SIERPINSKI TRIANGLE BASED DATA-CENTER NETWORK ARCHITECTURE IN CLOUD COMPUTING

QI HAN

FACULTY OF COMPUTER SCIENCE AND
INFORMATION TECHNOLOGY
UNIVERSITY OF MALAYA
KUALA LUMPUR

2016

SIERPINSKI TRIANGLE BASED DATA-CENTER NETWORK ARCHITECTURE IN CLOUD COMPUTING

QI HAN

THESIS SUBMITTED IN FULFILMENT
OF THE REQUIREMENTS
FOR THE DEGREE OF DOCTOR OF PHILOSOPHY

FACULTY OF COMPUTER SCIENCE AND
INFORMATION TECHNOLOGY
UNIVERSITY OF MALAYA
KUALA LUMPUR

2016

UNIVERSITI MALAYA

ORIGINAL LITERARY WORK DECLARATION

Name of Candidate: QI Han

Registration/Matrix No.: WHA110031

Name of Degree: Doctor of Philosophy

Title of Project Paper/Research Report/Dissertation/Thesis ("this Work"):

Sierpinski Triangle Based Data-Center Network Architecture in Cloud Computing

Field of Study: Cloud Computing

I do solemnly and sincerely declare that:

(1) I am the sole author/writer of this Work;
(2) This work is original;
(3) Any use of any work in which copyright exists was done by way of fair dealing and for permitted purposes and any excerpt or extract from, or reference to or reproduction of any copyright work has been disclosed expressly and sufficiently and the title of the Work and its authorship have been acknowledged in this Work;
(4) I do not have any actual knowledge nor do I ought reasonably to know that the making of this work constitutes an infringement of any copyright work;
(5) I hereby assign all and every rights in the copyright to this Work to the University of Malaya ("UM"), who henceforth shall be owner of the copyright in this Work and that any reproduction or use in any form or by any means whatsoever is prohibited without the written consent of UM having been first had and obtained;
(6) I am fully aware that if in the course of making this Work I have infringed any copyright whether intentionally or otherwise, I may be subject to legal action or any other action as may be determined by UM.

Candidate's Signature Date

Subscribed and solemnly declared before,

Witness's Signature Date

Name: Abdullah Gani
Designation: Professor

ABSTRACT

This thesis reports on the research to develop of a data center network (DCN) architecture to solve the problem of network performance in cloud-oriented data centers. Computational clouds are increasingly becoming popular for the provisioning of computing resources and service on demand basis. A DCN is an important component of data centers that consists of a large number of hosted servers and switches connected with high speed communication links. As a backbone in data centers, a DCN enables the deployment of resources centralization and on-demand access of the information and services of data centers to users. In recent years, the scale of the DCN has constantly increased with the widespread use of cloud-oriented services and applications configured over virtual machines (VMs), and the unprecedented amount of data delivery in/between data centers, whereas the traditional DCN tree-based architecture lacks aggregate bandwidth, scalability and cost effectiveness for coping with the increasing demands of tenants in accessing the services of cloud-oriented data centers. To solve this problem, the method developed in this research is used to mitigate the aggregation throughput and improve the network performance of DCN by using a novel DCN architecture. The proposed method, called Sierpinski Triangle Based (STB) DCN architecture, is developed on the basis of the well-know Sierpinski triangle fractal to mitigate throughput bottleneck in aggregate layers as accumulated in tree-based structure. STB is a fault-tolerant architecture which provides at least two parallel paths for each pair of servers. It also supports various bandwidth-intensive applications by providing high network throughput for all-to-all traffic. STB architecture was implemented in a real cloud data center environment and evaluated in Network Simulator 2 (NS2) simulation. The performance of STB architecture is validated by comparing the results with the traditional tree-based, and DCell DCN architectures. Theoretical analysis and implementation experiences verify that the

proportion of server to entire nodes in STB is same with DCell but higher than that of tree-based architecture. The average shortest path length is restricted between 5.0 and 6.7, when node failure proportion remains between 0.02 and 0.2, shorter than DCell in a 4-level architecture. The results of the experiment also show that the STB architecture has higher throughout than both traditional tree-based and DCell architectures from the scale of 12 to 363 servers with/without link failure happens. From the results of both simulation and experiment in actual devices, we speculate that STB still can achieve better network performance in throughput, server utilization, average shortest path length than DCell and tree-based architectures in real large-scale cloud-oriented DCN.

