamics of the traffic transmission will be considered in two different apostasies: i) when the traffic contains only short packets (*i.e.*, mouse type of traffic - *e.g.*, small packets like emails), and ii) when the traffic contains only very large packets (*i.e.*, the elephant traffic containing large files of large streams- *e.g.*, video streams, etc.). The TCP/IP Reference Model is considered in establishing the traffic model.

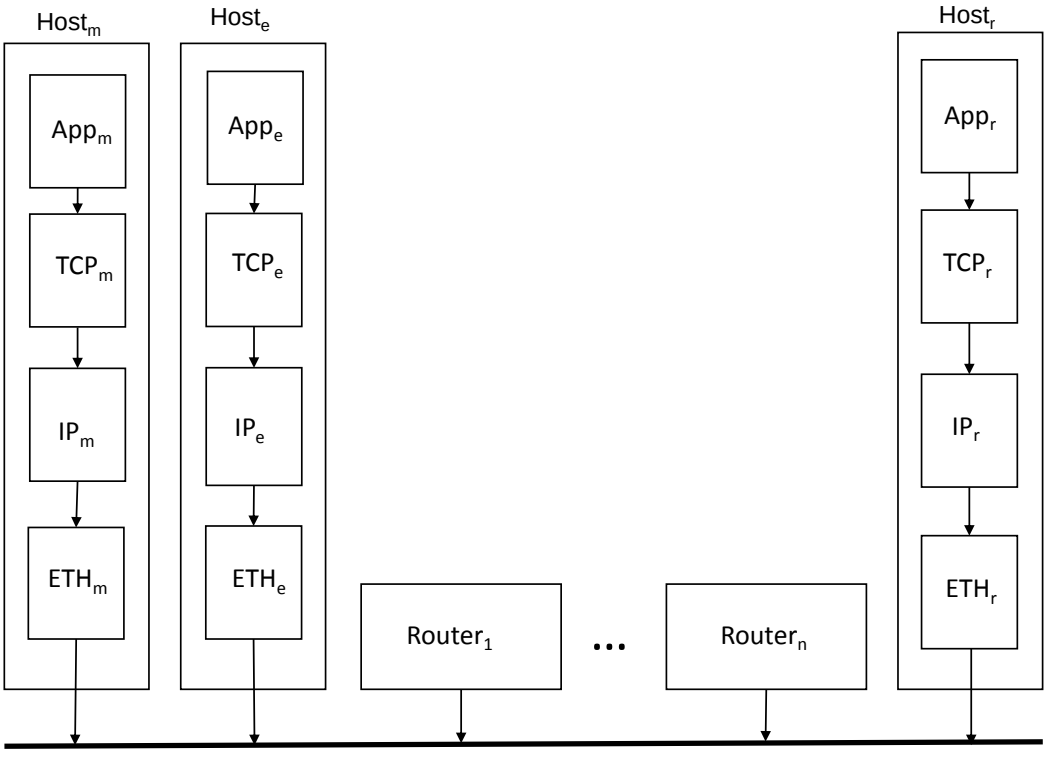


FIGURE 6.1: Traffic of Mice and Elephants Shapes

Nowadays, Data Centers have to manage a complex environment of co-located tenants running different heterogeneous virtual applications which generate various numbers of TCP windows with variable fragment sizes. The restriction on the TCP size of the fragment number is variably dependent on the number of fragments, the

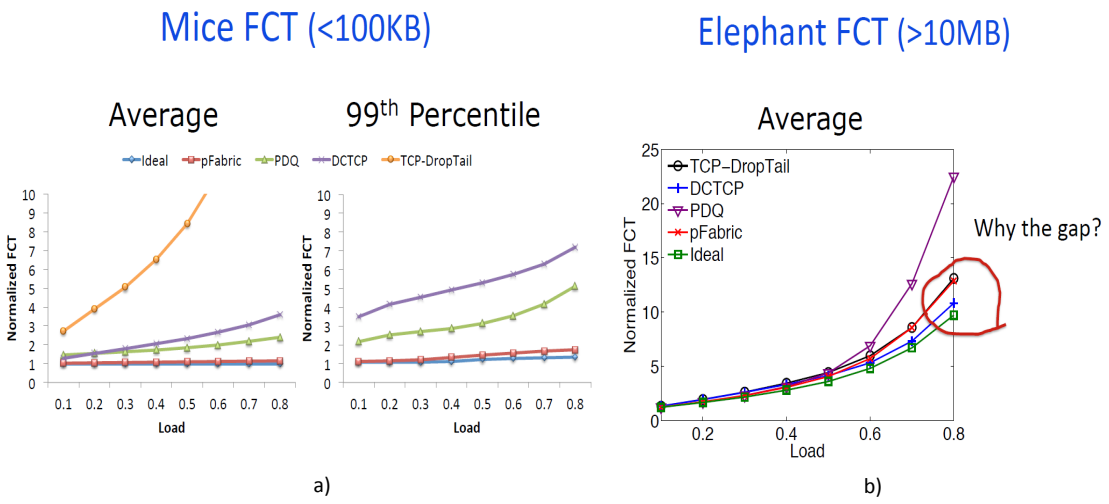


FIGURE 6.2: Mice and Elephants Traffic Sizes: a) Mice; b) Elephants

IP queue length, the algorithm used for solving the packet queue, and the size of the receiver's buffer. The literature and practical measurements have shown that Internet traffic can be classified into two categories based on the number of *Bytes* it contains. This leads to classifications defined by the size of the packets, such as packets under $100KB$, called mice packets while packets whose size is larger than $10MB$ are categorized as elephants packets.

It has been proven statistically that according to the mouse-elephant phenomenon, 80% of the traffic flows belong to the mouse category, contributing to 20% of the total traffic, while in contrast, the elephant flows are only responsible for 20% of the numbers of flows but comprise up to 80% of the Internet's total traffic. The above traffic dichotomy justifies the strong heavy-tail behaviour, leading to increases in per-flow bandwidth latency, which affects the efficiency of the traffic control and imposes reconsideration of the DC network architecture implementation. Internet TV, the use of video-conference tools such as Skype, Webex, Teams, and Zoom, is now a reality and no longer a research subject. The mice traffic generated by Tweets, Web queries, HTTP requests, etc., for which users expect very short response times, is hindered by elephant flows. These large flows are usually created by downloads processes that run in the background (say, a kernel image or a movie), triggering high response times up to orders of magnitude.

6.1.2 On Mice and Elephants Packets

Obviously, the mice traffic always is hindered because mice traffic always takes place in the *Slow-Start*(SS) phase, never reaching the congestion avoidance case. In contrast, the packet loss in the case of small flows very often leads to a *time-out* due to the use of a small congestion window parameter (*cwmd*). Under these conditions, the TCP becomes inefficient and prone to instability, regardless of a queue-solving algorithm's queuing size. It is therefore essential to find appropriate mechanisms or algorithms to resolve the internal conflict between mouse and elephants traffic. At the same time, as the Internet improves in bandwidth and speed, the number of TCP segments transmitted in either mouse or elephant traffic conditions in very high-bandwidth optical links requires the formulation of more adequate transmission parameters. Similarly, the 5G and soon 6G wireless technologies will also improve the wireless traffic conditions if the mice- and elephants-dichotomy is considered in the wireless transmission protocols.

6.1.3 On TCP Parameters

A possible solution to the above unbalanced traffic issue can be achieved if size-based scheduling of the traffic flows can be implemented. This proposed mechanism exploits the TCP behaviour during the SS phase of the TCP protocol in order to detect and de-prioritize large flows, thereby giving preference to small or large flows depending on the size and occupancy status of the ingress port buffer. One method is to measure the burst of packets contained in a congestion window *cwnd* that arrives at a bottleneck router, which is called a spike. The size of the spike can indicate in this way that a large flow is present at a specific port, and the port can then apply a dropping mechanism, thus de-prioritizing the elephant flows, and this action can be done instantly. This mechanism is extremely important in various networks which require flow monitoring, as in the case of emergency networks.

Conflict between mouse and elephants flows can happen in various DCs, especially in those dedicated to health services systems that provide medical services to very large populations. In such situations, it is important to know when the dynamic of the mice flows are affected by the dynamics of the elephant flows and vice-versa. Obviously, both the dynamic and steady-states of both types of flows are important to detect, measure, and control [180].

The transient behaviour of either mice or elephants is due to the fact that packets are formed and sent by the sender's communication stack by the combined algorithmic efforts of the Transmission Control Protocol (TCP) and the Internet Protocol.

Therefore, the dynamic behaviour of the mice or elephants flows is described by the parameter of the TCP that measures the number of TCP PDUs sent in a single round of transmission, called the Window size W. This is the average TCP window size measured in segments or sometimes in packets, which is the same (1,500- Byte segments), can fill a maximum of 65,535-Byte window, or equivalently 45 segments if a segment is the Maximum Transmission Unit (MTU) of the TCP. The Maximum Segment Size (MSS) of the TCP is 526 Bytes. The window can carry one or more segments to the receiver's buffer, irrespectively of whether the traffic flows comprise mice or elephants.

6.2 A Control View of TCP Flows

When a TCP connection is set, the sender will enter an SS process in which the sender will exponentially increase the segments sent until the sender and receiver both agree on a value (measured in segments) which will be used in both sending (sender's amount of segments) and receiving the segments. This value is called $ssthresh$, the performance of SS, and the TCP process also depends on it, as the sending process is very sensitive to its initial value. This parameter is generated in the same way for both sides of the TCP process [181]. Once the values for the window $W(t)$, $cwnd$, and $ssthresh$ are determined, the TCP process continues

transmitting segments for the entire amount of Bytes. Therefore , the segments are all in the receiver's buffer if either the sender or the receiver, do not stop participating in this process.

Although the sender's computer transmits Bytes grouped in segments in order to be compatible with the language used in the TCP RFPs, it will be considered that windows are sent from the sender to the receiver by cooperating with the TCP and IP protocols. The segments sent by the TCP are first embedded in a Datagram of the IP, and the segments are called packets with a precise IP address. The MTU of a packet can be of a maximum size of $65,535$ Bytes; however, the TCP segment cannot exceed 1500 Bytes. The packet is then sent to the router next to the host port (the gateway). The router next to the host computer will decide the address of the next router, which in turn will put the next router address in its routing algorithm (and table) and so on, until the original segment dressed up now as a packet, arrives at the receiver's host. After receiving the total number of Bytes, the receiver's host will send back to the sender's host an acknowledgement ACK letting the sender's computer know that the packet was buffered in the receiver's computer. In contrast, if the window initially sent by the sender arrives at the receiver host, but the receiver host cannot put the packet in its host buffer, then the TCP server at the receiver's side sends a $NACK$ that indicates the segment was not buffered. The sender's computer then has to re-transmit with a smaller window size than before. All this process takes place in time, and the duration of the transmission period is denoted by $R_{TT}(t)$ measured in seconds and called the round trip time $R_{TT}(t) = T_{_Q} + \frac{Q}{C}$ where $T_{_Q}$ is the propagation delay of the packet, which is also dependent of the size of the element of the queue Q while C is the total capacity of the link to the network node, measured in $\frac{packets}{seconds}$.

$$W(t) = k_{\mathcal{S}} S(t) \tag{6.1}$$

where $k_{\mathcal{S}}$ is an integer measuring the number of segments contained in a window, i.e., $k_{\mathcal{S}} \in \mathbb{N}$.

If the receiver's buffer cannot accept a packet, a loss happens and is detected by a window $W(t) = 1$ packet, the size of which is called the Maximum Size Segment (MSS). If there was a correct transmission of the window, its $W(t)$ size is increased with 2 packets. This part of the TCP protocol is called *Slow-Start* (SS).

6.3 An SDE Model for TCP Dynamics

The modeling of these processes can be done by either considering that the TCP window is discretely incrementing the number of segments transmitted, as is done in some modeling techniques, or by modeling a $Window$ as a recipient in which the segments are assembled by the TCP and then shipped by the corresponding IP time to the first edge router to which the host is connected. The router then receives the window as an IP packet and places it in an ingress queue solved by the router in a First Come First Served (FCFS) policy. For modeling the TCP window behaviour $W(t)$ is considered as a stochastic variable describing a Poisson counter having a parameter $\Lambda = \nu\lambda$. Traditional modeling of TCP traffic transmission with the probability of packet losses has been done from a source-centric point of view. Those models assume that packets go out on the network with some loss probability P, which may be constant or may depend upon factors like current window size etc. In this present model, the network is the source of losses (congestion), and sources receive these signals (loss indications) as a Poisson Counter Driven Process (PCDSDE) with some rate (λ) which will lead to a Stochastic Differential Equation (SDE) [182] modeling the window size of TCP as a fluid, having continuous increments as opposed to discrete ones. This model lends itself to a formulation of the window size behaviour described by the stochastic process. It combines a countably dimensional Wiener process and Poisson random measure. Similarly, the queue length will be described by the same mathematical description.

Consider that $X(t)$ is a state variable of a stochastic process described by a combination of a Wiener process and a PCDSDE power law (WPCDSDE). This translates in an SDE, mathematically described by the state equation given below as analyzed in [183], where a Wiener process $dw(t)$ is mixed with the Poisson process $N(dy, dt)$, while N and W are independent stochastic processes.

$$\begin{cases} dX(t) = a(t, X(t))dt + b(t, X(t))dW(t) + \\ \int_{\mathcal{E}} c(t, X(t_-))N(dy, dt), \ t \in [0, T] \quad \text{and} \quad T > 0 \\ X(0) = \delta \end{cases} \tag{6.2}$$

provided that $\mathcal{E}_0 = \mathbb{R}_0^d$ with $d \in \mathbb{N}$. The Wiener process $W = [W_1, W_2, ...]^T$ is a countably dimensional Wiener process on a complete probability space $(\Omega, \Sigma, \mathbb{P})$, i.e., an infinite sequence of independent scalar Wiener processes defined on the same probability space. The Poisson process is assumed to be an intensity measure $v(dy)dt$, where $v(dy)$ is a finite Lévy measure on $(\mathcal{E}, \mathcal{B}(\mathcal{E}))$. More simply, in (6.2), N is a Poisson counter process described in simple terms as:

$$dN = \begin{cases} 1 & \text{at Poisson arrival} \\ 0 & \text{elsewhere} \end{cases} \tag{6.3}$$

or equivalently:

$$\mathbb{E}N(t) = \lambda t \tag{6.4}$$

For the equation to exist, certain regularity conditions must be imposed on the system coefficients a, b and c. In the following, only the finite-dimensional case will be considered.

Similarly, a TCP expected window $W(t)$, measured in segments, i.e.,

$$W(t) = k_S S(t) \tag{6.5}$$

where $k_S \in \mathcal{R}$ is a constant which represents dynamically the number of segments per window and $W(t)$ is a Poisson counting process: $\{\mathbf{N}(t), t \leq 0\}$ representing

the collection of the total number of events that have occurred up to and including the time t defined by the following properties:

- $N(0) = 0$ where N is a counting process denoted $\{N(t), t\}$ with rate λ,
- has independent increments; and
- the number of events (or points) in any interval of length t is a Poisson random variable with parameter (or mean) λt.

This process uses the *ssthresh* and *cwnd* parameters, which are set for the whole network by the network administrator. For controlling the *cwnd* parameter, the TCP implements the additive increase-multiplicative decrease (AIMD) scheme in order to achieve a fair division of the available bandwidth in a network among competing sources (users). It is a window-based method in which at any time, the window size measured by the number of data packets is allowed in the network. The comparison between *cwnd* and *ssthresh* will define the correct variation of the window size that either triggers a linear increase in the number of segments in the window by adding one more segment in the *cwnd* after each ACK received, or reduces the window by half if the $NACK$ is received. In a more formal way:

$$\text{if}\, ssthresh - cwnd < 0 \text{ then } \; cwnd_{new} = \frac{1}{2} cwnd \tag{6.6}$$

$$\text{if } \; ssthresh - cwnd > 0 \; \text{ then } cwnd = cwnd + \lceil \frac{cwnd}{cwnd_{Max}} \rceil \tag{6.7}$$

These two lines of the TCP window dynamics can be written in a mathematical expression as:

$$dW = \begin{cases} -\frac{1}{2} W & \text{if } ssthresh - W < 0 \\ W + \lceil \frac{W}{W_{Max}} \rceil & \text{if } ssthresh - W > 0 \end{cases} \tag{6.8}$$

The process described above takes place in the transmission of the sender's segment/packets, both in the case of mouse and/or elephant flows. As any flow on

the Internet originated by the OSI upper-layer protocols, eventually, the Protocol Data Unit (PDU) of the TCP will be taken according to the TCP algorithms, formed as a segment, and wrapped in a Datagram by the IP in order to be shipped on the Internet as a packet. Thus, segments and packets, either mice or elephants, are generated by the sender's computer, while the receiver can either receive the packets sent by the sender' window, or reject them due to the receiver's buffer being full. The receiver's computer acknowledges the sender of its acceptance or rejection of the last segment received. As the segments are packed in TCP windows, in a number from 1 to 45 the window will suffer the variations caused by the dynamic capacity of the receiver's buffer and continuously comparing at every step of the transmission $cwnd$ with $ssthresh$. As the increase is done in increments of one segment at a time, and randomly, the window size is also a random process which is mathematically connected to the segment through the relationship given by (6.1):

$$k_S = \frac{W(t)}{S(t)} \tag{6.9}$$

However, the Window variable is described by a more general Stochastic Differential Equation (SDE), which generates discontinuous processes as solutions due to the inclusion of Poisson processes in the SDE equation.

6.4 A Control Solution for TCP Mice and Elephants Flows

From the dynamic point of view, the window size $W(t)$ increases with everyACK received from the receiver's host, making the $W(t)$ evolution similar to fluid filling a bucket with a valve that opens when $ssthresh$ is overpassed. The valve empties half of the bucket. Similarly, the window size of TCP as a fluid, has continuous increments as opposed to discrete ones. This model lends itself to a formulation of the window size behaviour as a Poisson Counter Driven Stochastic Differential Equation (PCSDE).

There are a few SDE versions of equations describing the queue variations for the one-server case. Below the proposed SDE described in [184] is shown.

When no loss occurs the window size increases linearly at rate $\frac{1}{R(t)}$ so the window size is increased by **approximately** $\frac{1}{W(t-R(t))}$ each time an acknowledgment returns to the source.

However, when a packet is lost the window is cut in half. Setting

$$u_1(t) = u_{-1}(t) - u_{-1}(t - \frac{1}{R_{TT}}) \tag{6.10}$$

where $u_{-1}(t)$ is the unit step function, while $u_1(t)$ is the sum of $u_{-1}(t)$, its value is shifted by $\frac{1}{R_{TT}}$.

Hence, the evolution of the window size is described by:

$$dW(t) = \frac{u_1(t)}{R_{TT}(t)}dt - \frac{W(t)}{2}dN_{TD} + (1 - W(t))dN_{TO} \tag{6.11}$$

where dN_{TD} is the Poisson counter for the Triple Duplicate, while dN_{TO} is the Poisson counter for timeout processes happening during segment transmission, while $W_0 = W(0)$, and $N(t)$ is a Poisson counting process.

However, the fluid queue model describes arrivals as continuous rather than discrete processes. In models like M/M/1 and M/G/1, queues in which $Q(t)$, $U(t)$, and $V(t)$ are defined by flow functions. In other words, functions such as $Q : \mathcal{R}(\mathbb{R}) \to \mathbb{R}$, which is the length of the queue, $U : \mathcal{R}(\mathbb{R}) \to \mathbb{R}$, which denotes the rate at which the segments/packets in the system are delivered to the queue buffer, and $V : \mathcal{R}(\mathbb{R}) \to \mathbb{R}$ which represents the rate of the segments are processed by the queue processor implementing, in general, the FIFO algorithm and delivered to the receiver, while the entity defined by $\frac{dQ(t)}{dt}$ is the rate or the speed of the variation of the queue length. There is therefore a need to use homogeneous entities in the model equations.