ABSTRAK

Tesis ini melaporkan penyelidikan untuk membangunkan satu rangkaian pusat data (DCN) seni bina untuk menyelesaikan masalah prestasi rangkaian di pusat-pusat data berorientasikan awan. Pengkomputeran awam semakin menjadi popular untuk penyediaan sumber dan perkhidmatan secara permintaan. A DCN adalah satu komponen penting dalam pusat data yang terdiri daripada sebilangan besar tuan rumah pelayan dan suis yang berkaitan dengan hubungan komunikasi kelajuan tinggi. Sebagai tulang belakang di pusat-pusat data, DCN membolehkan penggunaan sumber pemusatan dan akses atas permintaan maklumat dan perkhidmatan pusat data kepada pengguna. Dalam tahun-tahun kebelakangan ini, skala DCN telah sentiasa meningkat dengan penggunaan meluas perkhidmatan dan aplikasi berorientasikan awan yang dikonfigurasi dengan mesin maya (VM), dan jumlah yang belum pernah terjadi sebelumnya penghantaran data dalam / antara pusat-pusat data, manakala DCN pokok- tradisional seni bina berasaskan tidak mempunyai bandwidth agregat, berskala dan keberkesanan kos bagi menangani permintaan yang semakin meningkat penyewa dalam mengakses perkhidmatan pusat data berorientasikan awan. Untuk menyelesaikan masalah ini, kaedah yang dibangunkan dalam kajian ini adalah untuk mengurangkan throughput pengagregatan dan meningkatkan prestasi rangkaian dari DCN dengan menggunakan seni bina DCN novel. Kaedah yang dicadangkan, yang dipanggil Sierpinski Triangle Based (STB) DCN seni bina, dibangunkan atas dasar yang terkenal Sierpinski segitiga fraktal untuk mengurangkan kesesakan pemprosesan dalam lapisan agregat terkumpul dalam struktur berasaskan pokok. STB adalah seni bina kesalahan-toleran yang menyediakan sekurang-kurangnya dua laluan selari untuk setiap pasangan pelayan. Ia juga menyokong pelbagai aplikasi lebar jalur yang intensif dengan menyediakan rangkaian pemprosesan tinggi untuk semua-untuk-semua lalu lintas. Seni bina STB telah dilaksanakan dalam persekitaran pusat data awan sebenar dan dinilai da-

lam Network Simulator 2 (NS2) simulasi. Prestasi seni bina STB disahkan dengan membandingkan hasil dengan seni bina tradisional Tree-based dan DCell DCN. Analisis dan pelaksanaan teori pengalaman mengesahkan bahawa kadar pelayan kepada seluruh nod dalam STB adalah sama dengan DCell tetapi lebih tinggi daripada seni bina tree-based. Purata panjang laluan terpendek adalah terhad antara 5.0 dan 6.7, apabila kegagalan nod kadar tetap antara 0.02 dan 0.2, lebih pendek daripada DCell dalam seni bina 4-peringkat. Keputusan eksperimen juga menunjukkan bahawa seni bina STB mempunyai tinggi di seluruh daripada kedua-dua tree-based dan DCell seni bina tradisional dari skala 12-363 pelayan dengan / tanpa kegagalan link yang berlaku. Dari hasil kedua-dua simulasi dan eksperimen dalam peranti sebenar, kami membuat spekulasi bahawa STB masih boleh mencapai prestasi yang lebih baik dalam rangkaian pemprosesan, penggunaan pelayan, purata panjang jalan singkat daripada DCell dan seni bina tree-based dalam skala besar sebenar berorientasikan awan DCN.

ACKNOWLEDGEMENTS

Creating a Ph.D. thesis is not an individual experience; rather it takes place in a social context and includes several persons. Immeasurable appreciation and deepest gratitude for the help and support are extended to the following persons who in one way or another have contributed in making this research possible.

To the supervisor, Professor Abdullah Gani, dean of the Faculty of Computer Science and Information Technology (FCSIT), who has taken a particular interest in this research and given constant guidance and encouragement, without which this work would not have been possible. For his understanding, wisdom, patience, enthusiasm, and unwavering support, a special debt of gratitude is due. Many thanks also go to Dr. Nor Badrul Anuar Juma'at, Dr. Liew Chee Sun, and Dr. Por Lip Yee, whose kind advices and suggestions have been of great value to the research. The members of the thesis examination committee: Professor Xia Feng, Professor Abdul Hanan Abdullah, and Dr. Hamid Abdulla Jallb, who generously gave their time to offer valuable comments toward improving this work. Special thanks must go to them as well.

During the research, the constant association with the members of Center for Mobile Cloud Computing Research (C4MCCR) has been most pleasurable. Without their help and counsel, always generously and unstintingly given, the completion of this work would have been immeasurably more difficult. Hereby, deepest gratitude go to Muhammad Shiraz, Saied Abolfazli, Zohre Sanaei, Mehdi Sookhak, Ejaz Ahmed, Md Whaiduzzaman, Anjum Naveed, as well as Suleman Khan, Ibrar Yaqoob, and Liu Jie Yao, good times come and go, but the memories will last forever.

A heartfelt gratitude goes to the staffs of the main office in FCSIT, especially to Mr. Muhamad Afiq, Mr. Mazrulhisham, Ms. Ilyana, and Ms. Lily, who kindly provided various administrative supports to the author.

The author wishes to express his sincere appreciation for the supplies and facilities from the High Impact Research Grant funded by the Malaysian Ministry of Higher Education under the University of Malaya (UM.C/HIR/MOHE/FCSIT/03).

In this special moment, the author also would like to express his deepest gratitude to his beloved parents, Qi Weiping and Zou Guifen, whose love encouragement and supports both financially and mentally that made him possible to accomplish this undertaking. Dedicated to the author's parents-in-law, Zhu Liya and Wu Yunfu for patiently extended all sorts of help and unfailing emotional support in recent years that provided the foundation for this work.

This dissertation would not have been possible without Mrs. Qi Zhu Xiaomei, the author's wonderful wife and soulmate. All of her continuous love, selfless dedication, unconditional support, greatest trust, timely encouragement, and endless patience during the past few years are deeply appreciated. "You are not just my wife, you are also my life." More gratitude also goes to her for bringing Timmy, the greatest gift of life to the family. Hope this baby boy would be proud of his daddy.

To them of above, the author would like to say "Because of your support, so I can stand on the mountain; because of your company, so I will never feel alone; because of your encouragement, so I can walk farther than I thought I could go."

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS AND ABBREVIATIONS

AMAC	:	Actual Media Access Control
ARP	:	Address Resolution Protocol
ASP	:	Application Service Provider
CaaS	:	Communication as a Service
CBR	:	Constant Bit Rate
CDC	:	Cloud-oriented Data Center
DBaaS	:	Database as a Service
DC	:	Data Center
DCN	:	Data Center Network
DHCP	:	Dynamic Host Configuration Protocol
DIP	:	Direct Internet Protocol
EBGP	:	External Border Gateway Protocol
EC2	:	Elastic Compute Cloud
ECMP	:	Equal Cost Multi-Path
GAE	:	Google App Engine
GFS	:	Google File System
HDFS	:	Hadoop Distributed File System
HFN	:	Hyper Fat Tree Network
HUaaS	:	Human as a Service
IaaS	:	Infrastructure as a Service
LAN	:	Local Area Network
IDC	:	International Data Corporation
IGP	:	Interior Gateway Protocol
I/O	:	Input/Output
IT	:	Information Technology
LSR	:	Link State Routing
MaaS	:	Monitoring as A Service
MDC	:	Modular Data Center
MEMS	:	Micro Electro Mechanical Switch
NaaS	:	Network as a Service
NIC	:	Network Interface Card
NS2	:	Network Simulator 2
O-E-O	:	Optics to Electrical To Optics
OS	:	Operation System
OSA	:	Optical Switching Architecture
OSM	:	Optical Switching Matrix
OSPF	:	Open Shortest Path First
PaaS	:	Platform as a Service
PM	:	Physical Machine
PMAC	:	Pseudo Media Access Control
PR	:	Physical Resources

QoS	:	Quality of Service
RIP	:	Routing Information Protocol
RTO	:	Retransmission Time Out
SaaS	:	Software as a Service
SLA	:	Service Level Agreement
STB	:	Sierpinski Triangle Based
TCP	:	Transmission Control Protocol
ToR	:	Top of Rack
UDP	:	User Datagram Protocol
UI	:	User Interface
VIP	:	Virtual Internet Protocol
VLB	:	Variant Load Balancing
VM	:	Virtual Machine
VPN	:	Virtual Private Network
VR	:	Virtual Resources
WDCN	:	Wireless Data Center Network
WDM	:	Wavelength Division Multiplexing
WSS	:	Wavelength Selective Switch
WTU	:	Wireless Transmission Unit
XaaS	:	Everything as a Service

LIST OF APPENDICES

CHAPTER 1

INTRODUCTION

1.1 Background

This thesis reports on a research, which was aimed at solving the problem of limited throughput bottleneck in aggregate layers and lower server utilization in cloud computing data center networks (DCNs).

Cloud computing is a network-based computing model that provides services, such as Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS), on-demand (Armbrust et al., 2010). In 2006, Google proposed the 101 plan which evenly introduced the concept of cloud (Baker, 2007). Contrasted by the traditional personal-centric local computing, cloud computing is internet-centric and provides safe, reliable, fast, convenient, transparent high performance computing, mass data storage, and other internet-based services to clients by data centers. Such computing model emerged from Distributed Computing, Parallel Computing and Grid Computing. Mobile Cloud Computing, as an extension of cloud computing, provides more suitable conditions for mobile device users to fully enjoy the benefits and convenience of cloud computing via wireless networks (Sanaei et al., 2014) (Shiraz et al., 2014).

Since the advent of mobile cloud computing, data volume has increased tremendously on the Internet. The International Data Corporation (IDC) report announced that the size of big-data generated in 2011 reached 1.8 Zettabyte (1.8 trillion GB) and the data is expected to increase 50-fold in the next 10 years, which will reach 35.2ZB in 2020 (Gantz & Reinsel, 2012). The deployed data management and processing mechanisms in Data Center Network (DCN) such as Google File System (GFS)(Ghemawat et al., 2003),

The Hadoop Distributed File System (HDFS)(Borthakur, 2007), BigTable(Chang et al., 2008), Dryad(Isard et al., 2007), MapReduce (Dean & Ghemawat, 2008), are responsible for managing and processing the massive data. For supporting such cloud services and important applications (for example, scientific computations, financial analysis, massive data processing and warehousing, as well as utility computing), Amazon, Google, Sales-force.com and other corporations have established large data centers around the world (Buyya et al., 2008).

1.2 Motivation

As a basic hardware infrastructure of data center and cloud computing, the DCN has rapidly become a research focus issue in recent years. Data Center Network (DCN) is an important component of data center that consists of a large number of hosted servers and switches connected with high speed communication links. A DCN enables the deploy-ment of centralized resources and on-demand access of the information and services of data centers to users. The motivations for DCN establishment are as follows.