This is the role played by the two coefficients $\alpha(t)$ and $\beta(t)$ representing the relationship between the speed of the variation of the queue length and the requests

rates and the output rate. In the following, these two coefficients will be related to the ratio of the in-requests and solved requests. Attached to the above equation the queue process is described by the following equation:

$$dQ(t) = \beta(t)U(t)dZ(t) - \alpha(t)V(t)dt \tag{6.12}$$

where $U(t)$ describes the rate of the packets arriving in the buffer of the **router**, and $V(t)$ expresses the rate at which the packets are processed and routed by the receiving router to the next receiving router or the destination host. In the same (6.12), Z_t is a standard Poisson Counter process that describes the additive stochastic character of the packet arrival rate. A TCP complete model encompasses the process of transforming the document of Layer 7 into segments, packing the segments into windows, sending the windows to the next edge router, and routing the packets to their destination. In contrast, the receiver host checks whether the entire window arrived and was placed correctly in the receiver's buffer, and eventually acknowledges the sender that the packet(s) had enough memory space by sending the sender an ACK, or if there is not enough buffer space, a $NACK$.

In order to model the above process, let us list the symbols that will be linked in a mathematical construct. Thus:

- $W \doteq$ expected TCP window size, measured in segments (or packets for the case of segments and packets being of the same size with respect to the MSS)
- $Q \doteq$ expected queue length measured in packets
- $R \doteq$ the round trip time size, measured in seconds $R = \frac{Q}{C} + Tp$
- $p \doteq$ probability of dropped (marked) packet
- $N_{TCP_{Ses}} \doteq$ the number of TCP sessions in the process of sending data from layer 7 to the receiver.

There are therefore two different communication processes:

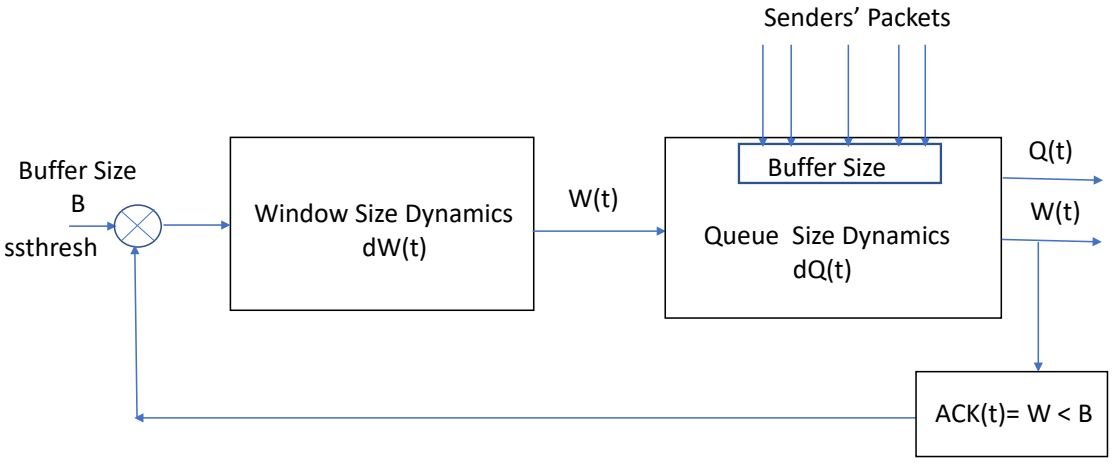


FIGURE 6.3: Control Loop for Mice and Elephants

- the Window communication process which is described by the dynamics of W the state variable, and
- the routing process, which describes the dynamics of Q.

Figure 6.3 depicts the two processes connected in series: i.e., $W(t)$ is described by a first-order SDE whose inputs are the static parameters set at the moment of linking the hosts to the network, and trying to keep the system under control, i.e., enough buffering space and continue to keep the size of the window (this time is *ssthresh*) and the result of the window dynamics processes is transmitted as input to the IP router to transmit it further up to the reception point where the packet (s) is placed in the receiver buffer.

There are therefore the two equations developed above:

$$dW(t) = \frac{u_1(t)}{R_{TT}(t)}dt - \frac{W(t)}{2}dN_{TD} + (1 - W(t))dN_{TO} \tag{6.13}$$

and:

$$dQ(t) = \beta(t)U(t)dZ(t) - \alpha(t)V(t)dt \tag{6.14}$$

which corresponds to the transmission of the windows transformed into packets to the next router to route the traffic received from the sender.

This is depicted in Figure 5.1, in which the flow of the traffic is separated by the decision element, which allocates the resources first to the mouse traffic and then to the elephant traffic. Thus there is a need to model the two flows separately:

$$dW_m(t) = \frac{u_1(t)}{R_{TT}(t)}dt - \frac{W_m(t)}{2}dN_{TD} + (1 - W_m(t))dN_{TO} \tag{6.15}$$

$$dQ_m(t) = \beta_m(t)W_m(t)dZ(t) - \alpha_m(t)V_m(t)dt \tag{6.16}$$

and similarly for the elephants flows.

$$dW_e(t) = \frac{u_1(t)}{R_{TT}(t)}dt - \frac{W_e(t)}{2}dN_{TD} + (1 - W_e(t))dN_{TO} \tag{6.17}$$

$$dQ_e(t) = \beta_e(t)W_e(t)dZ(t) - \alpha_e(t)V_e(t)dt \tag{6.18}$$

6.5 An SDE Solution for TCP Mouse and Elephant Flows

Stochastic processes are always generated by a combination of a desired process which contains one or more component whose event generator has an unpredictable evolution and is defined as a noise process fact expressed by:

$$\alpha(t) = f(t) + h(t)\zeta(t) \tag{6.19}$$

where $\zeta(t)$ denotes a noisy process, or more precisely a White Noise process. These facts lead to difficulties in applying the **Integral Calculus** rules in the description of $\alpha(t)$ and the general SDE equation (6.2). Solving SDEs, for those solvable, is usually done by taking the expectation $\mathbb{E}$ on the differential stochastic variables on both sides of the equation. However, Poisson processes are defined and solved in the discrete domain, i.e., $N_i(t) \in \mathbb{N}$. This causes the PCDSDE solution to be found in integers.

6.6 Poisson Driven SDE

The SDE approach in the presence of Poisson noise is based on taking the statistic expectation on both sides of the SDE.

$$\mathbb{E}[dW(t)] = \mathbb{E}[u_1(t)]\frac{1}{R_{TT}}dt - \frac{\mathbb{E}[W(t)]}{2}\mathbb{E}[dN_{TD}] + [(1 - \mathbb{E}[W(t)])\mathbb{E}[dN_{TO}] \quad (6.20)$$

while calculating the expectation on both sides

$$\mathbb{E}[dQ(t)] = \mathbb{E}[\beta(t)][\mathbb{E}[W(t)]\mathbb{E}[dZ(t)] - \mathbb{E}[\alpha(t)]\mathbb{E}[V(t)]dt \quad (6.21)$$

As $\mathbb{E}[W(t)]$ is differentiable (6.20) can be written as:

$$d\mathbb{E}[W(t)] = \frac{\mathbb{E}[u_1(t)]}{R_{TT}(t)}dt - \frac{\mathbb{E}[W(t)]}{2}\lambda_{TD}dt + (1 - \mathbb{E}[W(t)])\lambda_{TO}dt \quad (6.22)$$

$$\mathbb{E}[dQ(t)] = \mathbb{E}[\beta(t)]\mathbb{E}[W(t)]\mathbb{E}[dZ(t)] - \mathbb{E}[\alpha(t)]\mathbb{E}[V(t)]dt \quad (6.23)$$

As $\mathbb{E}[Q(t)]$ is differentiable, (6.23) can be written as:

$$\frac{d\mathbb{E}[W(t)]}{dt} = \mathbb{E}[u_1(t)]\frac{1}{R_{TT}} - \frac{\mathbb{E}[W(t)]}{2}\lambda_{TD} + (1 - \mathbb{E}[W(t)])\lambda_{TO} \quad (6.24)$$

and similarly:

$$\frac{\mathbb{E}[Q(t)]}{dt} = \mathbb{E}[\beta(t)]\mathbb{E}[W(t)]\lambda_Z - \mathbb{E}[\alpha(t)]\mathbb{E}[V(t)] \quad (6.25)$$

According to [185], the FIFO processing of the queue $Q(t)$ can be described by the following equation:

$$V(t) = \frac{\mu Q(t)}{a+Q(t)} \quad (6.26)$$

which is a positive bounded function of the $Q(t)$ and where $V(t)$ denotes the rate at which the packets are released to the network. Obviously the elephants and mice packets will be released with different rates $V_e(t)$ and $V_m(t)$.

This leads to:

$$\frac{\mathbb{E}[Q(t)]}{dt} = \mathbb{E}[\beta(t)]\mathbb{E}[W(t)]\lambda_Z - \mathbb{E}[\alpha(t)]\mathbb{E}[\frac{Q(t)}{a+Q(t)}] \tag{6.27}$$

where a is a constant and it is chosen usually $a = 1$, i.e.,

$$\frac{d\mathbb{E}[Q(t)]}{dt} = \mathbb{E}[\beta(t)]\mathbb{E}[W(t)]\lambda_Z - \mathbb{E}[\alpha(t)]\mathbb{E}[\frac{Q(t)}{1+Q(t)}] \tag{6.28}$$

The SDE system of equations becomes:

- The window equation

$$\frac{d\mathbb{E}[W(t)]}{dt} = \mathbb{E}[u_1(t)]\frac{1}{R_{TT}} - \frac{\mathbb{E}[W(t)]}{2}\lambda_{TD} + (1 - \mathbb{E}[W(t)])\lambda_{TO} \tag{6.29}$$

- The queue equation

$$\frac{d\mathbb{E}[Q(t)]}{dt} = \mathbb{E}[\beta(t)]\mathbb{E}[W(t)]Z_T - \mathbb{E}[\alpha(t)]\mathbb{E}[\frac{Q(t)}{1+Q(t)}] \tag{6.30}$$

The latter equation is a nonlinear SDE due to the term $\frac{Q(t)}{1+Q(t)}$. This nonlinear equation can be solved by applying the Taylor linearization/approximation formula for Ito stochastic processes. The parameters α and β are homogenizing parameters transforming windows to packets and packets to windows. Thus:

$$W(t) = K_{W-P}Q(t) \tag{6.31}$$

$$\begin{aligned}\mathbb{E}[\frac{dQ(t)}{dt}] = \mathbb{E}[\beta(t)]\mathbb{E}[W(t)]\mathbb{E}[Z_t] - (\mathbb{E}[\alpha(t)]\frac{Q_0}{1+Q_0}\lambda_t \\ + \mathbb{E}[\alpha(t)]\frac{1}{(1+Q_0)^2}(\mathbb{E}[Q(t)] - Q_0)\lambda_t)\end{aligned} \tag{6.32}$$

Taking into account that the two equations describe two processes $W(t)$ and $Q(t)$ connected in series, the output of the equation for $W(t)$ is the input for the $Q(t)$ equation. The above is obtained from the block diagram in Figure 6.4. Let us operate simple transforms:

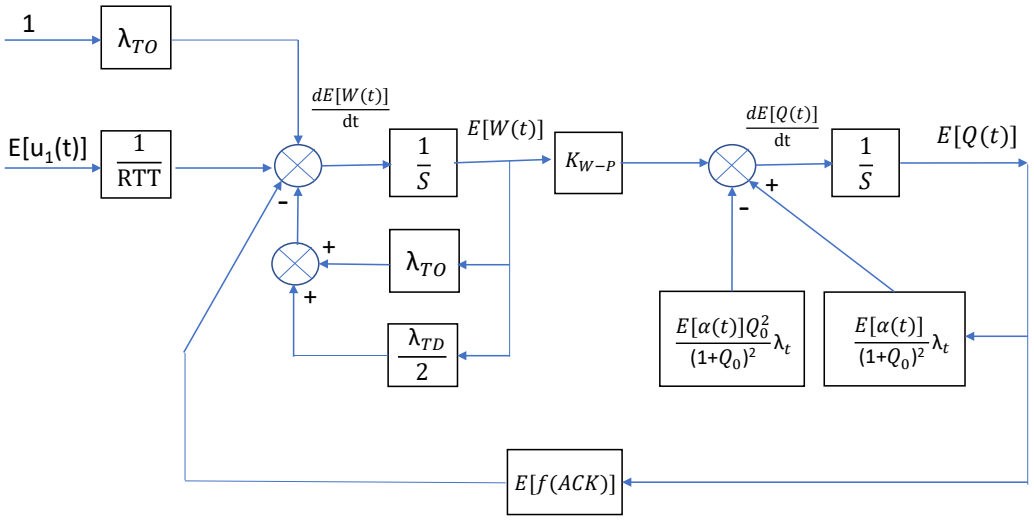


FIGURE 6.4: Block Diagram for Windows and Queues

$$\frac{d\mathbb{E}[W(t)]}{dt} = \mathbb{E}[u_1(t)]\frac{1}{R_{TT}} - \frac{\mathbb{E}[W(t)]}{2}\lambda_{TD} + (1 - \mathbb{E}[W(t)])\lambda_{TO} \tag{6.33}$$

$$\frac{d\mathbb{E}[Q(t)]}{dt} = \mathbb{E}[\beta(t)]\mathbb{E}[W(t)]\lambda_{Z_t} - \frac{\mathbb{E}[\alpha(t)]}{(1+Q_0)^2}Q_0^2\lambda_t + \frac{\mathbb{E}[\alpha(t)]}{(1+Q_0)^2}\mathbb{E}[Q(t)]\lambda_t \tag{6.34}$$

From (6.33) and (6.34), one obtains:

$$\begin{aligned}\frac{d^2\mathbb{E}[Q(t)]}{dt^2} + (\lambda_{TO} + \frac{\lambda_{TD}}{2} + \frac{\mathbb{E}[\alpha(t)]}{(1+Q_0)^2}\lambda_t)\frac{d\mathbb{E}[Q(t)]}{dt} + \mathbb{E}[f(ACK)]\mathbb{E}[Q(t)] \\ = \mathbb{E}[u_1(t)]\frac{1}{R_{TT}} + \mathbb{E}[\beta(t)]\mathbb{E}[W(t)]\lambda_{Z_t}\end{aligned} \tag{6.35}$$

Equation (6.35) is second-order with variable coefficients that can be solved by applying the "Variations of Parameters" method, i.e., $\mathbb{E}[Q_1(t)]$ and $\mathbb{E}[Q_2(t)]$ are a fundamental set of solutions, and $\eta_1(t)$ and $\eta_2(t)$ are a pair of functions which satisfy the following:

$$\mathbb{E}[Q_P(t)] = \eta_1(t)\mathbb{E}[Q_1(t)] + \eta_2(t)\mathbb{E}[Q_2(t)] \tag{6.36}$$

where $\eta_1(t)$ and $\eta_2(t)$ satisfy:

$$\eta_1'(t)\mathbb{E}[Q_1(t)] + \eta_2'(t)\mathbb{E}[Q_2(t)] = 0 \tag{6.37}$$

and:

$$\eta_1'(t)\mathbb{E}[Q_1'(t)] + \eta_2'(t)\mathbb{E}[Q_2'(t)] = \mathbb{E}[g(t)] \tag{6.38}$$

where $\mathbb{E}[g(t)]$ is a function that is the external term of the nonhomogeneous equation, while $\mathbb{E}[Q_P(t)]$ is the particular solution of the nonhomogeneous equation.

It can be shown that:

$$\eta_1(t) = -\int \frac{\mathbb{E}[Q_2(t)]g(t)}{W(\mathbb{E}[Q_1(t)], \mathbb{E}[Q_2(t)])} \tag{6.39}$$

and:

$$\eta_2(t) = -\int \frac{\mathbb{E}[Q_1(t)]g(t)}{W(\mathbb{E}[Q_1(t)], \mathbb{E}[Q_2(t)])} \tag{6.40}$$

where $W(\mathbb{E}[Q_1(t)], \mathbb{E}[Q_2(t))$ is the Wronskian of the system of equations satisfied by $\mathbb{E}[Q_1(t)], \mathbb{E}[Q_2(t)$ and $\mathbb{E}[Q_1'(t)], \mathbb{E}[Q_2'(t)$, i.e.,

$$W(\mathbb{E}[Q_1(t)], \mathbb{E}[Q_2(t)) = \det \begin{bmatrix} \mathbb{E}[Q_1(t)] & \mathbb{E}[Q_2(t) \\ \mathbb{E}[Q_1'(t)] & \mathbb{E}[Q_2'(t) \end{bmatrix}$$

or as written in a row version:

$$W(\mathbb{E}[Q_1(t)], \mathbb{E}[Q_2(t)]) = \mathbb{E}[Q_1(t)]\mathbb{E}[Q_2'(t)] - \mathbb{E}[Q_2(t)]\mathbb{E}[Q_1'(t)] \tag{6.41}$$

Solving the SDEs will provide the traffic generated sequentially by the short packets source and by the fat packets source respectively. The homogeneous (general) solution of the SDE is determined by the characteristic equation. As the SDE equation of:

$$\mathbb{E}[Q_P(t)] = \mathbb{E}[Q_1(t)]\eta_1(t) + \mathbb{E}[Q_2(t)]\eta_2(t) \tag{6.42}$$

which eventually leads to the particular solution of the non-homogeneous second-order SDE:

$$\begin{aligned}\mathbb{E}[Q_P(t)]) = &- \mathbb{E}[Q_1(t)] \int \frac{\mathbb{E}[Q_2(t)]g(t)}{W(\mathbb{E}[Q_1(t)], \mathbb{E}[Q_2(t)])} \\ &+ \mathbb{E}[Q_2(t)] \int \frac{\mathbb{E}[Q_1(t)]g(t)}{W(\mathbb{E}[Q_1(t)], \mathbb{E}[Q_2(t)])}\end{aligned} \tag{6.43}$$

In the SDE equation above (6.35), being second-order and non-homogeneous, the characteristic polynomial, whose highest order coefficient is unitary ($a_2(t) = 1$), will have two roots only, which can be real, or complex numbers depending on the outcome of the following condition:

$$\mathcal{P}(s) = (\lambda_{TO} + \frac{\lambda_{TD}}{2} + \frac{\mathbb{E}[\alpha(t)]}{(1+Q_0)^2})^2 - 4\mathbb{E}[f(ACK)] \tag{6.44}$$

The $|\mathcal{P}(s)|$ discriminant of the characteristic polynomial is as in (6.44) and can either be negative ≤ 0 or ≥ 0 positive; a negative value will lead to complex conjugate roots of the $\mathcal{P}(s)$ which will generate a complementary solution such as:

$$\mathbb{E}[Q_C(t)] = C_1 e^{\lambda t} cos\mu t + C_2 e^{\lambda t} sin\mu t \tag{6.45}$$

which is recommended to be implemented practically for the control of the equation (6.35) and $\mathbb{E}[Q_{C_e}(t)]$ $\mathbb{E}[Q_{C_m}(t)]$ because the sin/cosin laws, if well implemented, will lead to faster reaching the steady-state of either the elephants or mice flows.