First of all, the scale of the DCN is constantly increasing with the widespread use of cloud-oriented services. It is common for a cloud-oriented DC to contain hundreds to thousands of servers in an economy of scale (Beloglazov et al., 2011). For supporting such cloud services and important applications, Microsoft, for example, established a 707,000-square-foot DC building in Chicago, 2009 (Vahdat et al., 2010). There are 162 containers of 2,500 servers each with total 60 Megawatts of electricity in the building which costs 500 million US dolors. Apple data center in Maiden was established in 2010 with 500,000-square-foot and costs 1 billion dollars(Tarantino, 2012). Thereby, a method to effectively interconnect the number of exponentially increasing servers is desired.

Secondly, unprecedented amount of data delivery in/between data centers. As the above mentioned systems and applications (such as GFS) are data and communication

intensive (a simple Web search request may need a cooperation among more than 1,000 servers), the information exchanging among remote nodes and local servers to proceed computation is increasing rapidly. Thereby, a great data traffic flow stress is taken to the data center and the limited inter-node communication bandwidth among servers is becoming serious bottleneck to DCN.

Thirdly, reasonable cost and elastic utilization according to the business requirements from Information Technology (IT) investment of enterprises for DCNs. The cost of using enterprise-class network equipment is large (upwards of $12 million per month for a 100,000 servers data center) and is not suitable to accommodate Internet-scale services in data centers. To be profitable, data centers better use some lower cost network equipment to achieve high utilization with agile end-to-end network capacity assignment and un-fragmented server pools (Greenberg, Hamilton, et al., 2008).

In recent years, with the constantly increased scale of DCN, the traditional architecture such as tree-based and Clos network, however, lacks aggregate bandwidth, scalability, and faces the cost of coping with the increasing demands of tenants in accessing the services of cloud-oriented data centers. Therefore, the design of a new DCN architecture with the features of scalability, low cost, robustness, and energy conservation is required, because the architectures have an impact on the overall properties of the DCN.

1.3 Statement of Problem

Currently, the widely used architecture in DCN is a typical multi-root tree architecture, commonly composed by either two- or three-layer of switches (T. Wang et al., 2014). A three-layers architecture has a core layer in the root, an aggregation layer in the middle and an edge layer at the leaves of the tree. A two-layer architecture has only the core and the edge layers. The tree-based structure is simple and can be easily deployed and extended by increasing the rack of servers and switches. However, due to the architecture

is initially designed for relatively small or medium scale networks, the shortcomings that deployed in large-scale cloud-oriented DCNs mainly focus on the following.

Throughput Restriction. Data flow between lower layer servers is transferred through the upper layer, however the links between the lower and core layers are normally over subscribed by factors of 5:1 or more due to the equipment cost concerns, which limits the communication among the servers in different branches of the tree, and leads to congestion and computation hot-spots even if the network capacity is available elsewhere (Greenberg et al., 2009).

Network Scalability. The tree-based hierarchical architecture, typically, can only support up to 8,000 servers due to the restricted number of network ports (I/O interfaces) and meeting the requirement of a fast failure recovery mechanism (Al-Fares et al., 2008). Therefore, it is difficult to support the large numbers of servers in the cloud-oriented data centers. Furthermore, each parent node normally has several child nodes in the tree-based architecture. From the core to edge switches, a switch $(parentnode)$ can affect tens to thousands of the existing operational servers without redundancy because the switch poses as the bandwidth bottlenecks due to the single-point-of-failure (Guo et al., 2008).

Low Rates of Resource Utilization. Redundancy is a common approach to improve the reliability and availability of a network. In tree-based architecture, 1:1 equipment redundancy is used on the network devices in upper layers. Because the IP assignment and network topology is closely related in the architecture, it is inconvenient to reconfigure the whole IP address for all devices in a scale-increasing network. Therefore, normally a number of standby servers, bandwidth and IP resources are reserved to ensure the well running of DCN. However, these resources only be provided when node or link failure happens.

Price-Performance Ratio. The cost for building a cloud-oriented DCN greatly affects architecture design decisions. As we discussed above, a tree-based architecture

supports a few thousand servers. To sustain the exponential increasing of servers, more higher level are added, and more expensive advanced switches and network equipment are used, which affects the price-performance ratio in DCN.

This research will address the problem of low performance of the traditional tree-based DCN architecture from the perspectives of network throughput, scalability, rates of resource utilization and price-performance ratio.

1.4 Research Aim and Objectives

The aim of this research is to develop a new cloud-oriented DCN architecture with the features of higher network throughput, scalability, rates of server utilization and price-performance ratio, more fault-tolerant routing, and lower average shortest path length in DCN. The objectives of this research are as follows:

•To study the traditional tree-based network architecture and other existing architectures for DCNs, in order to identify the gap of performance flaws in DCN.

•To conducting the recent investigations in routing mechanism research on DCNs.

•To design the Sierpinski Triangle Based (STB) architecture framework that can

(a) increase the throughput in upper layer of DCN,

(b) optimize the scalability of DCN,

(c) improve the rates of server utilization and price-performance ratio in DCN,

(d) reduce the average shortest path length in DCN,

(e) provide fault-tolerance routing in DCN without redundancy devices.

•To evaluate the performance of the proposed STB architecture via benchmarking on simulation and real cloud experiment environment.

•To validate the results of performance evaluation using T-Test statistical examination.

1.5 Research Questions

The central question of this research is, how to improve the network performance of cloud-oriented DCN from the perspectives of network throughput and scalability, rates of server utilization and price-performance ratio, average shortest path length, as well as fault-tolerance routing provision.

The following are subquestions:

1. What are the reasons for the traditional tree-based architecture lacks in the above perspective matters?

2. What is the current state of cloud-oriented DCN architecture research?

3. What is a better design for cloud-oriented DCN architecture?

4. How can we evaluate the performance of the above selected parameters?

1.6 Scope of Work

In this research, the tree-based network architecture is selected as a benchmarking, and also be used to evaluated the performance of throughput, network scalability , average shortest path length, and fault tolerance routing in cloud-oriented DCN. Improvement will be made to the above network performance of DCN by deploying a new STB architecture. The tree-based architecture is chosen because of its simplicity, and be widely used in most of current DCN, despite the fact that it has certain weaknesses such as the uncertain aggregation layer throughput bottleneck in the data delivery process. The tree-based architecture is not good for use in current cloud-oriented DCN Greenberg et al. (2009).

DCell Guo et al. (2008) architecture is also selected in this research to make a performance comparison to the tree-based and proposed STB architectures. The reason of choosing DCell is the method of construction in DCN. Both STB and DCell are all recursively defined architectures with similar features in network properties and performance

by deploying more interfaces and ports in network devices. Even though the DCell exists some shortcomings such as lower level servers undertake more forwarding tasks, it has been a milestone in the relevant research because of the novel thinking in design.

Because it is difficult to establish a real large-scale cloud-oriented data center, the network scenarios of the model developed in this research, are started from a small-sale cloud-oriented DCN with real devices, to medium-scale with virtual machines, and large-scale simulation using Network Simulator 2(NS2). The Hadoop MapReduce model is deployed in DCN to establish cloud computing environment. The performance analysis in this research is using network analysis tools nTop and NS2. Some mathematical models are also created to analyze the selected architectures from different parameters as well. In this research, all the outputs are validated using Paired Two-Sample for Means t-test.

In a real cloud-oriented DCN, the network performance is affected by bandwidth, power supply, energy consumption, memory capacity, CPU performance and so on. In this research, tests are carried out to evaluate the network throughput and the other selected parameters which are indicated in previous sections. The above mentioned affect issues will be adopted a given set of values and not be discussed in this research.

1.7 Proposed Methodology

In this research, a new architecture for cloud-oriented DCN, called Sierpinski Triangle Based (STB) architecture, is designed. For achieving this objectives, the methodology of this research are as follows:

We study the latest research efforts to identify issues in the traditional tree-based architecture and current cloud-oriented architectures for DCNs, determine the weaknesses and shortcomings of the variable architectures. We review recent literature collected from on-line scholarly databases like IEEE and ACM for instance, to identify the most critical and hot-spot problem issues in related research.

The research problem is investigated by using theoretical analysis and implementation benchmarking in real DCN environment. The theoretical analysis is performed by using mathematical model to validate the significance of the research problem.

We designed the Sierpinski Triangle Based (STB) architecture for cloud-oriented DCNs to address the research problem and achieve the research objectives. The designed architecture uses Sierpinski triangle fractal to mitigate throughput bottleneck in aggregate layers and increase the rates of server utilization in DCNs. In addition, a STB routing and node identification mechanism is designed for addressing the issue of lower resource utilization in DCNs.

The designed architecture is evaluated in a real cloud data center environment and simulated using Network Simulator 2 (NS2). We establish STB architecture in NS2, and use MapReduce workloads to systematically measure the proposed routing and node identification mechanism to ensure the operation in a real cloud experiments. We also build the designed architecture and benchmarking method in a real cloud DCN environment to measure the MapReduce execution time and network throughput.

The evaluation is analyzed from the perspective of throughput, rates of server utilization and the average shortest path length in DCN. The experimental results are validated by comparing with tree-based and DCell cloud-oriented architectures. The statistical model is validated using T-Test two-samples means validation examination.

1.8 Research Contribution

The major contributions from this research is as follows:

1. The research exposes the limitations of the traditional tree-based architecture deployed in cloud-oriented DCN.

2. The research establishes a taxonomy to analyze the implications and critical aspects of current DCN architectures and making a comparison from the significant param-

eter metrics.

3. The research deals with the lower network performance of existing large-scale cloud-oriented DCN by proposing the STB network architecture.

4. The research creates new knowledge in designing a network architecture for a cloud-oriented DC in further research.

1.9 Thesis Layout

The thesis has a total of 7 chapters which are illustrated in Figure 1.1 for skimming and brief understanding the content of the thesis. The remainder of this thesis is organized as follows:

Chapter 2 presents the literature review from the state-of-the-art research and identifies the open research problems in cloud-oriented DCNs. It starts with a quick overview about cloud computing and DCNs, followed by highlighting the major requirements and challenges of cloud-oriented DCN architecture design. This chapter also presents an overview of existing architectures for DCNs according to taxonomy and the comparison of these architectures on the basis of significant parameters. Furthermore, the chapter identifies the most significant problem to be addressed in this thesis.