As the $f(ACK)$ is represented by a small packet containing the ACK signal transmitted by the receiver acknowledging that the packets sent by the sender are stored correctly in the receiver's buffer. These reasons lead to comparing $(\lambda_{TO} + \frac{\lambda_{TD}}{2} + \frac{\mathbb{E}[\alpha(t)]}{(1+Q_0)^2})^2$ to $\mathbb{E}[f(ACK)]$ to implement the sin/cosin solution $(\lambda_{TO} + \frac{\lambda_{TD}}{2} + \frac{\mathbb{E}[\alpha(t)]}{(1+Q_0)^2})^2 \leq \mathbb{E}[f(ACK)]$.

6.7 A New Algorithm and Method for Controlling Mouse and Elephant Traffic in DCs

A ratio type of control can be derived if the feedback obtained above can be homogenized such that m packets of mouse traffic can represent e packets of elephant traffic. As such, a ratio of $\frac{m}{e}$ or $\frac{e}{m}$ can be used when one type of traffic increases/decreases too much. The ratio above represents the amount of the combined traffic over the last router interface whose bandwidth is B_{wd}.

In order to control the flows of the elephant and mouse packets, an advanced traffic control architecture is used in order to maintain the relationship between the two variables e and m, while a third variable B_{wd} is required to be measured, and kept steady. The proposed control architecture is used to maintain the mouse flow rate (the dependent, thus the controlled flow) and the elephant flow in a constant composition proportion. This means that the elephants flow is defined at a specified number which is used to control proportionality such that the mice flow is relatively controlled in a ratio relatively to another independent wild feed stream of packets in order to control the composition of a resultant which is the full bandwidth of the router interface connected to the receiver. The architecture of the network comprises switches and routers capable of dynamically executing the equations controlling the ratio of mice (numbers) to elephants. To begin, a P controller is assumed, i.e.,

$$\mathbb{E}[\mathcal{A}_{\mathcal{C}}(t) = K_{PC}\mathbb{E}[\mathcal{E}_{\mathcal{R}}(s)] \tag{6.46}$$

where K_{PC} is the proportional gain, $\mathcal{A}_{\mathcal{C}}(s)$ is the output of the controller which is selected to be a PID controller, and $\mathcal{E}_{\mathcal{R}}(s)$ is the error obtained through comparing the dynamic ratio of the mice and elephants to their prescribed value $\frac{\mathcal{M}}{\mathcal{E}_{\mathcal{EL}}}$ of their dynamic bandwidth, which has to be stable in a ratio of $\frac{4}{1}$.

Assuming that a Proportional (P) Controller is used to maintain the traffic on both elephant and mouse branches, the mapping of the blocks of the Figure 6.5

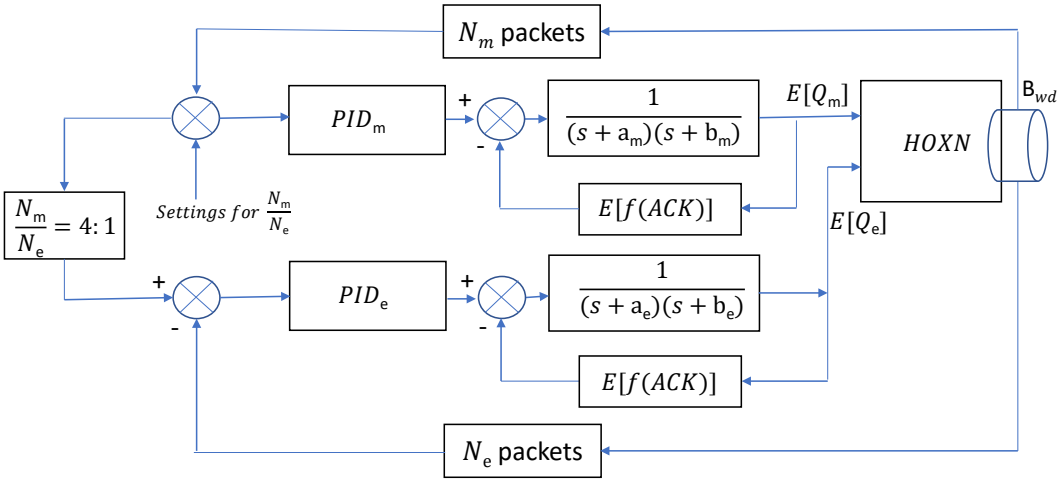


FIGURE 6.5: Mice And Elephants Ratio Control

into the corresponding equations developed above, then the combined mouse and elephant traffic is controlled such that the ratio of 4 : 1 is maintained anytime a transmission of packets takes place.

Using Figure 6.5 and the transfer functions obtained by reducing the two processes in one, one obtains the following input-output relationship:

$$\begin{aligned}\frac{d^2\mathbb{E}[Q(t)]}{dt^2} &+ (\lambda_{TO} + \frac{\lambda_{TD}}{2} + \frac{\mathbb{E}[\alpha(t)]}{(1+Q_0)^2}\lambda_t)\frac{d\mathbb{E}[Q(t)]}{dt} \\ &+ (\lambda_{TO} + \frac{\lambda_{TD}}{2} + \frac{\mathbb{E}[\alpha(t)]}{(1+Q_0)^2}\lambda_t)\mathbb{E}[f(ACK)]\mathbb{E}[Q(t)] \\ &= \mathbb{E}[u_1(t)]\frac{1}{R_{TT}} + \mathbb{E}[\beta(t)]\mathbb{E}[W(t)]\lambda_{Z_t}\end{aligned} \tag{6.47}$$

For proportional controllers, bias $\bar{p}$ can be adjusted through a manual reset and saturates at a certain gain. In practice, the industry adopted a Proportional(P), Integral(I), Derivative(D) or a PID controller is recommended. In this case the controller's equation is:

$$\mathbb{E}[\mathcal{A}_\mathcal{C}(t)] = K_{PC}\mathbb{E}[\mathcal{E}_\mathcal{R}(t)] + \frac{1}{\tau_I}\int_0^t [\mathcal{E}_\mathcal{R}(t)]dt + \tau_D\frac{d[\mathcal{E}_\mathcal{R}(t)]}{dt} \tag{6.48}$$

where the controller parameters are proportional gain K_{PC}, integral time τ_I, and derivative time τ_D. To investigate the performance of the PID controller in maintaining the traffic level of loop controlling the mice traffic or the elephant traffic

respectively, the following notations are necessary for simplification of the formula. The in-series connection of the PID controller and the combined controller and TCP/IP is described by the following equation:

$$\begin{aligned} g(\frac{d^3\mathbb{E}[Q(t)]}{dt^3} &+ (a_1 + K_{W-P})\mathbb{E}[f(ACK)]\tau_D\frac{d^2\mathbb{E}[Q(t)]}{dt^2} \\ &+ (a_0 + K_{PID}K_{W-P})\mathbb{E}[f(ACK)]\frac{d\mathbb{E}[Q(t)]}{dt} \\ &+ \mathbb{E}[f(ACK)]K_{W-P}\mathbb{E}[Q(t)]) \\ &= K_{W-P}\tau_I\tau_D\frac{d^2\mathbb{E}[U_1(t)]}{dt^2} + K_{W-P}\frac{d\mathbb{E}[U_1(t)]}{dt} + K_{W-P}\frac{1}{\tau_I}\mathbb{E}[U_1(t)] \end{aligned} \tag{6.49}$$

where:

$$g = \frac{K_{W-P}}{\tau_I} \tag{6.50}$$

is the system gain.

Using the following notations, such as $a_0, a_1, \alpha_0, \alpha_1, \alpha_2, \beta_0, \beta_1, \beta_2$ as defined below:

$$\begin{aligned} a_1 &= \lambda_{TO} + \lambda_{TD} + \frac{Q_0^2\lambda_t}{(1+Q_0)^2} \\ a_0 &= (\lambda_{TO} + \lambda_{TD})\frac{Q_0^2\lambda_t}{(1+Q_0)^2} \end{aligned} \tag{6.51}$$

and by a more compact notation now the left hand side of the closed loop equation (6.49) becomes:

$$\begin{aligned} \alpha_2 &= a_1 + K_{W-P}\mathbb{E}[f(ACK)] \\ \alpha_1 &= a_0 + K_{PID}K_{W-P}\tau_D\mathbb{E}[f(ACK)] \\ \alpha_0 &= K_{W-P}\frac{1}{\tau_I}\mathbb{E}[f(ACK)] \end{aligned} \tag{6.52}$$

and:

$$\begin{aligned} \beta_2 &= K_{W-P}\tau_I\tau_D \\ \beta_1 &= K_{W-P}K_{PID} \\ \beta_0 &= K_{W-P}\frac{1}{\tau_I} \end{aligned} \tag{6.53}$$

The final dynamic traffic model is written as follows:

$$\begin{aligned} g(\frac{d^3\mathbb{E}[Q(t)]}{dt^3} &+ \alpha_2\frac{d^2\mathbb{E}[Q(t)]}{dt^2} + \alpha_1\frac{d\mathbb{E}[Q(t)]}{dt} + \alpha_0\mathbb{E}[Q(t)]) \\ &= \beta_2\frac{d^2\mathbb{E}[U_1(t)]}{dt^2} + \beta_1\frac{d\mathbb{E}[U_1(t)]}{dt} + \beta_0\mathbb{E}[U_1(t)] \end{aligned} \tag{6.54}$$

As depicted by (6.54), the traffic dynamics are described by a third-order differential equation for the queue length expectation. The solution for the differential equation is developed based on the three roots of the characteristic equation (6.54); one is real, and the remaining two have to be determined on the assumption that they are complex conjugated. The real root damps the solution, while the two complex-conjugated ones, if well placed in the complex plane, will make the transient process arrive at the final value more quickly. The three roots of the characteristic equation can be written as follows:

$$\begin{aligned} s_1 &= -\alpha \\ s_2 &= K_1\alpha + jK_2\alpha \\ s_3 &= K_1\alpha - jK_2\alpha \end{aligned} \tag{6.55}$$

where value α is among the divisors of the free term of the characteristic equation $K_{W-P}\mathbb{E}[f(ACK)]$. The following set of equations can be used for solving the system of equations which link $\alpha_0, \alpha_1, \alpha_2$ to α, K_1, K_2:

$$\begin{aligned} \alpha_0 &= \alpha^3(K_1^2 + K_2^2) \\ \alpha_1 &= \alpha^2(K_1^2 + K_2^2 + K_1) \\ \alpha_2 &= \alpha(2K_1 + 1) \end{aligned} \tag{6.56}$$

With α, K_1, K_2 obtained the solution for the above equation can be written as three components which correspond to the terms at the right-hand side sum of terms. The formal solution will be used "mutatis mutandis" for both the mouse and elephant traffic processes.

$$\mathbb{E}[Q(t)] = \mathbb{E}[Q_1(t)] + \mathbb{E}[Q_2(t)] + \mathbb{E}[Q_3(t)] \tag{6.57}$$

The mice equation above (6.57) is written as follows:

$$\mathbb{E}[Q_m(t)] = \mathbb{E}[Q_{1m}(t)] + \mathbb{E}[Q_{2m}(t)] + \mathbb{E}[Q_{3m}(t)] \tag{6.58}$$

while for elephant flows, the traffic through the sender to the receiver path is described by the following equation:

$$\mathbb{E}[Q_e(t)] = \mathbb{E}[Q_{1e}(t)] + \mathbb{E}[Q_{2e}(t)] + \mathbb{E}[Q_{3e}(t)] \tag{6.59}$$

It is well known that such equations are better tuned if there is only one real negative root, with the other two being complex-conjugate, while the real part of the complex-conjugate ones is negative. The real root is better calculated if it is in high "frequencies" relatively speaking, relatively considered high if compared to the natural frequency of the process. In this case, the TCP/IP process dynamics are sometimes as fast as the process can carry Gbps.

The solution for $\mathbb{E}[Q_1(t)]$ that corresponds to the first component of the right-hand side of the $\mathbb{E}[Q(t)]$ equation is:

$$\begin{aligned}\mathbb{E}[Q_1(t)] = &\frac{\mathbb{E}[U_1(t)]}{[(K_1-1)^2+K_2^2]}[(K_1^2+K_2^2)e^{-\alpha t}] \\ &- \frac{e^{-K_1\alpha t}}{K_2}[(K_1^2+K_2^2-K_1)sin(K_2\alpha t) + K_2(2K_1-1)cos(K_2\alpha t)]\end{aligned} \tag{6.60}$$

This solution corresponds to the free term in (6.49). The second term of the $Q(t)$ solution of the mouse and elephant traffic corresponds to the first derivative component on the right hand side of (6.49). The $Q_2(t)$ term in (6.57) is therefore:

$$\begin{aligned}\mathbb{E}[Q_2(t)] = \frac{\mathbb{E}[U_1(t)]\alpha(K_1^2+K_2^2)}{(K_1-1)^2+K_2^2)}[&\frac{e^{-K_1\alpha t}}{K_2}(K_1-1)sin(K_2\alpha t) + K_2cos(K_2\alpha t)] \\ &[(K_1^2+K_2^2-K_1)sin(K_2\alpha t) + K_2cos(K_2\alpha t)]\end{aligned} \tag{6.61}$$

The third term of the $Q(t)$ solution for the mouse and elephant traffic corresponds to the second derivative component written on the right-hand side of (6.49):

$$\begin{aligned}\mathbb{E}[Q_3(t)] = &\frac{\mathbb{E}[U_1(t)]\alpha^2(K_1^2+K_2^2)}{[(K_1-1)^2+K_2^2]}[e^{-\alpha t} - (K_1^2+K_2^2)e^{-\alpha t}] \\ &- \frac{e^{-K_1\alpha(t)}}{K_2}[(K_1^2+K_2^2-K_1)sin(K_2\alpha t) + K_2cos(K_2\alpha(t))]\end{aligned} \tag{6.62}$$

Assuming that a Proportional Integral Derivative (PID) controller is used to maintain the traffic on both the elephant and mouse branches, the mapping of the blocks of the Figure 6.5 into the corresponding equations developed above, then the combined mouse and elephant traffic is controlled such that the ratio of $4:1$ is maintained anytime a transmission of packets takes place.

The PID controller settings allow the use of the maximum of HyperOXN bandwidth if the PID controllers are set for "Model-based Predictive Control" or MPC.

6.8 Simulation

6.8.1 Settings

The capacity of each link is configured as 10Gbps, and the buffer size of the router is set to 320 packets. The link delay is configured to 10μs, and RTT is around 100μs. The number of TCP sessions $N_{TCP_{Ses}}$ is 10 for mice and elephants respectively. The simulation uses a combination of OMNET++ and Wireshark with PID controllers, as shown in Figure 6.5.

6.8.2 Results

As shown in Figures 6.6 and 6.7, the instantaneous queue length converges and results in around 48 packets in oscillation for mouse flows, and 190 packets for elephant flows.

The mouse and elephant traffic is well controlled such that the ratio of $\frac{e}{m}$ $(4:1)$ is maintained under the SDE model and PID controllers.

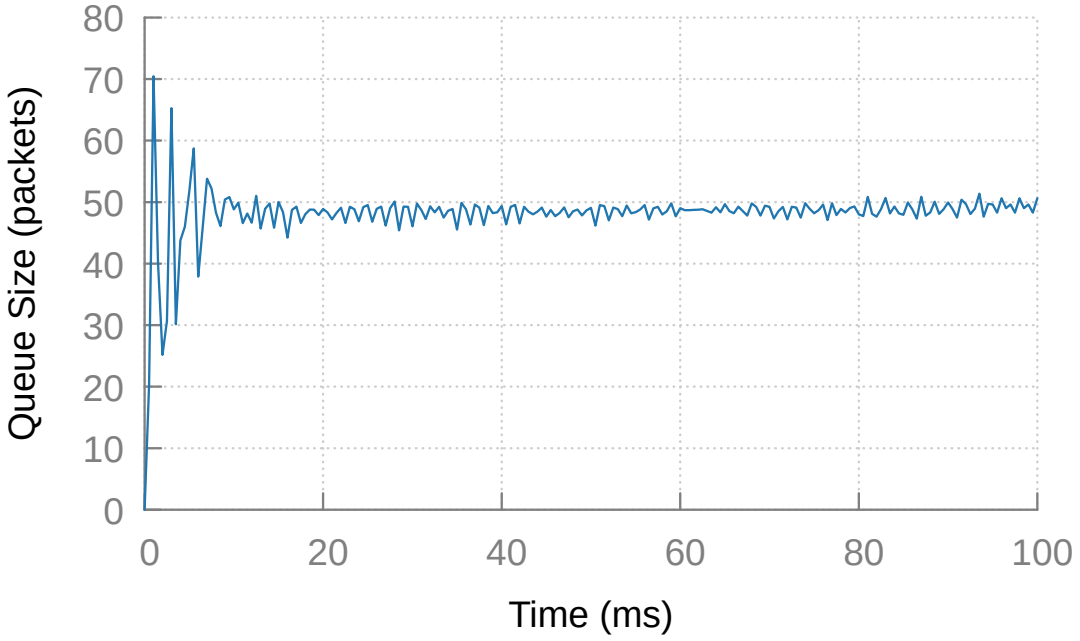


FIGURE 6.6: Instantaneous Queue Length for Mice Flows

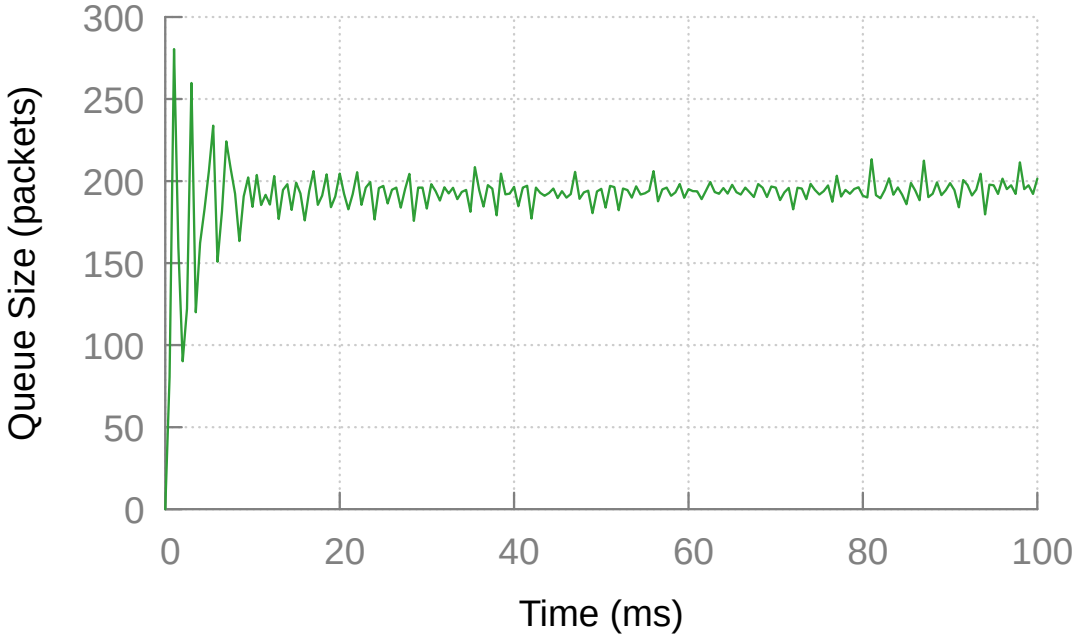


FIGURE 6.7: Instantaneous Queue Length for Elephant Flows

6.9 Conclusion

A new SDE formalism for traffic control based on traffic classification by taking into account the number and size of packets, splitting the DC traffic into two major traffic classes (mice and elephants), is introduced in this chapter. One implements a ratio control method to separate the DC network usage and achieves high network utility. As a traffic input to the HyperOXN architecture for simulations, better performance can be accomplished, as shown in Chapter 7.