For demonstrate the significance of the identified problem from theoretical analysis and implementation experiences, we analyze the performance of the traditional tree-based architecture in cloud environment in chapter 3. We use mathematical equations to identify the rate of server utilization and cost, and implementation to demonstrate throughput of the tree-based architecture in DCN.

Chapter 4 provides an overview of the Sierpinski Triangle Based architecture in cloud-oriented DCN. It presents the physical components of the proposed architecture and describe the building procedure of STB in detail. The properties, theoretical findings and routing mechanism are presented as well.

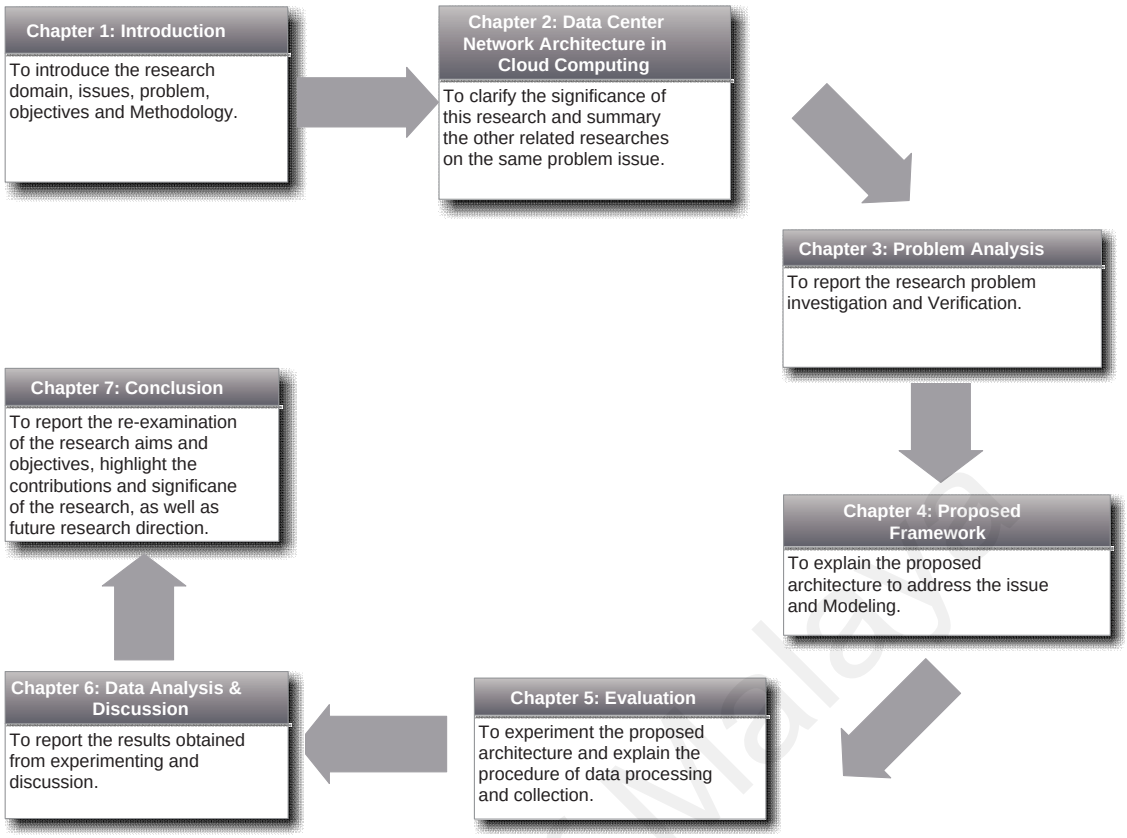

Figure 1.1: Thesis Orgnization

Chapter 5 reports on the implementation of experiment in real cloud computing environment and the simulation study of the research. It explains the test-bed design and the configuration of the simulation, as well as the tools for data generation, processing, and collection.

Chapter 6 gives a discussion of the experimental results of STB. It shows the effects of the proposed architecture on the performance of DCN. The advantages and disadvantages of STB are summarized by comparing it to the traditional tree-based and DCell architectures. Performance of throughput, proportion of servers to switches, and average shortest path length will also be discussed in this chapter.

Chapter 7 concludes the thesis by summarizing achievements of the objectives, and highlights the benefits and the limitations of the proposed architecture. The contributions of this research, including international scholarly publications are stated before finally pointing out future directions of this research for enhancing this research further.

CHAPTER 2

DATA CENTER NETWORK ARCHITECTURE IN CLOUD COMPUTING - LITERATURE REVIEW

This chapter reviews relevant literature and research findings pertaining to cloud-oriented data center network (DCN). For understanding more detailed knowledge of the subject matter, this chapter reviews the state-of-the-art DCN architecture according to the thematic taxonomy, and gives a comparison of the selected architectures as well. Several open research issues and challenges in cloud-oriented DCN architecture design are also highlighted.

The remainder of this chapter is as follows. Section 2.1 presents the fundamental concepts of cloud computing from the aspects of service level and technology level. An overview of DCN is also introduced in this section. Section 2.2 presents taxonomy of the DCN architectures and reviews current architectures on the basis of taxonomy. A comparison of the introduced architectures is given according to selected significant parameters in section 2.3. Section 2.4 summarizes the chapter with conclusive remarks.