Chapter 7

HyperOXN Performance Evaluation: Packet-Level Simulation

There are four aspects to evaluating a DCN performance: topology, routing, congestion control and load balancing. The flow model abstracts over the complexity of traffic engineering, which focuses on a topology intrinsically and neglects any inefficiencies from routing and flow control algorithms. The packet-level simulation is done to overcome the trade-offs made by flow evaluation; however, translating performance in flow models to that at the packet-level can be challenging, particularly with dynamic, unpredictable traffic.

The packet-level simulation will be focusing on load balancing and congestion control. ECMP, ECMP with Valiant Load Balancing (HYB) [19], and HOLMES^{+} are employed in the comparison based on flowlet switching [186] [43].

7.1 Packet-level simulation

The experiments in this section use packet-level simulation with routing and congestion control to target the optimal results, as in Section 4.1.

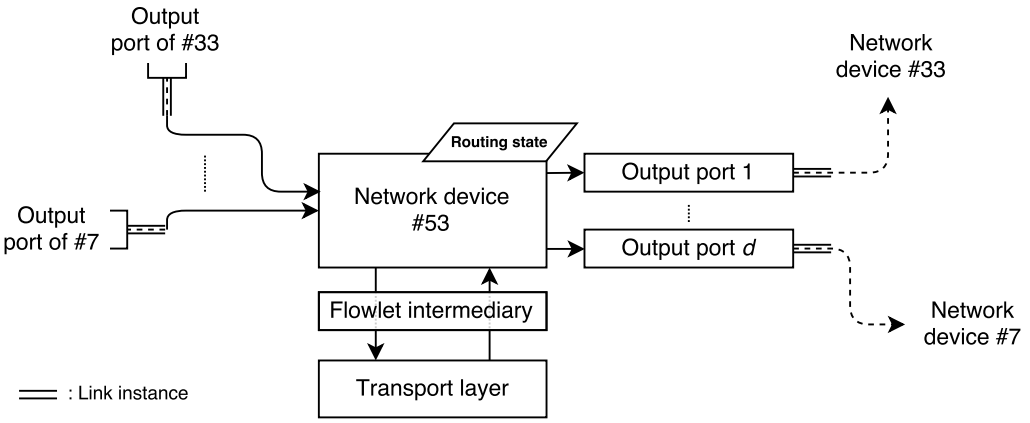


FIGURE 7.1: NetBench infrastructure example. Network device #53 has two bi-directional links to #33 and #7 [19].

7.1.1 Experimental setup

7.1.1.1 Settings

The packet-level simulation tool, *NetBench*, which can be downloaded online 7.1, is adopted for the comparison of three topologies: Fat-Tree, Xpander, and HyperOXN. All these topologies are constructed by the identical switch with radix $r = 128$ and a $10Gps/s$ link bandwidth ($20ns$ delay) for each port. There are $12,800$ servers and 500 switches for the full-bisection bandwidth Fat-Tree, while the Xpander is built with 376 switches with a network degree of 93 and $13,160$ servers at a 30% lower switch cost, as in [19]. HyperOXN has a 15% lower total cost advantage, as shown in Section 3.6.1.4 (page 91) such that the full-bisection bandwidth $HyperOXN(2,1,2,24,23)$ (denoted as HOXN576) supporting 13248 servers and 576 switches will be used for the following comparisons at the same cost of the Fat-Tree. The lower cost $HyperOXN(2,1,2,20,32)$ (denoted as HOXN400) with 400 switches and 12800 servers is also used for the comparisons. The oversubscription value for HOXN400 is 0.5.

As shown in Figure 7.1, NetBench is a discrete packet simulator. It has three main modules: the infrastructure features, the routing initialization, and the traffic generation model.

7.1.1.2 Routing and congestion control with load balancing

ECMP, HYB [19] and HOLMES$^+$ are examined for the comparisons in Fat-Tree, Xpander, and HyperOXN. The congestion control mechanism from DCTCP will be used in the experiments, with the ECN marking threshold set to 20 full-sized packets. Further, flowlet switching [186] for the above three topologies will be employed with the time gap set to $50\mu s$.

1. $ECMP$: Utilizes only shortest path routing on Fat-Trees (denoted as Fat-Tree-ECMP).
2. HYB: A simple hybrid scheme on Xpander (denoted as Xpander-HYB) combines the advantages of ECMP and VLB. First, it uses ECMP routing until the amount of flow data sent by the sender exceeds the threshold Q, and after that it will switch to VLB for the remainder of the flow.
3. $HOLMES^+$: This is detailed in Chapter 5, and denoted as HOXN400-HOLMES or HOXN576-HOLMES for HOXN400 and HOXN576 respectively.

7.1.1.3 Traffic workloads

The workloads in the experiments are defined by distributions for flow arrivals, flow sizes, and participating pairs of servers.

A) Three flow-size distributions are employed in the workloads: Alizadeh Web Search (ALW) [90], Pareto (PAR) [187], and SDE, proposed in Chapter 6. The distributions for ALW, SDE, and PAR support a diverse mix of mouse and elephant hybrid flows with heavy-tailed characteristics.

1. ALW: The discrete flow-size distribution, introduced in pFabric [90] as web search workloads is cited as the more challenging of the two distributions in that paper and is also used by ProjecToR [79]. ALW is considered a moderately skewed flow-size distribution with a high average, in which over 95% of all Bytes are from the 30% of the flows that are 1-20MB.

2. $PAR(\theta,\mu)$: A Pareto distribution which is also used in HULL [187]. μ represents the average flow size, and θ for the shape. PAR is considered a heavily skewed flow size distribution with a low average. The shape parameter θ is set to 1.05, and μ = 100KB. This distribution simulates a heavy-tailed workload where most of the flows are small but the majority of traffic comes from large flows, as is commonly observed in production DCNs. There are 95% of mouse flows, which of size is less than 100KB and comprise a little over 25% of total traffic, but 0.03% of the flows are elephant flows that are larger than 10MB, and contribute over 40% of all Bytes.

3. $SDE(4:1)$: A new SDE formalism for traffic control based on traffic classification by taking into account the number and size of packets, splitting the DC traffic into two major traffic classes (mice and elephants), is proposed in Chapter 6. The combined mouse and elephant traffic is controlled such that the ratio of 4 : 1 is maintained anytime a transmission of packets takes place. The HOXN is evaluated using the ratio of 4 : 1 in the experiments.

B) Three communication pair probability distributions are used as follows:

1. Node all-to-all-fraction, $A2A(x)$: A certain fraction x of nodes is chosen to communicate in all-to-all traffic. The probability of a flow to start between any pair of servers at active racks is uniform. This distribution can approximate Facebook's workloads [4].

2. $ProjecToR$: A node-level directed communication pair probability from a Microsoft DCN [79]. It is heavily skewed, in which only a few nodes ("hot" racks) are very likely to participate, with 77% of Bytes transferred between 4% of the TOR-pairs. The ProjectTor distribution can be used to simulate the skewed TM, in which only a small fraction of racks participate in the simulation, with no flows between non-involved servers.

3. $Permute(x)$: A random permutation traffic distribution between x fraction of racks, with the rest inactive. This is a challenging traffic workload, as

it concentrates loads on individual source-destination pairs, which tends to stress the load balance of the topology and routing algorithm [3]. Also, the rack-to-rack traffic consolidation of flows limits opportunities for load balancing.

C) Assume that all flow arrivals follow a Poisson distribution with a flow arrival rate λ which determines how many flows are started per second. The flow arrival rate is used to create the corresponding load and bottlenecks on the network.

7.1.2 Results and Analysis

When a flow arrives, one pair of servers for the flow is picked from a chosen flow size distribution. The experiments calculate three metrics in the [0.5s, 1.5s) time interval: average FCT for all flows, 99^{th} percentile FCT for mouse flows less than 100KB, and average throughput for elephant flows($\geq$100KB).

7.1.2.1 A2A Traffic Matrices

The pFabric's flow size distribution (ALW) and 167 flow arrivals per second per server are used in the experiment for $A2A(x)$. As shown in Figure 7.2, for low fraction($\leq$5%) of active nodes, the Xpander-HYB outperforms the full-bisection bandwidth Fat-Tree-ECMP, but both HOXN400-HOLMES and HOXN576-HOLMES rival the performance of the above two topologies. This is because HOLMES^{+} leverages global information to achieve relatively optimal load balances among the multiple shared paths. It also guarantees low latency for short-lived mouse flows, and high throughput for long-lasting elephant flows with the separation of these two types of flows in DCN paths. It is worth noting that oversubscription of the HOXN400 is 50% compared with the full-bisection bandwidth Fat-Tree. Even Xpander-HYB is able to attain sufficient path diversity, but it uses longer paths, especially for mouse flows, for which it is more reliant on round-trip time. For larger fractions, throughput and overall average FCT (over all flows) with

Xpander-HYB deteriorate. As expected, HOXN400-HOLMES and HOXN576-HOLMES can achieve a reduced 99^{th} percentile FCT (Figure 7.3) for mouse flows and higher throughput for elephant flows (Figure 7.4).

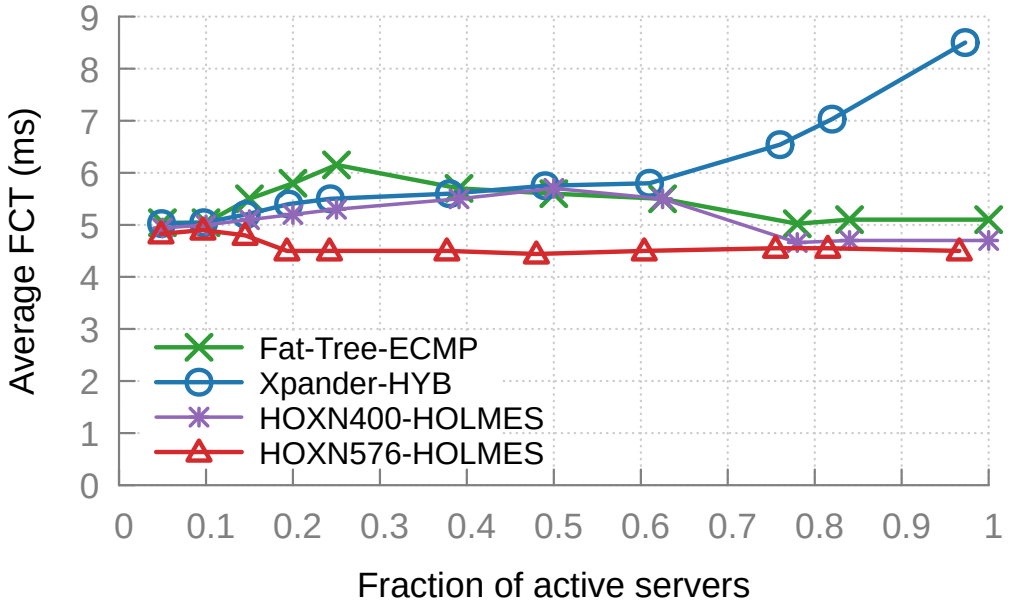


FIGURE 7.2: Average FCT w/ $A2A(x)$ on the increasing fraction of active servers

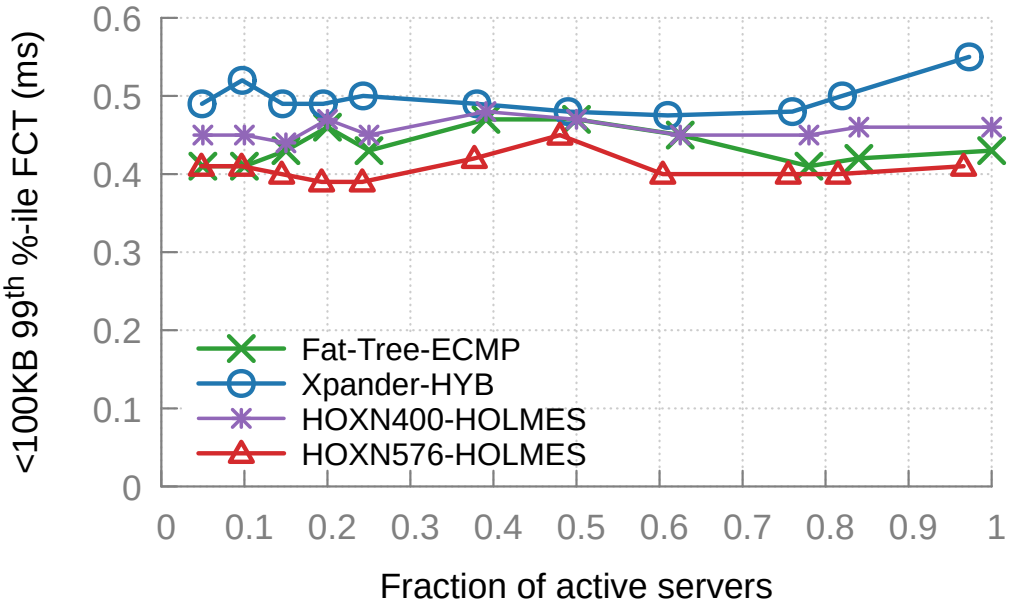


FIGURE 7.3: 99^{th} %-tile FCT for short flows w/ $A2A(x)$ on the increasing fraction of active servers

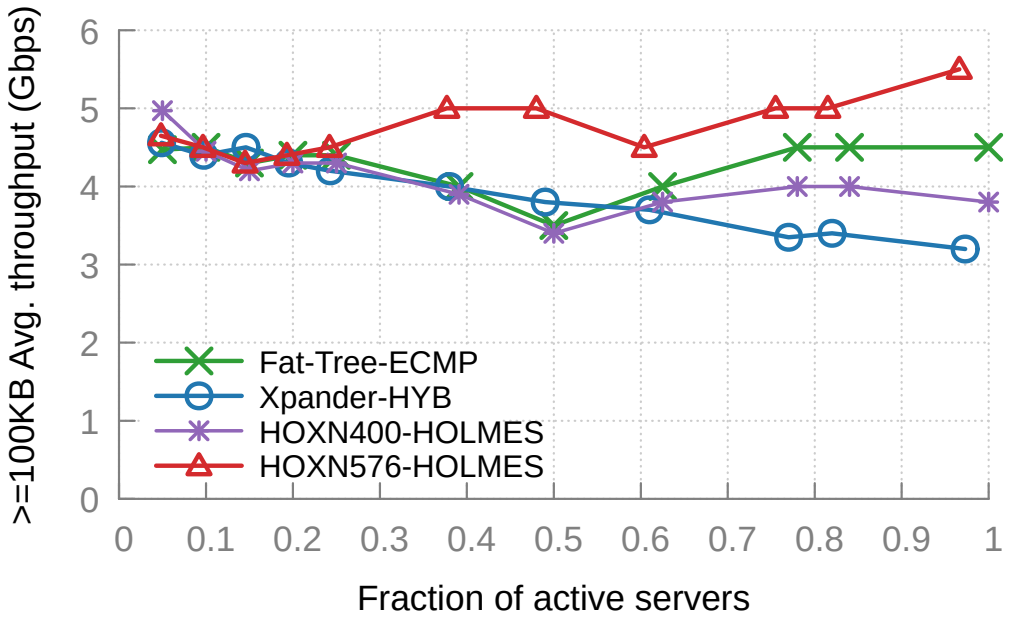


FIGURE 7.4: Average throughput for large flows w/ $A2A(x)$ on the increasing fraction of active servers

One can control the combined mouse and elephant traffic and maintain the ratio of 4 : 1 more accurately. When SDE traffic is used in the experiment for $A2A(x)$, as shown in Figures 7.5, 7.6 and 7.7, HOXN400-HOLMES and HOXN576-HOLMES can achieve better performance than the other two while leveraging Holistic Mice-Elephants Stochastic Scheduling (HOLMES$^+$). As noted in Chapter 5, HOLMES$^+$ is based on a network partition solution that separately partitions all the end-to-end DCN paths into two logical sub-networks in order to isolate the transmission of elephant and mouse flows. Significant low latency and high throughput can be achieved compared to other topologies with different scheduling methods.

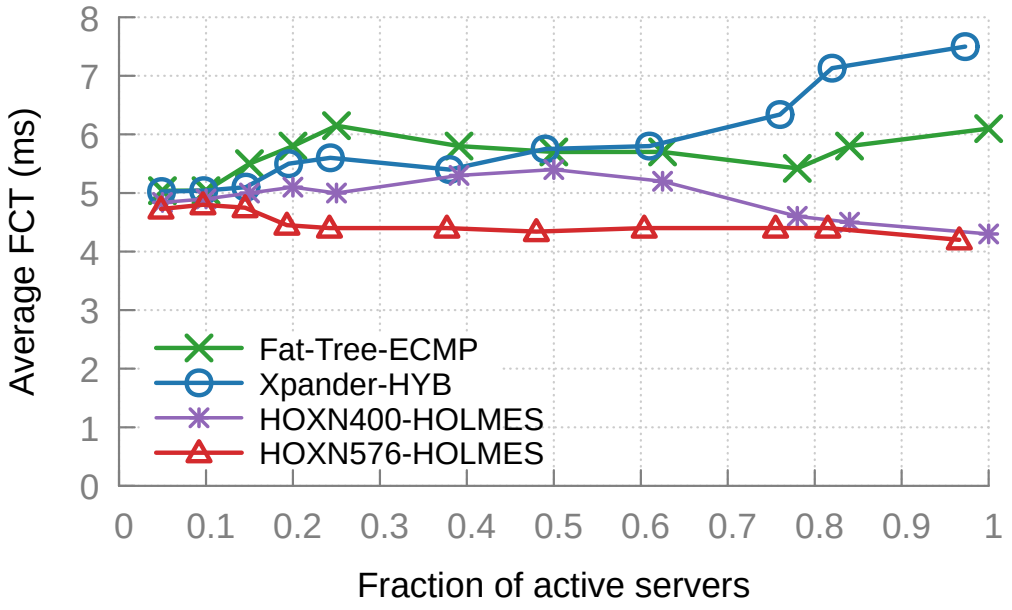


FIGURE 7.5: Average FCT w/ $A2A(x)$ on the increasing fraction of active servers w/ SDE

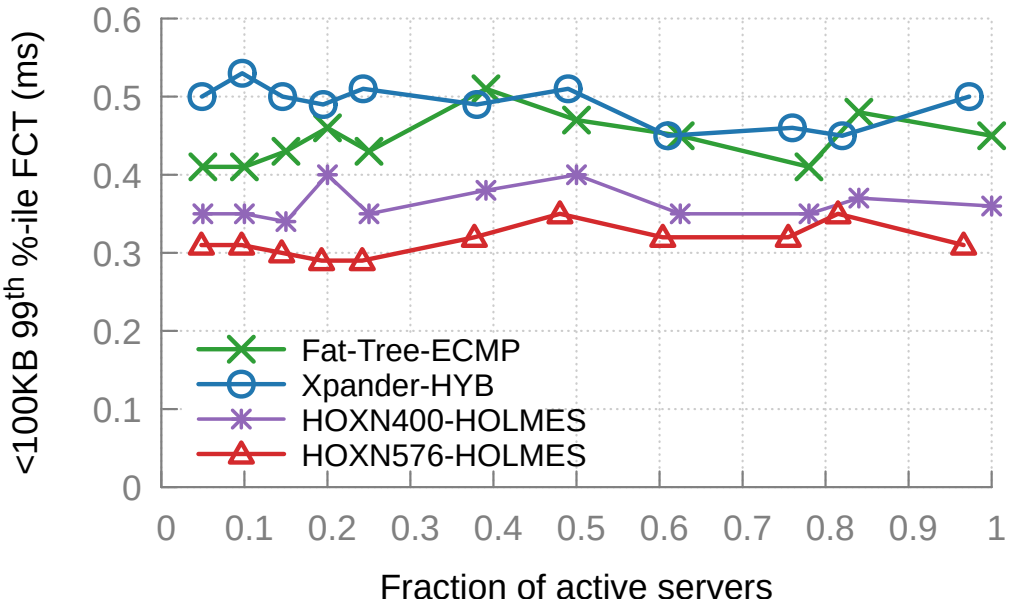


FIGURE 7.6: 99^{th} %-tile FCT for short flows w/ $A2A(x)$ on the increasing fraction of active servers w/ SDE

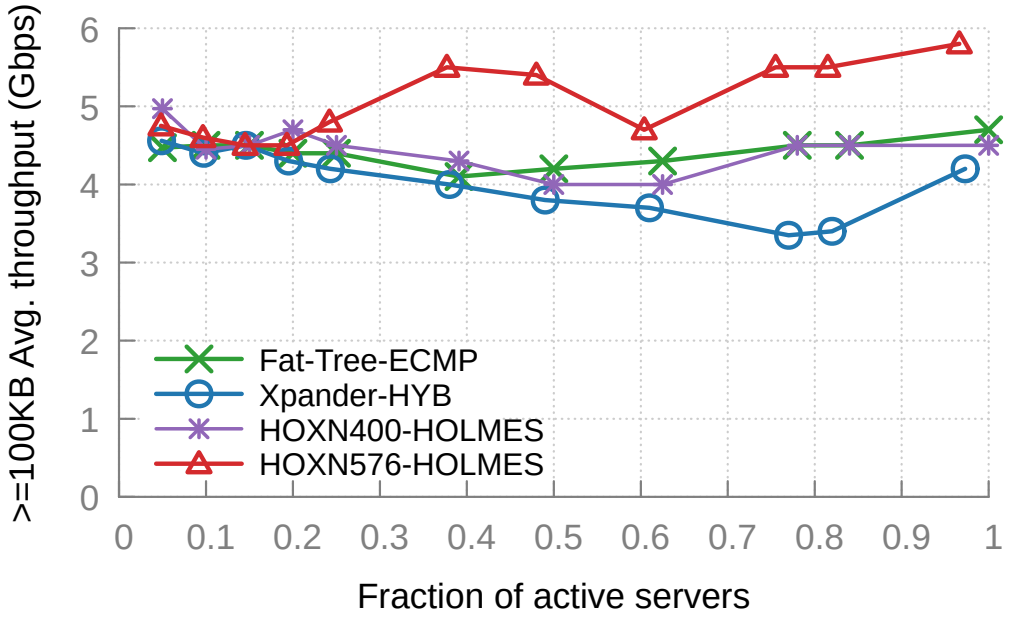


FIGURE 7.7: Average throughput for large flows w/ $A2A(x)$ on the increasing fraction of active servers w/ SDE

When x, the fraction of servers, is fixed at 0.31, Figures 7.8, 7.9, and 7.10 show the results for the A2A with increasing flow arrival rates, λ. The HOXN is evaluated using the Pareto flow-size distribution $PAR(1.05, 100KB)$, with most of the flows being mouse flows. The HOXN400 and HOXN576 are better in all results comparing with the other two topologies. Besides the HOLMES^{+} algorithm, the topologies for HOXN400 and HOXN576 have a only two-dimensional structure with shorter paths, which also results in a lower FCT than Xpander and Fat-Tree.

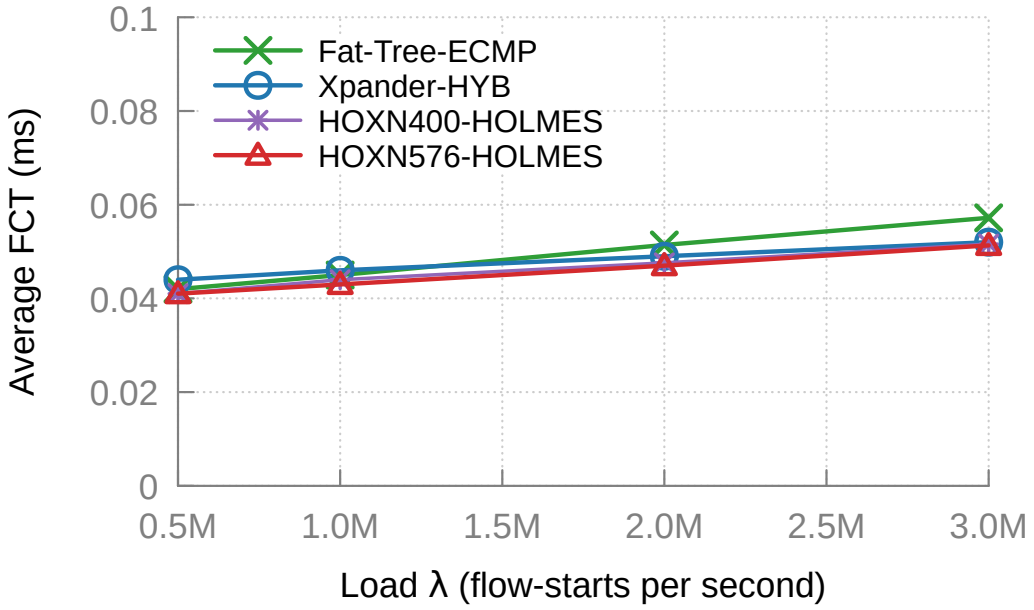


FIGURE 7.8: Average FCT w/ $A2A(0.31)$ on the increasing aggregate flow arrival rate

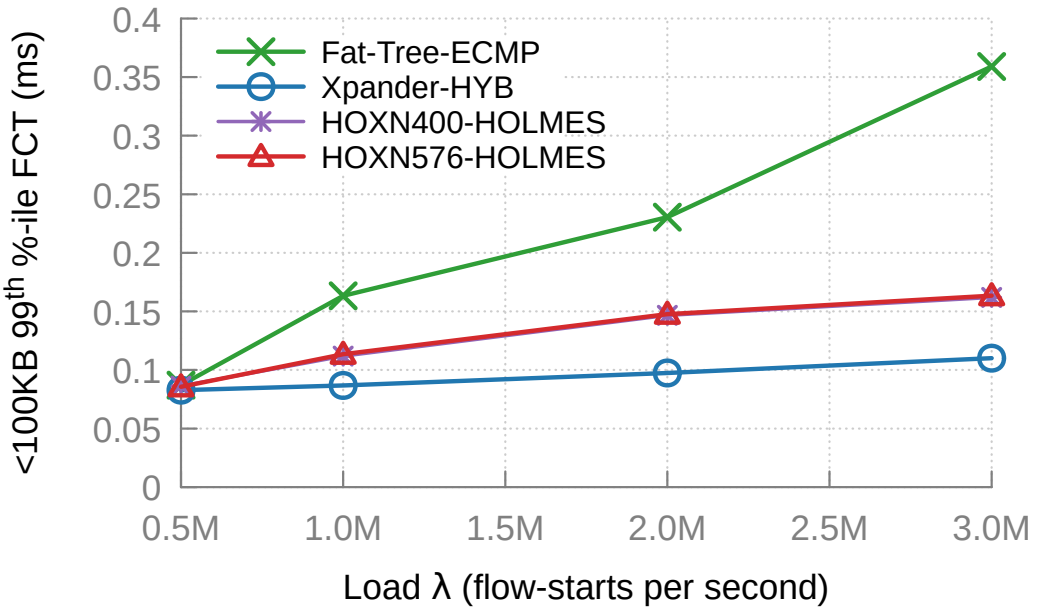


FIGURE 7.9: 99^{th} %-tile FCT for short flows w/ $A2A(0.31)$ on the increasing aggregate flow arrival rate

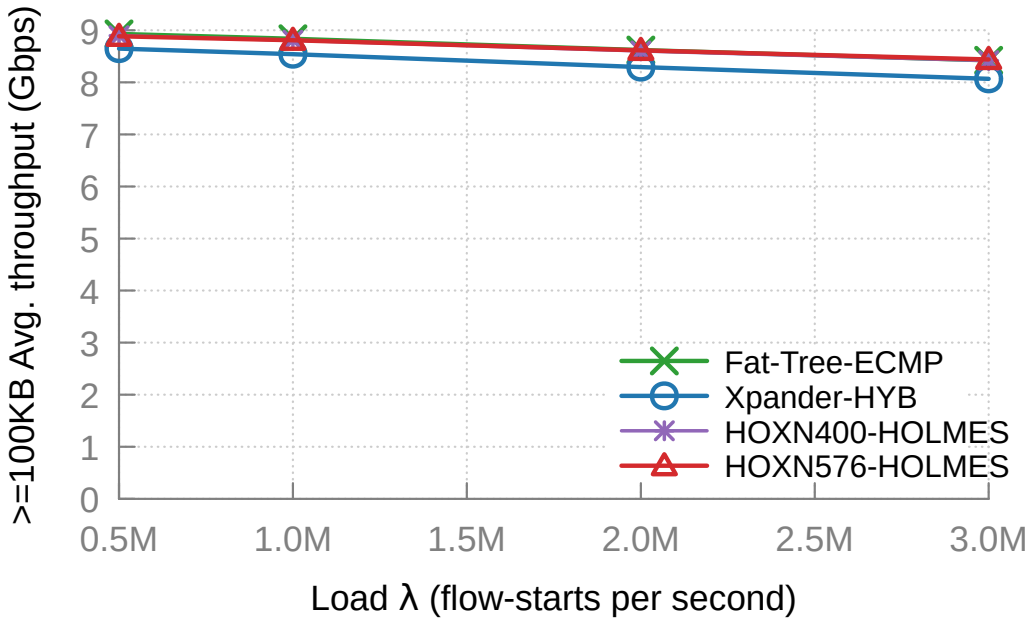


FIGURE 7.10: Average throughput for large flows w/ $A2A(0.31)$ on the increasing aggregate flow arrival rate

7.1.2.2 Permute Traffic Matrices

More challenging $Permute(x)$ workloads are simulated on the topologies. The analogous results as $A2A(x)$ are shown in Figure 7.11, Figure 7.12, and Figure 7.13. HOXN400 and HOXN576 demonstrate more advantages over the Fat-Tree-ECMP and Xpander-HYB.

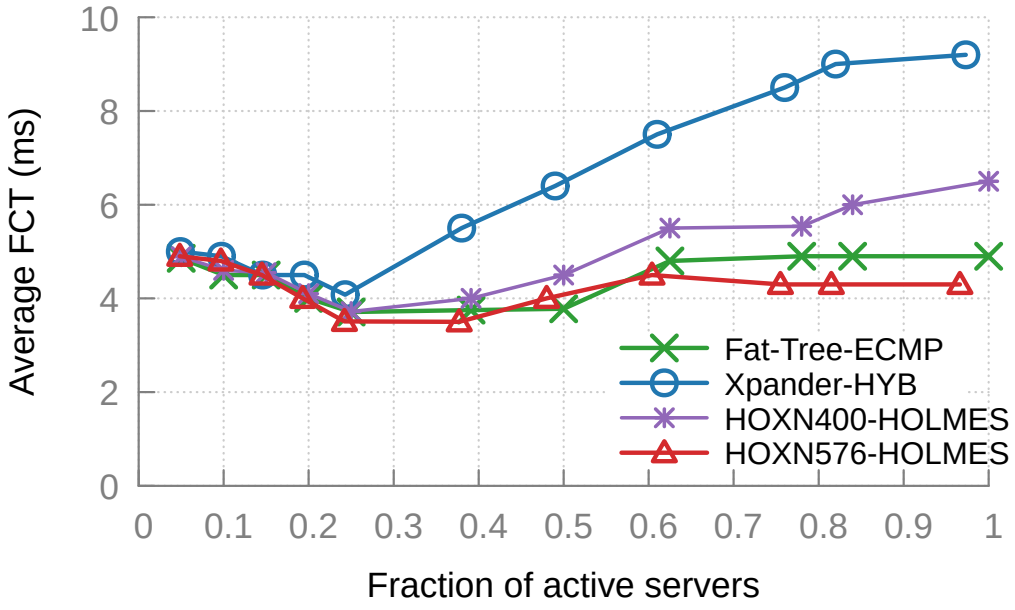


FIGURE 7.11: Average FCT w/ $Permute(x)$ on the increasing fraction of active servers

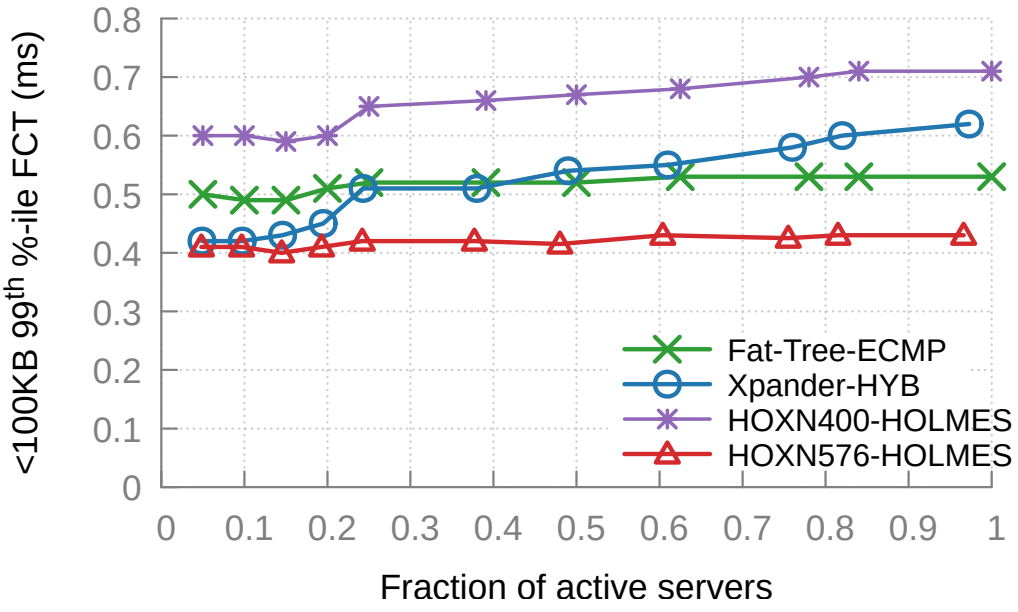


FIGURE 7.12: 99^{th}%-tile FCT for short flows w/ $Permute(x)$ on the increasing fraction of active servers

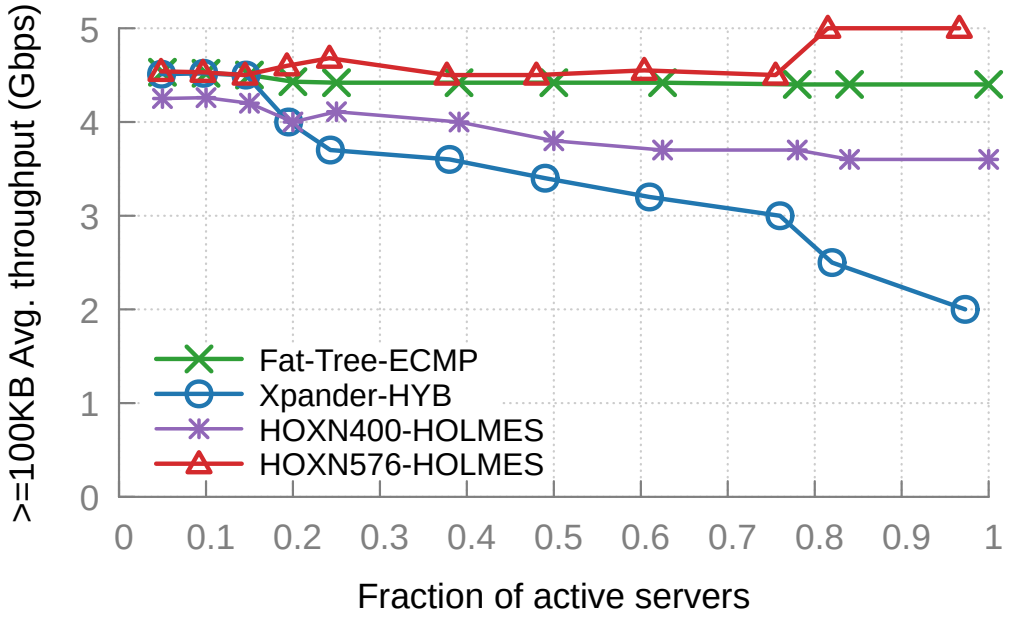


FIGURE 7.13: Average throughput for large flows w/ $Permute(x)$ on the increasing fraction of active servers

$Permute(0.31)$ with increasing flow arrival rates for ALW flow size distribution is shown in Figure 7.14, Figure 7.15, and Figure 7.16. HOXN400 and HOXN576 match the performance closely for Fat-Tree-ECMP and Xpander-HYB. This scenario hampers load balancing as of the rack-to-rack consolidation. As such, Fat-Tree-ECMP can also achieve good performance.