2.1 Background

Over the past few years, advances in the field of network based computing and application on demand have led to an explosive growth of application model such as cloud computing, software as a service, community network, web store and so on. As a major application model in the era of the Internet, cloud computing has become a significant research topic in the communities of scientific and industrial since 2007.

2.1.1 Cloud Computing

Commonly, cloud computing is described as a range of services which are provided by an Internet-based cluster system. Such cluster systems consist by a group of low-cost servers or PCs, organizing the various resources of the computers according to some certain management strategy, and offering safe, reliable, fast, convenient and transparent services such as data storage, accessing and computing to clients. The cloud computing system is the development of parallel processing, distributed and grid computing on the Internet, which provides various QoS guaranteed services such as hardware, infrastructure, platform, software and storage to different Internet applications and users. Figure 2.1 indicates an overview of cloud computing. The surrounding computers are users, the "cloud" is resource and service of cloud computing, and several service providers are shown as well.

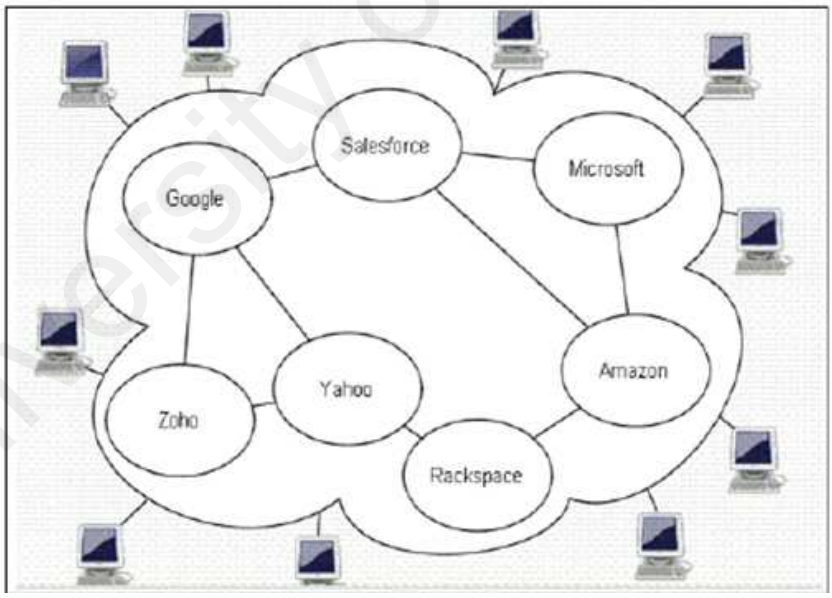

Figure 2.1: Overview of Cloud Computing

2.1.1.1 Infrastructure

The idea of cloud computing is to fully utilize the existing computer and network technology to integrate computing resource, maximize resource sharing, thereby to solve the

issues of cloud management and large-scale computing. At present, there is no uniform standard in cloud computing infrastructure. Each cloud computing service providers is using their own cloud infrastructure. In this section, we introduce the infrastructure of cloud computing from the following three aspects: 1. Basic architecture of cloud computing; 2. Functions and services in different layer; 3. Describe cloud computing from technical aspect.

The platform of cloud computing consists of a "CLOUD" with large-scale and multi-functions. In cloud, mass scattered computing resources and services are connected through the Internet, and allocated to users by virtualization. Therefore, a platform that can provide large-scale computing and mass data storage is established. The following figure 2.2 shows a common architecture of cloud computing.

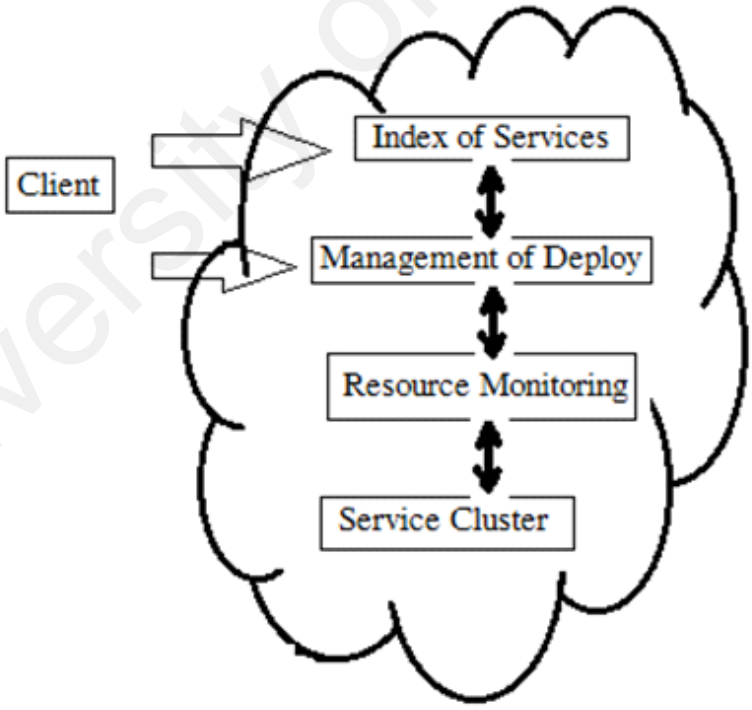

Figure 2.2: Common Infrastructure of Cloud Computing

In figure 2.2, the "Client" is an access port for user using cloud, which provides the interface for registration, login, request a service and so on. from web browser. The

operation is as simple as in local PC. In “Index of services”, users select or customize their desire services, or unsubscribe the previous services through the “Client” access interface. The main function of “management of deploy” are manage clients and services deployment, which include user authentication, authorization and login, response service request, allocation, as well as result output. Resource in cloud is monitored and measured by “Resource Monitoring”, to ensure it can be allocated correctly. Server cluster includes a group of managed virtual and physical servers to process mass applications, computing process, and data storage as well.