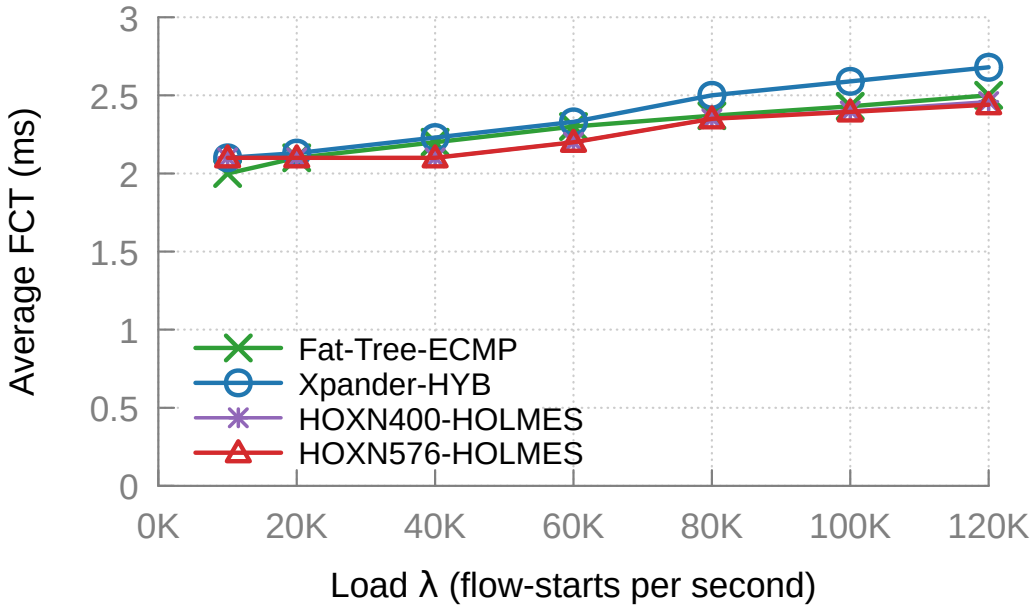


FIGURE 7.14: Average FCT w/ *Permute*(0.31) on the increasing aggregate flow arrival rate

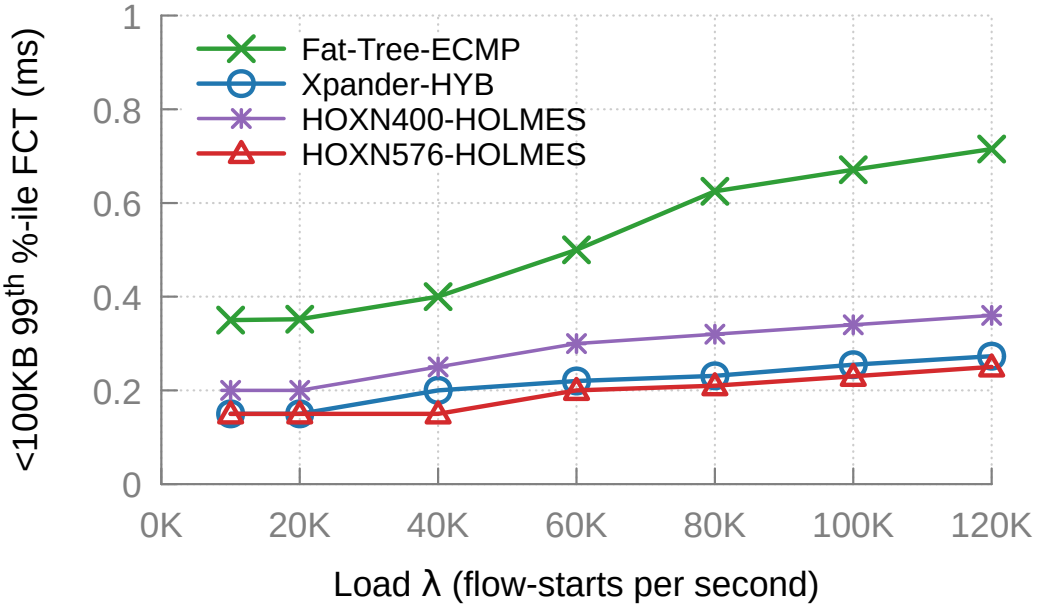


FIGURE 7.15: 99^{th}%-tile FCT for short flows w/ *Permute*(0.31) on the increasing aggregate flow arrival rate

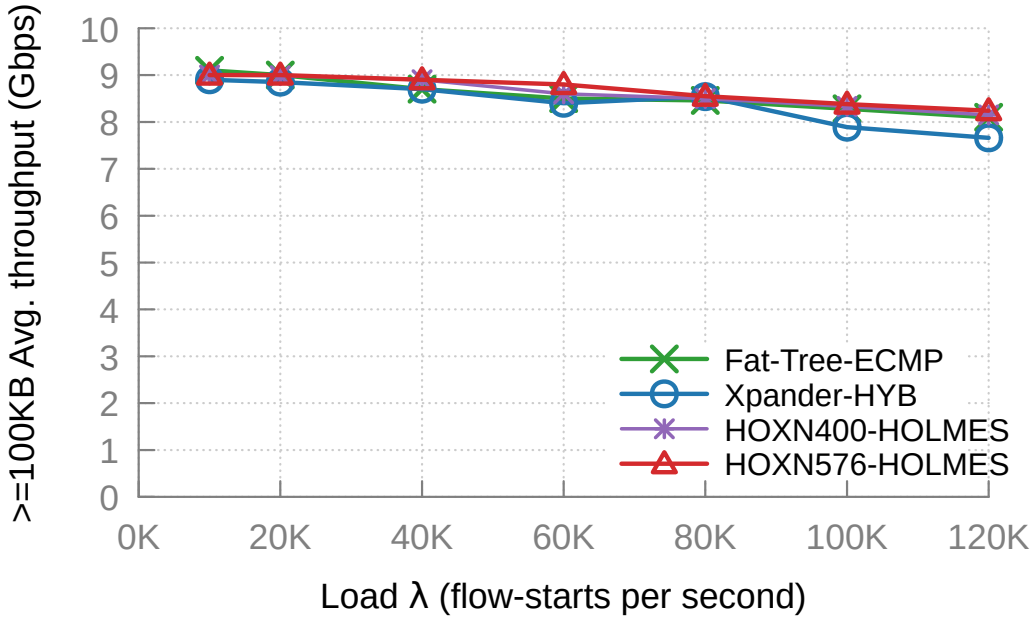


FIGURE 7.16: Average throughput for large flows w/ *Permute*(0.31) on the increasing aggregate flow arrival rate

7.1.2.3 ProjecTor Traffic Matrices

ProjecTor traffic is a highly skewed workload in DCNs. The two models are used as in [19]: one in which links between servers and switches have the same 10 Gbps capacity as all other switch-switch links, and another with unconstrained bandwidth for server-switch links. The latter model was also used in [79] to simulate an oversubscribed Fat-Tree with additional servers under each TOR.

The average FCT and 99^{th} percentile FCT for mouse flows are shown in Figures 7.17 and Figure 7.18 for the model with unconstrained capacity for server-switch links. The Xpander-HYB, HOXN400 and HOXN576 achieve performance gains over the Fat-Tree, while flow arrival rate increases. HOXN576 achieves up to 95% lower for both of the metrics, and HOXN400 matches the gain of Xpander-HYB over Fat-Tree-ECMP .

The results for links with a constrained capacity between servers and switches are shown in Figure 7.19. In this setting, Fat-Tree-ECMP with a full-bisection bandwidth topology performs a little better than Xpander-HYB, but all are matched closely.

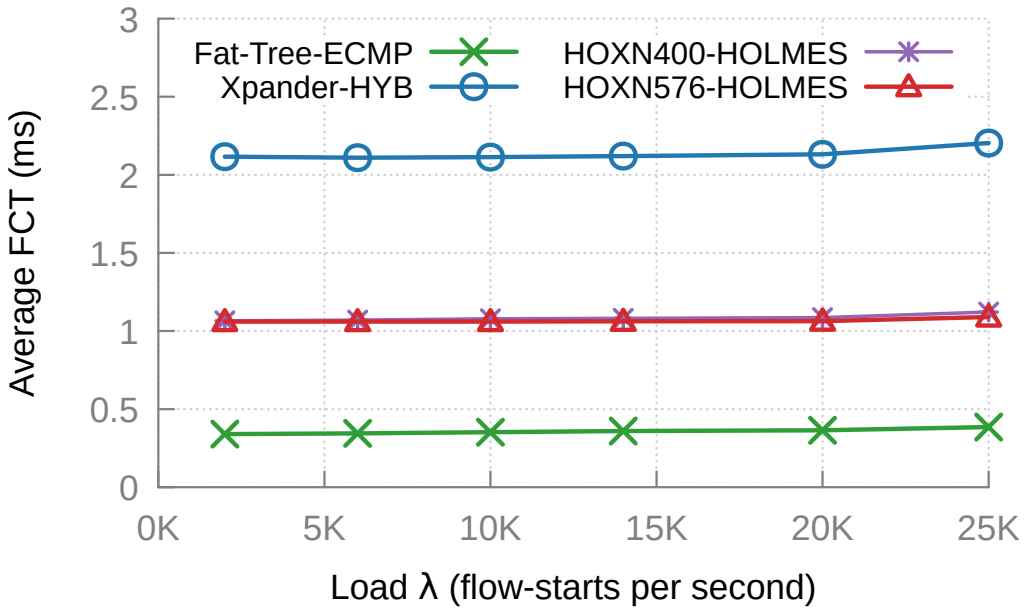


FIGURE 7.17: Average FCT w/ *ProjecTor* on the increasing aggregate flow arrival rate

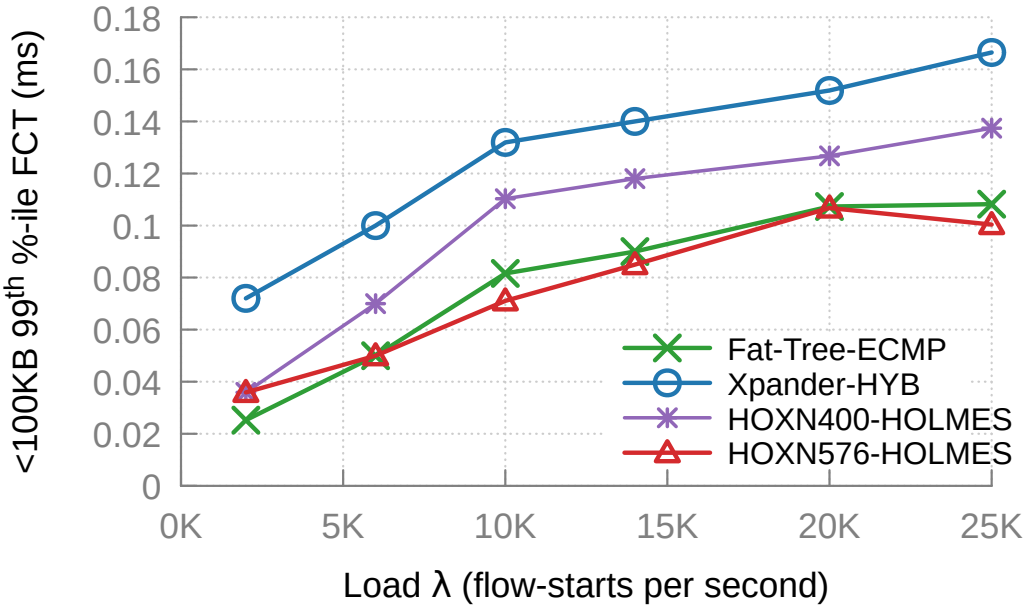


FIGURE 7.18: 99^{th}%-tile FCT for short flows w/ *ProjecTor* on the increasing aggregate flow arrival rate

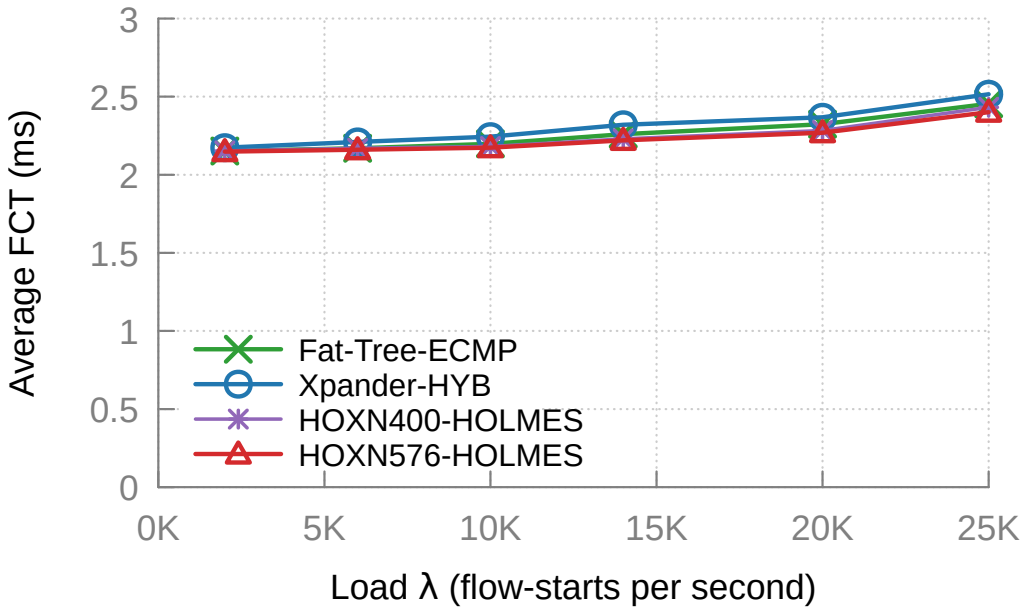


FIGURE 7.19: Average FCT w/ *ProjecTor* when server-level bottlenecks are modeled on the increasing aggregate flow arrival rate

7.2 Summary

The experiments have shown that the HyperOXN topology combined with HOLMES$^+$ (HOXN-HOLMES) is, in fact, able to handle these supposedly adverse traffic scenarios, rivaling the performance of the full-bisection bandwidth Fat-Tree at 15-30% of its total cost. Observe that even HOXN400 provides comparable or better performance than the Fat-Tree at 50% lower cost. In comparison, the results establish that HOXN-HOLMES can achieve reduced FCT for mouse flows, and high throughput for elephant flows that is better than Fat-Tree-ECMP and Xpander-HYB. The simulations show that the HOLMES$^+$ network partition policy can successfully eliminate the interference between the mouse and elephant data flows and improve the performance for FCT and throughput.

Chapter 8

Conclusions and Further Work

8.1 Summary

This thesis factors in the available direct topologies' advantages, and innovatively proposes a HyperOXN topology. Leveraging state-of-the-art silicon photonics technologies, HyperOXN is based on a hyperedge network to allow a single switch element to connect to multiple end-terminals while improving topology scalability. The numbers of channels may be doubled to make it rearrangeably non-blocking, which results in improving data flow propagation in the DCN, simplifying buffering and reducing system response time. Through direct-to-indirect topology transformation and using the mathematical induction method in Section 3.3 (page 78), the $2l$-dilated HyperOXN topology non-blocking property is proven. The aforementioned embedding property makes it favorable for parallel algorithm implementation and topology simulation. What is described in the thesis reflects the research work of the author to date. For future research, the author plans to continue improving HyperOXN topology performance by considering routing algorithms, load balancing and flow control operations. Detailed hardware implementation will also be introduced in future work.

A new stochastic TCP traffic model for traffic control based on traffic classification is introduced. As a traffic input to the HyperOXN architecture for simulations, one implements a ratio control method to separate the DC network usage.

This thesis also presents HOLMES$^+$, a novel DCN flow scheduling scheme, which tackles mixed (mice vs. elephants) Data Center traffic. Using a stochastic performance model, it proves the necessity of isolating mice and elephants with a closed form. Then it presents the HOLMES$^+$ architecture, which partitions a DCN into high-throughput and low-latency sub-networks. Further, the design of a stochastic and global congestion-aware load balancing algorithm that schedules the corresponding DC traffic to each sub-network will be developed. Initial simulation results show that the HOLMES$^+$ network partition policy can successfully eliminate interference between mouse and elephant data flows. Finally, it is proved that the HOLMES$^+$ flow scheduling algorithm is stable and scalable for large-scale Data Centers.

8.2 Future Work

Photonics technology provides excellent potential by exploiting the high bandwidth and energy efficiency. In future work, a hardware prototype needs to be considered for a topic of another future thesis. Energy optimization strategies and application-specific constraints will be studied in order to improve HyperOXN.

In addition, the design and evaluation of the AI module in the HOLMES$^+$ architecture in Figure 5.1 are left for another future topic.

References

[1] Cisco, "Cisco global cloud index: Forecast and methodology," 2016-2021. [Online]. Available: https://www.cisco.com/c/en/us/solutions/collateral/service-provider/global-cloud-index-gci/white-paper-c11-738085.html

[2] M. A. Taubenblatt, "Optical interconnects for high-performance computing," *Journal of Lightwave Technology*, vol. 30, no. 4, pp. 448–457, Feb 2012.

[3] W. Dally and B. Towles, *Principles and Practices of Interconnection Networks*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2003.

[4] A. Roy, H. Zeng, J. Bagga, G. Porter, and A. C. Snoeren, "Inside the social network's (datacenter) network," *SIGCOMM Comput. Commun. Rev.*, vol. 45, no. 4, pp. 123–137, Aug. 2015.

[5] A. Andreyev, "Introducing data center fabric, the next-generation facebook data center network," 14 November, 2014. [Online]. Available: https://code.facebook.com/posts/360346274145943/

[6] J. H. Ahn, N. Binkert, A. Davis, M. McLaren, and R. S. Schreiber, "Hyperx: topology, routing, and packaging of efficient large-scale networks," in *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*. ACM, 2009, p. 41.

[7] P. Costa, A. Donnelly, G. O'Shea, and A. Rowstron, "Camcubeos: A key-based network stack for 3d torus cluster topologies," in *Proceedings of*

the 22Nd International Symposium on High-performance Parallel and Distributed Computing, ser. HPDC '13. New York, NY, USA: ACM, 2013, pp. 73–84.

[8] G. Wang, D. G. Andersen, M. Kaminsky, K. Papagiannaki, T. E. Ng, M. Kozuch, and M. g. Ryan, "c-through: part-time optics in data centers," *SIGCOMM Comput. Commun. Rev.*, vol. 41, no. 4, pp. –, Aug. 2010.

[9] K. Chen, A. Singla, A. Singh, K. Ramachandran, L. Xu, Y. Zhang, X. Wen, and Y. Chen, "Osa: An optical switching architecture for data center networks with unprecedented flexibility," *IEEE/ACM Transactions on Networking*, vol. 22, no. 2, pp. 498–511, April 2014.

[10] P. N. Ji, D. Qian, K. Kanonakis, C. Kachris, and I. Tomkos, "Design and evaluation of a flexible-bandwidth ofdm-based intra-data center interconnect," *IEEE Journal of Selected Topics in Quantum Electronics*, vol. 19, no. 2, pp. 3 700 310–3 700 310, March 2013.

[11] R. Jain and K. K. Ramakrishnan, "Congestion avoidance in computer networks with a connectionless network layer: concepts, goals and methodology," in *[1988] Proceedings. Computer Networking Symposium*, 1988, pp. 134–143.

[12] T. T. Yuan, C. Xu, T. Huang, and J. Li, "Holmes : Holistic mice-elephants stochastic scheduling in data center networks," *Computer Science and Information Technology (CS & IT)*, vol. 8, no. 9, pp. 01 –21, June 2018.

[13] W. Ni, C. Huang, Y. L. Liu, W. Li, K. W. Leong, and J. Wu, "Poxn: A new passive optical cross-connection network for low-cost power-efficient datacenters," *Journal of Lightwave Technology*, vol. 32, no. 8, pp. 1482–1500, April 2014.

[14] D. Seyringer, *Arrayed Waveguide Gratings*. Spie Press, June 2016, vol. SL16.

[15] K. W. Leong, Z. Li, and Y. L. Liu, "Reliable multicast using remote direct memory access (rdma) over a passive optical cross-connect fabric enhanced with wavelength division multiplexing (wdm)," *APSIPA Transactions on Signal and Information Processing*, vol. 8, 2019.

[16] C. Katsinis and B. Nabet, "A scalable interconnection network architecture for petaflops computing," *J. Supercomput.*, vol. 27, no. 2, pp. 103–128, Feb. 2004.

[17] Z. Li, B. Li, D. Jiang, and L. C. Lau, "On achieving optimal throughput with network coding," in *Proceedings IEEE 24th Annual Joint Conference of the IEEE Computer and Communications Societies.*, vol. 3, March 2005, pp. 2184–2194 vol. 3.

[18] C. Xu, B. Tu, G. Li, T. Yuan, and J. Yang, "Dual channel adaptive congestion control for datacenters," in *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, 2019, pp. 514–521.

[19] S. Kassing, A. Valadarsky, G. Shahaf, M. Schapira, and A. Singla, "Beyond fat-trees without antennae, mirrors, and disco-balls," in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, ser. SIGCOMM '17. New York, NY, USA: ACM, 2017, pp. 281–294.

[20] C. Kachris, K. Kanonakis, and I. Tomkos, "Optical interconnection networks in data centers: recent trends and future challenges," *IEEE Communications Magazine*, vol. 51, no. 9, pp. 39–45, September 2013.

[21] L. Atzori, A. Iera, and G. Morabito, "The internet of things: A survey," *Comput. Netw.*, vol. 54, no. 15, pp. 2787–2805, Oct. 2010.

[22] Gartner, "Gartner says the internet of things installed base will grow to 26 billion units by 2020," 12 December, 2013. Retrieved 2 January 2014. [Online]. Available: http://www.gartner.com/newsroom/id/2636073

[23] Y.-F. Lu, J. Wu, and C.-F. Kuo, "A path generation scheme for real-time green internet of things," *SIGAPP Appl. Comput. Rev.*, vol. 14, no. 2, pp. 45–58, Jun. 2014.

[24] J. Kim, W. J. Dally, and D. Abts, "Flattened butterfly: A cost-efficient topology for high-radix networks," in *Proceedings of the 34th Annual International Symposium on Computer Architecture*, ser. ISCA '07. New York, NY, USA: ACM, 2007, pp. 126–137.

[25] C. Guo, H. Wu, K. Tan, L. Shi, Y. Zhang, and S. Lu, "Dcell: A scalable and fault-tolerant network structure for data centers," in *Proceedings of the ACM SIGCOMM 2008 Conference on Data Communication*, ser. SIGCOMM '08. New York, NY, USA: ACM, 2008, pp. 75–86.

[26] A. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. A. Maltz, P. Patel, and S. Sengupta, "Vl2: A scalable and flexible data center network," *SIGCOMM Comput. Commun. Rev.*, vol. 39, no. 4, pp. 51–62, Aug. 2009.

[27] R. Niranjan Mysore, A. Pamboris, N. Farrington, N. Huang, P. Miri, S. Radhakrishnan, V. Subramanya, and A. Vahdat, "Portland: A scalable fault-tolerant layer 2 data center network fabric," *SIGCOMM Comput. Commun. Rev.*, vol. 39, no. 4, pp. 39–50, Aug. 2009.

[28] C. E. Leiserson, "Fat-trees: Universal networks for hardware-efficient supercomputing," *IEEE Transactions on Computers*, vol. C-34, no. 10, pp. 892–901, Oct 1985.

[29] D. Li, J. Wu, Z. Liu, and F. Zhang, "Towards the tradeoffs in designing data center network architectures," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 1, pp. 260–273, Jan 2017.

[30] L. N. Bhuyan and D. P. Agrawal, "Generalized hypercube and hyperbus structures for a computer network," *IEEE Trans. Comput.*, vol. 33, no. 4, pp. 323–333, Apr. 1984.

[31] A. Thamarakuzhi and J. A. Chandy, "2-dilated flattened butterfly: A non-blocking switching topology for high-radix networks," *Comput. Commun.*, vol. 34, no. 15, pp. 1822–1835, Sep. 2011.

[32] S. Z. Zhonghua Zhu and L. Chen, "Fully programmable and scalable optical switching fabric for petabyte data center," *Optics Express*, Feb 2015.

[33] B. Speitkamp and M. Bichler, "A mathematical programming approach for server consolidation problems in virtualized data centers," *IEEE Transactions on Services Computing*, vol. 3, no. 4, pp. 266–278, Oct 2010.

[34] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan, "Data center tcp (dctcp)," in *Proceedings of the ACM SIGCOMM 2010 Conference*, ser. SIGCOMM '10. New York, NY, USA: ACM, 2010, pp. 63–74.

[35] T. Benson, A. Akella, and D. A. Maltz, "Network traffic characteristics of data centers in the wild," in *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*. ACM, 2010, pp. 267–280.

[36] A. Shaikh, J. Rexford, and K. G. Shin, "Load-sensitive routing of long-lived ip flows," *SIGCOMM Comput. Commun. Rev.*, vol. 29, no. 4, pp. 215–226, Aug. 1999.

[37] T. Benson, A. Anand, A. Akella, and M. Zhang, "Understanding data center traffic characteristics," *ACM SIGCOMM Computer Communication Review*, vol. 40, no. 1, pp. 92–99, 2010.

[38] A. Roy, H. Zeng, J. Bagga, G. Porter, and A. C. Snoeren, "Inside the social network's (datacenter) network," *SIGCOMM Comput. Commun. Rev.*, vol. 45, no. 4, pp. 123–137, Aug. 2015.

[39] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," in *Proceedings of the ACM SIGCOMM 2008 Conference on Data Communication*, ser. SIGCOMM '08. New York, NY, USA: ACM, 2008, pp. 63–74.

[40] H. Wang, Y. Xia, K. Bergman, T. S. E. Ng, S. Sahu, and K. Sripanidkulchai, "Rethinking the physical layer of data center networks of the next decade: using optics to enable efficient *-cast connectivity," *Computer Communication Review*, vol. 43, pp. 52–58, 2013.

[41] T. T. Yuan, C. Xu, T. Huang, and B. Fleischer, "An all-cloud approach to iot applications and its network infrastructure optimization," *Design Automation Conference*, 2017.

[42] T. T. Yuan, C. Xu, T. Huang, and J. Li, "Holmes: Holistic mice-elephants stochastic scheduling in data center networks," *Computer Science & Information Technology*, vol. 8, no. 9, pp. 01–21, June 2018.

[43] M. Alizadeh, T. Edsall, S. Dharmapurikar, K. Vaidyanathan, Ramanan and Chu, A. Fingerhut, V. T. Lam, F. Ma-tus, R. Pan, N. Yadav, and G. Varghese, "Conga: Distributed congestion-aware load balancing for datacenters," *SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 4, pp. 503–514, Aug. 2014.

[44] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat, "Hedera: Dynamic flow scheduling for data center networks." in *NSDI*, vol. 10, 2010, pp. 19–19.

[45] U.-D. R. Chris Snijders, Uwe Matzat, "Big data: Big gaps of knowledge in the field of internet science," *International Journal of Internet Science*, vol. 7, no. 1, pp. 1–5, Jan. 2012.

[46] X. Qiao, H. Wang, W. Tan, A. V. Vasilakos, J. Chen, and M. B. Blake, "A survey of applications research on content-centric networking," *China Communications*, vol. 16, no. 9, pp. 122–140, Sep. 2019.

[47] J. Dean and et al., "Mapreduce: Simplified data processing on large clusters," 2004.

[48] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: Cluster computing with working sets," 2010.

[49] S. Chintapalli, D. Dagit, B. Evans, R. Farivar, T. Graves, M. Holderbaugh, Z. Liu, K. Nusbaum, K. Patil, B. J. Peng, and P. Poulosky, "Benchmarking streaming computation engines: Storm, flink and spark streaming," in *2016 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, May 2016, pp. 1789–1792.

[50] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly, "Dryad: Distributed data-parallel programs from sequential building blocks," in *Proceedings of the 2007 Eurosys Conference*. Lisbon, Portugal: Association for Computing Machinery, Inc., March 2007.

[51] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016. [Online]. Available: http://www.deeplearningbook.org

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

[53] M. García, E. Vallejo, R. Beivide, M. Odriozola, and M. Valero, "Efficient routing mechanisms for dragonfly networks," in *2013 42nd International Conference on Parallel Processing*. IEEE, 2013, pp. 582–592.

[54] G. Linden, "Geeking with greg," 04 December, 2006. [Online]. Available: http://glinden.blogspot.com/2006/12/slides-from-my-talk-at-stanford.html

[55] G. LINDEN, "Geeking with greg," 09 November, 2006. [Online]. Available: http://glinden.blogspot.com/2006/11/marissa-mayer-at-web-20.html

[56] T. Group, "The value of a millisecond," April, 2008. [Online]. Available: https://community.rti.com/sites/default/files/archive/V06-007_Value_of_a_Millisecond.pdf

[57] A. F. Benner, "Cost-effective optics: Enabling the exascale roadmap," in *2009 17th IEEE Symposium on High Performance Interconnects*, Aug 2009, pp. 133–137.

[58] J. Hamilton, "Overall data center costs," 2010. [Online]. Available: https://perspectives.mvdirona.com/2010/09/overall-data-center-costs/

[59] M.-C. Heydemann, *Cayley graphs and interconnection networks.* Dordrecht: Springer Netherlands, 1997, pp. 167–224.

[60] D. P. Agrawal, C. Chen, and J. R. Burke, "Hybrid graph-based networks for multiprocessing," *Telecommun. Syst.*, vol. 10, no. 1-2, pp. 107–134, Oct. 1998.

[61] E. Kowalski, "An introduction to expander graphs," 25 November, 2017. [Online]. Available: https://people.math.ethz.ch/~kowalski/expander-graphs.pdf

[62] S. R. Ohring, M. Ibel, S. K. Das, and M. J. Kumar, "On generalized fat trees," in *Proceedings of 9th International Parallel Processing Symposium*, April 1995, pp. 37–44.

[63] C. Guo, G. Lu, D. Li, H. Wu, X. Zhang, Y. Shi, C. Tian, Y. Zhang, and S. Lu, "Bcube: A high performance, server-centric network architecture for modular data centers," in *Proceedings of the ACM SIGCOMM 2009 Conference on Data Communication*, ser. SIGCOMM '09. New York, NY, USA: ACM, 2009, pp. 63–74.

[64] H. Wu, G. Lu, D. Li, C. Guo, and Y. Zhang, "Mdcube: A high performance network structure for modular data center interconnection," in *ACM SIGCOMM CoNEXT*, December 2009.

[65] R. V. Tomic, "Optimal networks from error correcting codes," in *Architectures for Networking and Communications Systems*, Oct 2013, pp. 169–179.

[66] A. Valadarsky, G. Shahaf, M. Dinitz, and M. Schapira, "Xpander: Towards optimal-performance datacenters," in *CoNEXT*, 2016.

[67] M. Besta and T. Hoefler, "Slim fly: A cost effective low-diameter network topology," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '14. Piscataway, NJ, USA: IEEE Press, 2014, pp. 348–359.

[68] A. Singla, C.-Y. Hong, L. Popa, and P. B. Godfrey, "Jellyfish: Networking data centers randomly," in *Presented as part of the 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12)*. San Jose, CA: USENIX, 2012, pp. 225–238.

[69] T. W. Judson, "Abstract algebra: Theory and applications," 2018. [Online]. Available: http://www.gurobi.com

[70] Y. Yang and G. M. Masson, "The necessary conditions for clos-type nonblocking multicast networks," in *Proceedings of International Conference on Parallel Processing*, Apr 1996, pp. 789–795.

[71] K. Bilal, S. U. R. Malik, O. Khalid, A. Hameed, E. Alvarez, V. Wijaysekara, R. Irfan, S. Shrestha, D. Dwivedy, M. Ali *et al.*, "A taxonomy and survey on green data center networks," *Future Generation Computer Systems*, vol. 36, pp. 189–208, 2014.

[72] M. G. M. and J. B. W., "Generalized multistage connection networks," *Networks*, vol. 2, no. 3, pp. 191–209, 1972.

[73] J. Networks, "The qfabric architecture," 2013. [Online]. Available: www.enpointe.com/images/pdf/The-Qfabric-Architecture.pdf

[74] Z. Liu and D. W. Cheung, "Oblivious routing for lc permutations on hypercubes," *Parallel Computing*, vol. 25, no. 4, pp. 445–460, Apr 1999.

[75] J. F. Martínez, J. Torrellas, and J. Duato, "Improving the performance of bristled cc-numa systems using virtual channels and adaptivity," in *Proceedings of the 13th international conference on Supercomputing*. ACM, 1999, pp. 202–209.

[76] T. Szymanski, "" hypermeshes": Optical interconnection networks for parallel computing," *Journal of Parallel and Distributed Computing*, vol. 26, no. 1, pp. 1–23, 1995.

[77] L. N. Bhuyan and D. P. Agrawal, "Generalized hypercube and hyperbus structures for a computer network," *IEEE Transactions on computers*, vol. 100, no. 4, pp. 323–333, 1984.

[78] D. Guo, C. Li, J. Wu, and X. Zhou, "Dcube: A family of network structures for containerized data centers using dual-port servers," *Computer Communications*, vol. 53, pp. 13 – 25, 2014.

[79] M. Ghobadi, R. Mahajan, A. Phanishayee, N. Devanur, J. Kulkarni, G. Ranade, P.-A. Blanche, H. Rastegarfar, M. Glick, and D. Kilper, "Projector: Agile reconfigurable data center interconnect," in *Proceedings of the 2016 ACM SIGCOMM Conference*, ser. SIGCOMM '16. New York, NY, USA: ACM, 2016, pp. 216–229.

[80] N. Farrington, G. Porter, S. Radhakrishnan, H. H. Bazzaz, V. Subramanya, Y. Fainman, G. Papen, and A. Vahdat, "Helios: A hybrid electrical/optical switch architecture for modular data centers," in *Proceedings of the ACM SIGCOMM 2010 Conference*, ser. SIGCOMM '10. New York, NY, USA: ACM, 2010, pp. 339–350.

[81] W. M. Mellette, R. McGuinness, A. Roy, A. Forencich, G. Papen, A. C. Snoeren, and G. Porter, "Rotornet: A scalable, low-complexity, optical datacenter network," in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, ser. SIGCOMM '17. New York, NY, USA: ACM, 2017, pp. 267–280.

[82] J. Y. Shin, E. G. Sirer, H. Weatherspoon, and D. Kirovski, "On the feasibility of completely wireless datacenters," in *2012 ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS)*, Oct 2012, pp. 3–14.

[83] D. Halperin, S. Kandula, J. Padhye, P. Bahl, and D. Wetherall, "Augmenting data center networks with multi-gigabit wireless links," *SIGCOMM Comput. Commun. Rev.*, vol. 41, no. 4, pp. 38–49, Aug. 2011.

[84] H. Hammad, "Cisco unified fabric," 2010. [Online]. Available: https://www.cisco.com/c/dam/global/en_dz/assets/expoegypt2010/assets/docs/cisco_unified_fabric_hatem_hammad.pdf

[85] E. Walker, "Benchmarking amazon ec2 for high-performance scientific computing," 2008. [Online]. Available: https://www.usenix.org/legacy/publications/login/2008-10/openpdfs/walker.pdf

[86] TOP500. [Online]. Available: http://www.top500.org/

[87] H. CE, "Analysis of an equal-cost multipath algorithm," *RFC*, 01 2000.

[88] J. Cao, R. Xia, P. Yang, C. Guo, G. Lu, L. Yuan, Y. Zheng, H. Wu, Y. Xiong, and D. Maltz, "Per-packet load-balanced, low-latency routing for clos-based data center networks," in *Proceedings of the ninth ACM conference on Emerging networking experiments and technologies*. ACM, 2013, pp. 49–60.

[89] A. Dixit and el atl., "On the impact of packet spraying in data center networks," in *In Proc. of IEEE INFOCOM 2013*, 2013.

[90] M. Alizadeh, S. Yang, M. Sharif, S. Katti, N. McKeown, B. Prabhakar, and S. Shenker, "pfabric: Minimal near-optimal datacenter transport," in *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*, ser. SIGCOMM '13. New York, NY, USA: ACM, 2013, pp. 435–446.

[91] D. Zats, T. Das, P. Mohan, D. Borthakur, and R. Katz, "Detail: reducing the flow completion time tail in datacenter networks," *ACM SIGCOMM Computer Communication Review*, vol. 42, no. 4, pp. 139–150, 2012.

[92] J. Zhou, M. Tewari, M. Zhu, A. Kabbani, L. Poutievski, A. Singh, and A. Vahdat, "Wcmp: Weighted cost multipathing for improved fairness in

data centers," in *Proceedings of the Ninth European Conference on Computer Systems*. ACM, 2014, p. 5.

[93] P. Wang and H. Xu, "Expeditus: Distributed load balancing with global congestion information in data center networks," in *Proceedings of the 2014 CoNEXT on Student Workshop*. ACM, 2014, pp. 1–3.

[94] S. Ghorbani, B. Godfrey, Y. Ganjali, and A. Firoozshahian, "Micro load balancing in data centers with drill," in *Proceedings of the 14th ACM Workshop on Hot Topics in Networks*. ACM, 2015, p. 17.

[95] N. Katta, M. Hira, C. Kim, A. Sivaraman, and J. Rexford, "Hula: Scalable load balancing using programmable data planes," in *Proc. ACM Symposium on SDN Research*, 2016.

[96] R. Mittal, V. T. Lam, N. Dukkipati, E. Blem, H. Wassel, M. Ghobadi, A. Vahdat, Y. Wang, D. Wetherall, and D. Zats, "Timely: Rtt-based congestion control for the datacenter," *SIGCOMM Comput. Commun. Rev.*, vol. 45, no. 4, pp. 537–550, Aug. 2015.

[97] D. A. Hayes and G. Armitage, "Revisiting tcp congestion control using delay gradients," in *Proceedings of the 10th International IFIP TC 6 Conference on Networking - Volume Part II*, ser. NETWORKING'11. Berlin, Heidelberg: Springer-Verlag, 2011, pp. 328–341.

[98] W. Wang, Y. Sun, K. Salamatian, and Z. Li, "Adaptive path isolation for elephant and mice flows by exploiting path diversity in datacenters," *IEEE Transactions on Network and Service Management*, vol. 13, no. 1, pp. 5–18, 2016.

[99] W. Cheng, R. Ren, W. Jiang, K. Qian, T. Zhang, and R. Shu, "Isolating mice and elephant in data centers," *arXiv preprint arXiv:1605.07732*, 2016.

[100] C. Zhang and V. Tsaoussidis, "Tcp-real: Improving real-time capabilities of tcp over heterogeneous networks," in *Proceedings of the 11th International*

Workshop on Network and Operating Systems Support for Digital Audio and Video, ser. NOSSDAV '01. New York, NY, USA: ACM, 2001, pp. 189–198.

[101] C. Raiciu, M. J. Handley, and D. Wischik, "Coupled Congestion Control for Multipath Transport Protocols," RFC 6356, Oct. 2011. [Online]. Available: https://rfc-editor.org/rfc/rfc6356.txt

[102] Y. Zhu, H. Eran, D. Firestone, C. Guo, M. Lipshteyn, Y. Liron, J. Padhye, S. Raindel, M. H. Yahia, and M. Zhang, "Congestion control for large-scale rdma deployments," *SIGCOMM Comput. Commun. Rev.*, vol. 45, no. 4, pp. 523–536, Aug. 2015.

[103] D. Ionescu, Z. Shi, and D. Zhang, "Packet loss control using tokens at the network edge," *IEEE Latin America Transactions*, vol. 10, no. 1, pp. 1391–1393, Jan 2012.

[104] N. Cardwell, Y. Cheng, C. S. Gunn, S. H. Yeganeh, and V. Jacobson, "Bbr: Congestion-based congestion control," *Queue*, vol. 14, no. 5, pp. 50:20–50:53, Oct. 2016.

[105] K. Tan, J. Song, Q. Zhang, and M. Sridharan, "A compound tcp approach for high-speed and long distance networks," in *Proceedings IEEE INFOCOM 2006. 25TH IEEE International Conference on Computer Communications*, April 2006, pp. 1–12.

[106] D. X. Wei, C. Jin, S. H. Low, and S. Hegde, "Fast tcp: Motivation, architecture, algorithms, performance," *IEEE/ACM Transactions on Networking*, vol. 14, no. 6, pp. 1246–1259, Dec 2006.

[107] W.-C. Feng, D. D. Kandlur, D. Saha, and K. G. Shin, "Techniques for eliminating packet loss in congested tcp/ip networks," 1997.

[108] W. chang Feng, K. G. Shin, D. D. Kandlur, and D. Saha, "The blue active queue management algorithms," *IEEE/ACM Transactions on Networking*, vol. 10, no. 4, pp. 513–528, Aug 2002.

[109] C. V. Hollot, V. Misra, D. Towsley, and W.-B. Gong, "On designing improved controllers for aqm routers supporting tcp flows," in *Proceedings IEEE INFOCOM 2001. Conference on Computer Communications. Twentieth Annual Joint Conference of the IEEE Computer and Communications Society (Cat. No.01CH37213)*, vol. 3, 2001, pp. 1726–1734 vol.3.

[110] V. Firoiu and M. Borden, "A study of active queue management for congestion control," in *Proceedings IEEE INFOCOM 2000. Conference on Computer Communications. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies (Cat. No.00CH37064)*, vol. 3, Mar 2000, pp. 1435–1444 vol.3.

[111] C.-Y. Hong, S. Kandula, R. Mahajan, M. Zhang, V. Gill, M. Nanduri, and R. Wattenhofer, "Achieving high utilization with software-driven wan," in *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*, ser. SIGCOMM '13. New York, NY, USA: ACM, 2013, pp. 15–26.

[112] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu, J. Zolla, U. Hölzle, S. Stuart, and A. Vahdat, "B4: Experience with a globally-deployed software defined wan," in *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*, ser. SIGCOMM '13. New York, NY, USA: ACM, 2013, pp. 3–14.

[113] C. Raiciu, S. Barre, C. Pluntke, A. Greenhalgh, D. Wischik, and M. Handley, "Improving datacenter performance and robustness with multipath tcp," in *Proceedings of the ACM SIGCOMM 2011 Conference*, ser. SIGCOMM '11. New York, NY, USA: ACM, 2011, pp. 266–277.

[114] T. Edsall, A. Fingerhut, T. Lam, R. Pan, and G. Varghese, *Flame: Efficient and Robust Hardware Load Balancing for Data Center Routers.* [Department of Computer Science and Engineering], University of California, San Diego, 2012.

[115] A. Dixit, P. Prakash, and R. R. Kompella, "On the efficacy of fine-grained traffic splitting protocolsin data center networks," in *Proceedings of the ACM*

SIGCOMM 2011 Conference, ser. SIGCOMM '11. New York, NY, USA: ACM, 2011, pp. 430–431.

[116] K. He, E. Rozner, K. Agarwal, W. Felter, J. Carter, and A. Akella, "Presto: Edge-based load balancing for fast datacenter networks," *ACM SIGCOMM Computer Communication Review*, vol. 45, no. 4, pp. 465–478, 2015.

[117] L. G. Valiant and G. J. Brebner, "Universal schemes for parallel communication," in *Proceedings of the Thirteenth Annual ACM Symposium on Theory of Computing*, ser. STOC '81. New York, NY, USA: ACM, 1981, pp. 263–277.

[118] R. Zhang-Shen and N. McKeown, "Designing a fault-tolerant network using valiant load-balancing," in *IEEE INFOCOM 2008 - The 27th Conference on Computer Communications*, April 2008.

[119] S. Kandula, D. Katabi, B. Davie, and A. Charny, "Walking the Tightrope: Responsive Yet Stable Traffic Engineering," in *ACM SIGCOMM*, Philadelphia, PA, August 2005.

[120] S. Liu, W. Bai, H. Xu, K. Chen, and Z. Cai, "Repnet: Cutting tail latency in data center networks with flow replication," *arXiv preprint arXiv:1407.1239*, 2014.

[121] R. Gallager, "A minimum delay routing algorithm using distributed computation," *IEEE Transactions on Communications*, vol. 25, no. 1, pp. 73–85, January 1977.

[122] N. Michael and A. Tang, "Halo: Hop-by-hop adaptive link-state optimal routing," *IEEE/ACM Transactions on Networking*, vol. 23, no. 6, pp. 1862–1875, Dec 2015.

[123] S. Sen, D. Shue, S. Ihm, and M. J. Freedman, "Scalable, optimal flow routing in datacenters via local link balancing," in *Proceedings of the ninth ACM conference on Emerging networking experiments and technologies*. ACM, 2013, pp. 151–162.

[124] S. Kandula and et al., "Dynamic load balancing without packet reordering," *ACM SIGCOMM Computer Communication Review*, vol. 37, no. 2, 2007.

[125] M. Mitzenmacher, "The power of two choices in randomized load balancing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 12, no. 10, pp. 1094–1104, 2001.

[126] Y.-T. He and D. G. Down, "Limited choice and locality considerations for load balancing," *Perform. Eval.*, vol. 65, pp. 670–687, 2008.

[127] F. P. Kelly, A. Maulloo, and D. Tan, "Rate control in communication networks: Shadow prices, proportional fairness and stability," *Journal of the Operational Research Society*, vol. 49, pp. 237–252, 1998.

[128] W. Wang, Y. Sun, K. Zheng, M. A. Kaafar, D. Li, and Z. Li, "Freeway: Adaptively isolating the elephant and mice flows on different transmission paths," in *2014 IEEE 22nd International Conference on Network Protocols*. IEEE, 2014, pp. 362–367.

[129] A. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, and S. Banerjee, "Devoflow: Scaling flow management for high-performance networks," in *Proceedings of the ACM SIGCOMM 2011 Conference*, ser. SIGCOMM '11. New York, NY, USA: ACM, 2011, pp. 254–265.

[130] Y. Wang, D. Lin, C. Li, J. Zhang, P. Liu, C. Hu, and G. Zhang, "Application driven network: Providing on-demand services for applications," in *Proceedings of the 2016 ACM SIGCOMM Conference*, ser. SIGCOMM '16. New York, NY, USA: ACM, 2016, pp. 617–618.

[131] L. Chen, K. Chen, Z. Zhu, M. Yu, G. Porter, C. Qiao, and S. Zhong, "Enabling wide-spread communications on optical fabric with megaswitch," in *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*. Boston, MA: USENIX Association, 2017, pp. 577–593.

[132] K. Hasharoni, S. Benjamin, A. Geron, G. Katz, S. Stepanov, N. Margalit, and M. Mesh, "A 1.34 tb/s full duplex optical interconnect for optical

backplane applications," in *2012 International Conference on Photonics in Switching (PS)*, Sept 2012, pp. 1–3.

[133] Z. Zhu, S. Zhong, L. Chen, and K. Chen, "Fully programmable and scalable optical switching fabric for petabyte data center," *Optics express*, vol. 23, no. 3, pp. 3563–3580, 2015.

[134] R. Hemenway, R. Grzybowski, C. Minkenberg, and R. Luijten, "Optical-packet-switched interconnect for supercomputer applications," *J. Opt. Network*, vol. 3, no. 12, pp. 900–913, Dec 2004.

[135] D. V. Plant, M. B. Venditti, E. Laprise, J. Faucher, M. Chateauneuf, and J. S. Ahearn, "256-channel bidirectional optical interconnect using vcsels and photodiodes on cmos," *Journal of Lightwave Technology*, vol. 19, no. 8, pp. 1093–1103, Aug 2001.

[136] D. Abts, M. R. Marty, P. M. Wells, P. Klausler, and H. Liu, "Energy proportional datacenter networks," in *Proceedings of the 37th Annual International Symposium on Computer Architecture*, ser. ISCA '10. New York, NY, USA: ACM, 2010, pp. 338–347.

[137] C. A. Thraskias, E. N. Lallas, N. Neumann, L. Schares, B. J. Offrein, R. Henker, D. Plettemeier, F. Ellinger, J. Leuthold, and I. Tomkos, "Survey of photonic and plasmonic interconnect technologies for intra-datacenter and high-performance computing communications," *IEEE Communications Surveys Tutorials*, pp. 1–1, 2018.

[138] Colfaxdirect, "Colfaxdirect," 2018. [Online]. Available: http://www.colfaxdirect.com/

[139] V. Stojanović, R. J. Ram, M. Popović, S. Lin, S. Moazeni, M. Wade, C. Sun, L. Alloatti, A. Atabaki, F. Pavanello, N. Mehta, and P. Bhargava, "Monolithic silicon-photonic platforms in state-of-the-art cmos soi processes," *Opt. Express*, vol. 26, no. 10, pp. 13 106–13 121, May 2018.

[140] H. Li, Z. Xuan, A. Titriku, C. Li, K. Yu, B. Wang, A. Shafik, N. Qi, Y. Liu, R. Ding, T. Baehr-Jones, M. Fiorentino, M. Hochberg, S. Palermo, and P. Y. Chiang, "A 25 gb/s, 4.4 v-swing, ac-coupled ring modulator-based wdm transmitter with wavelength stabilization in 65 nm cmos," *IEEE Journal of Solid-State Circuits*, vol. 50, no. 12, pp. 3145–3159, Dec 2015.

[141] M. Pantouvaki, P. D. Heyn, M. Rakowski, P. Verheyen, B. Snyder, S. A. Srinivasan, H. Chen, J. D. Coster, G. Lepage, P. Absil, and J. V. Campenhout, "50gb/s silicon photonics platform for short-reach optical interconnects," in *Optical Fiber Communication Conference*. Optical Society of America, 2016, p. Th4H.4.

[142] K. Yu, C. Li, H. Li, A. Titriku, A. Shafik, B. Wang, Z. Wang, R. Bai, C. Chen, M. Fiorentino, P. Y. Chiang, and S. Palermo, "A 25 gb/s hybrid-integrated silicon photonic source-synchronous receiver with microring wavelength stabilization," *IEEE Journal of Solid-State Circuits*, vol. 51, no. 9, pp. 2129–2141, Sept 2016.

[143] N. B. Feilchenfeld, F. G. Anderson, T. Barwicz, S. Chilstedt, Y. Ding, J. Ellis-Monaghan, D. M. Gill, C. Hedges, J. Hofrichter, F. Horst, M. Khater, E. Kiewra, R. Leidy, Y. Martin, K. McLean, M. Nicewicz, J. S. Orcutt, B. Porth, J. Proesel, C. Reinholm, J. C. Rosenberg, W. D. Sacher, A. D. Stricker, C. Whiting, C. Xiong, A. Agrawal, F. Baker, C. W. Baks, B. Cucci, D. Dang, T. Doan, F. Doany, S. Engelmann, M. Gordon, E. Joseph, J. Maling, S. Shank, X. Tian, C. Willets, J. Ferrario, M. Meghelli, F. Libsch, B. Offrein, W. M. J. Green, and W. Haensch, "An integrated silicon photonics technology for o-band datacom," in *2015 IEEE International Electron Devices Meeting (IEDM)*, Dec 2015, pp. 25.7.1–25.7.4.

[144] F. Boeuf, S. Crmer, N. Vulliet, T. Pinguet, A. Mekis, G. Masini, L. Verslegers, P. Sun, A. Ayazi, N. . Hon, S. Sahni, Y. Chi, B. Orlando, D. Ristoiu, A. Farcy, F. Leverd, L. Broussous, D. Pelissier-Tanon, C. Richard, L. Pinzelli, R. Beneyton, O. Gourhant, E. Gourvest, Y. Le-Friec, D. Monnier, P. Brun, M. Guillermet, D. Benoit, K. Haxaire, J. R. Manouvrier, S. Jan,

H. Petiton, J. F. Carpentier, T. Qumerais, C. Durand, D. Gloria, M. Fourel, F. Battegay, Y. Sanchez, E. Batail, F. Baron, P. Delpech, L. Salager, P. D. Dobbelaere, and B. Sautreuil, "A multi-wavelength 3d-compatible silicon photonics platform on 300mm soi wafers for 25gb/s applications," in *2013 IEEE International Electron Devices Meeting*, Dec 2013, pp. 13.3.1–13.3.4.

[145] A. Jajszczyk, "Nonblocking, repackable, and rearrangeable clos networks: fifty years of the theory evolution," *IEEE Communications Magazine*, vol. 41, no. 10, pp. 28–33, Oct 2003.

[146] S. B. Choi and A. K. Somani, "Rearrangeable circuit-switched hypercube architectures for routing permutations," *Journal of Parallel and Distributed Computing*, vol. 19, no. 2, pp. 125–130, Oct 1993.

[147] K. Y. LEE, "On the rearrangeability of 2(log2n)-1 stage permutation networks," *IEEE Transactions on Computers*, vol. c-34, no. 5, pp. 412–425, May 1985.

[148] S. B. Choi and A. K. Somani, "The generalized folding-cube network," *Networks*, vol. 21, no. 3, pp. 267–294, May 1991.

[149] F. T. Leighton, *Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes.* Morgan Kaufmann Pub, Sep 1991.

[150] S.-Y. Kim and K.-Y. Chwa, "Optimal embeddings of multiple graphs into a hypermesh," in *International Conference on Parallel and Distributed Systems.* IEEE, Dec 1997, pp. 436–443.

[151] L. Popa, S. Ratnasamy, G. Iannaccone, A. Krishnamurthy, and I. Stoica, "A cost comparison of datacenter network architectures," in *Proceedings of the 6th International COnference*, ser. Co-NEXT '10, New York, NY, USA, 2010, pp. 16:1–16:12.

[152] D. Abts and J. Kim, *High Performance Datacenter Networks: Architectures, Algorithms, and Opportunities.* Morgan & Claypool, 2011.

[153] F. Petrini and M. Vanneschi, "k-ary n-trees: high performance networks for massively parallel architectures," in *Proceedings 11th International Parallel Processing Symposium*, Apr 1997, pp. 87–93.

[154] R. Ahlswede, N. Cai, S. Y. R. Li, and R. W. Yeung, "Network information flow," *IEEE Transactions on Information Theory*, vol. 46, no. 4, pp. 1204–1216, Jul 2000.

[155] K. Jain, M. Mahdian, and M. R. Salavatipour, "Packing steiner trees," in *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, ser. SODA '03. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 2003, pp. 266–274.

[156] X. Li and M. Freedman, "Scaling ip multicast on datacenter topologies," in *CoNEXT 2013 - Proceedings of the 2013 ACM International Conference on Emerging Networking Experiments and Technologies*, ser. CoNEXT 2013 - Proceedings of the 2013 ACM International Conference on Emerging Networking Experiments and Technologies. Association for Computing Machinery, 1 2013, pp. 61–72.

[157] G. O. Inc., "Gurobi optimizer reference manual," 2018. [Online]. Available: http://www.gurobi.com

[158] S. Kandula, S. Sengupta, A. Greenberg, P. Patel, and R. Chaiken, "The nature of data center traffic: measurements & analysis," in *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement conference*. ACM, 2009, pp. 202–208.

[159] B. Atikoglu, Y. Xu, E. Frachtenberg, S. Jiang, and M. Paleczny, "Workload analysis of a large-scale key-value store," *SIGMETRICS Perform. Eval. Rev.*, vol. 40, no. 1, pp. 53–64, Jun. 2012.

[160] J. Gettys and K. Nichols, "Bufferbloat: Dark buffers in the internet," *Queue*, vol. 9, no. 11, pp. 40:40–40:54, Nov. 2011.

[161] J. Padhye, V. Firoiu, D. F. Towsley, and J. F. Kurose, "Modeling tcp reno performance: a simple model and its empirical validation," *IEEE/ACM Transactions on Networking*, vol. 8, no. 2, pp. 133–145, Apr 2000.

[162] D. G. Kendall, "On the generalized "birth-and-death" process," *Ann. Math. Statist.*, vol. 19, no. 1, pp. 1–15, 03 1948.

[163] R. M. Dudley, *Uniform Central Limit Theorems*, ser. Cambridge Studies in Advanced Mathematics. Cambridge University Press, 1999.

[164] W. MathWorld, "Normal distribution." [Online]. Available: http://mathworld.wolfram.com/NormalDistribution.html

[165] L. Bernaille, R. Teixeira, I. Akodkenou, A. Soule, and K. Salamatian, "Traffic classification on the fly," *SIGCOMM Comput. Commun. Rev.*, vol. 36, no. 2, pp. 23–26, Apr. 2006.

[166] V. Jalaparti, P. Bodik, I. Menache, S. Rao, K. Makarychev, and M. Caesar, "Network-aware scheduling for data-parallel jobs: Plan when you can," in *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, ser. SIGCOMM '15. New York, NY, USA: ACM, 2015, pp. 407–420.

[167] M. Chowdhury and I. Stoica, "Efficient coflow scheduling without prior knowledge," in *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, ser. SIGCOMM '15. New York, NY, USA: ACM, 2015, pp. 393–406.

[168] X. Lin and V. J. Venkataramanan, "On the large-deviations optimality of scheduling policies minimizing the drift of a lyapunov function," in *2009 47th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, Sep. 2009, pp. 919–926.

[169] E. Frazzoli and M. Dahleh, "Chapter 13, lectures on dynamic systems and control," 2011. [Online]. Available: https://ocw.mit.edu/

[170] H.-C. Lin and C. S. Raghavendra, "An approximate analysis of the join the shortest queue (jsq) policy," *IEEE Transactions on Parallel and Distributed Systems*, vol. 7, no. 3, pp. 301–307, Mar 1996.

[171] R. D. Foley and D. R. McDonald, "Join the shortest queue: stability and exact asymptotics," *Ann. Appl. Probab.*, vol. 11, no. 3, pp. 569–607, 08 2001.

[172] P. Key, L. Massoulie, and D. Towsley, "Multipath routing, congestion control and dynamic load balancing," in *2007 IEEE International Conference on Acoustics, Speech and Signal Processing - ICASSP '07*, vol. 4, April 2007, pp. IV–1341–IV–1344.

[173] Y. Geng, V. Jeyakumar, A. Kabbani, and M. Alizadeh, "Juggler: a practical reordering resilient network stack for datacenters," in *Proceedings of the Eleventh European Conference on Computer Systems*. ACM, 2016, p. 20.

[174] P. Key, L. Massoulie, and D. Towsley, "Path selection and multipath congestion control," in *IEEE INFOCOM 2007 - 26th IEEE International Conference on Computer Communications*, May 2007, pp. 143–151.

[175] Y. Chen, R. Griffith, J. Liu, R. H. Katz, and A. D. Joseph, "Understanding tcp incast throughput collapse in datacenter networks," in *Proceedings of the 1st ACM workshop on Research on enterprise networking*. ACM, 2009, pp. 73–82.

[176] A. Fox, S. D. Gribble, Y. Chawathe, E. A. Brewer, and P. Gauthier, "Cluster-based scalable network services," in *Proceedings of the Sixteenth ACM Symposium on Operating Systems Principles*, ser. SOSP '97. New York, NY, USA: ACM, 1997, pp. 78–91.

[177] M. Dahlin, "Interpreting stale load information," *IEEE Transactions on Parallel and Distributed Systems*, vol. 11, no. 10, pp. 1033–1047, Oct 2000.

[178] A. Varga and R. Hornig, "An overview of the omnet++ simulation environment," in *Proceedings of the 1st International Conference on Simulation Tools and Techniques for Communications, Networks and Systems &*

Workshops. Brussels, BEL: ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2008.

[179] A. Orebaugh, G. Ramirez, J. Beale, and J. Wright, *Wireshark & Ethereal Network Protocol Analyzer Toolkit*. Syngress Publishing, 2007.

[180] T. Bassetto and F. Mason, "Transient behaviour in some queueing system related to an emergency service," *Computer Science*, 2004.

[181] N. Hu and P. Steenkiste, "Improving tcp startup performance using active measurements: algorithm and evaluation," in *11th IEEE International Conference on Network Protocols, 2003. Proceedings.* IEEE, 2003, pp. 107–118.

[182] C. Hollot, V. Misra, D. Towsley, and W.-B. Gong, "A control theoretic analysis of red," in *Proceedings IEEE INFOCOM 2001. Conference on Computer Communications. Twentieth Annual Joint Conference of the IEEE Computer and Communications Society (Cat. No.01CH37213)*, vol. 3, 2001, pp. 1510–1519 vol.3.

[183] P. Przybylowicz, M. Sobieraj, and L. Stepien, "Efficient approximation of sdes driven by countably dimensional wiener process and poisson random measure," *arXiv preprint arXiv:2108.02394*, 2021.

[184] F. Baccelli, D. R. McDonald, and J. Reynier, "A mean-field model for multiple tcp connections through a buffer implementing red," *Performance Evaluation*, vol. 49, no. 1-4, pp. 77–97, 2002.

[185] V. Guffens, "Compartmental fluid-flow modelling in packet switched networks with hop-by-hop control," PhD Thesis, Universit catholique de Louvain, 2005.

[186] S. Sinha, S. Kandula, and D. Katabi., "Harnessing tcps burstiness with flowlet switching," *in ACM Hotnets*, 2004.

[187] M. Alizadeh, A. Kabbani, T. Edsall, B. Prabhakar, A. Vahdat, and M. Yasuda, "Less is more: Trading a little bandwidth for ultra-low latency in the data center," in *Proceedings of the 9th USENIX Conference on Networked*

Systems Design and Implementation, ser. NSDI'12. Berkeley, CA, USA: USENIX Association, 2012, pp. 19–19.