2.1.1.2 Service Level

Cloud computing is emerging as viable services model, therefore Everything as a Service (XaaS) (Rimal et al., 2009) is viewed as a significant trend. Such as, Software as a Service (SaaS), Platform as a Service (PaaS), Hardware as a Service (HaaS), Infrastructure as a Service (IaaS), Network as a Service (NaaS), Monitoring as a Service (MaaS), Database as a Service (DBaaS), Communications as a Service (CaaS), Human as a Service (HuaaS) and so on. See figure 2.3.

Infrastructure as a Service (IaaS) (Bhardwaj et al., 2010) is the delivery of computer infrastructure as a service. Aside from the higher flexibility , a key benefit of IaaS is the latest technology and usage-based payment scheme. In IaaS, provider offers virtual resources (VR), physical resources (PR), storage, load balancers and local area network (LAN) and/or virtual private network (VPN) to users. Users are responsible for setting up operating system, installing own application software, patching and maintaining the operating system and application software. Amazon Elastic Compute Cloud (EC2) (Shankar, 2009) is a typical example in IaaS.

Platform as a Service (PaaS) (Beimborn et al., 2011) enables application developers with a platform including all the systems and environments to run their software solu-

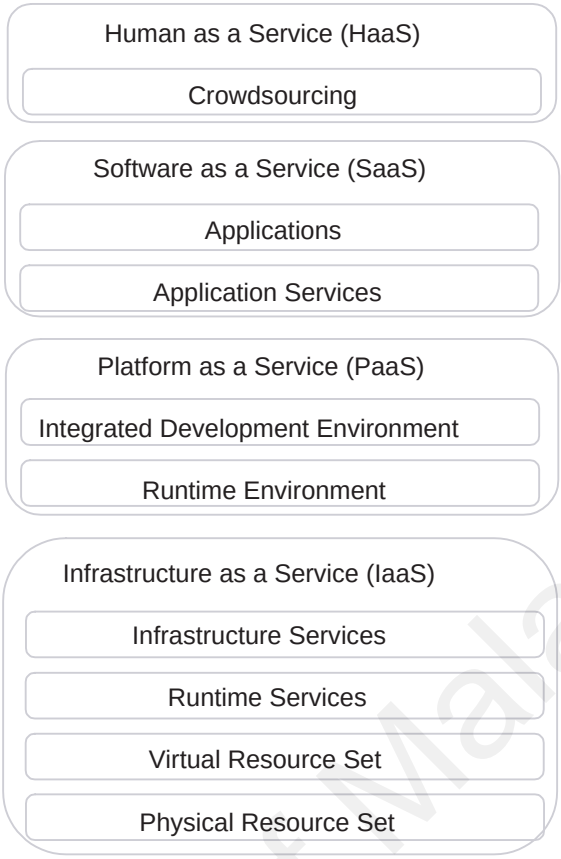

Figure 2.3: Cloud Layering Concept

tions in a cloud-based environment without having to buy costly hardware. Compared with conventional application development, cloud providers offer programming and execution environment, operation system, programming language, database, web server and vary available tools with quickly scale. Key examples are Google App Engine (GAE) (Zahariev, 2009) and Microsoft Azure (Redkar & Guidici, 2011).

Software as a Service (SaaS) (Buxmann et al., 2008) is a multi-tenant platform and commonly referred to as the Application Service Provider (ASP) model, which offers application software, programming interfaces, elasticity, manages cloud infrastructure and platform, and charges typically on a monthly or yearly basis. Examples of the key providers are Microsoft windows live, Google Docs, Salesforce.com and so on.

Human as a Service (HuaaS) (W. Li & Svard, 2010) is an Extension of XaaS to non-IT services. A group of humans can be used to perform tasks such as translation, design,

research, development and so on. Key examples for HuaaS are Amazon Mechanical Turk, Microworkers, Wikipedia, YouTube and so on.

Currently, most of cloud services belong to IaaS and PaaS, only few providers such as Amazon provides SaaS to users. However, the SaaS will be the predominate component for users in short future because they can save the budget from IT maintenance and pay more attention on their own business.

Though the various services be able to provide via cloud, we believe that the key component of cloud-oriented service is 'computing', the rests, such as storage, platform, security, are all additional services based on cloud. From the perspective of technology, the cloud computing actually is a kind of computing virtualization.

2.1.1.3 Technology Level

The division of service in cloud computing is from aspect of provided services. By contrast, the division of technology in cloud computing is according the character of cloud and the property of system. Hence, cloud computing can be divided into 4 parts: physical resource, virtualized resource, system management, and service interface (see figure 2.4).

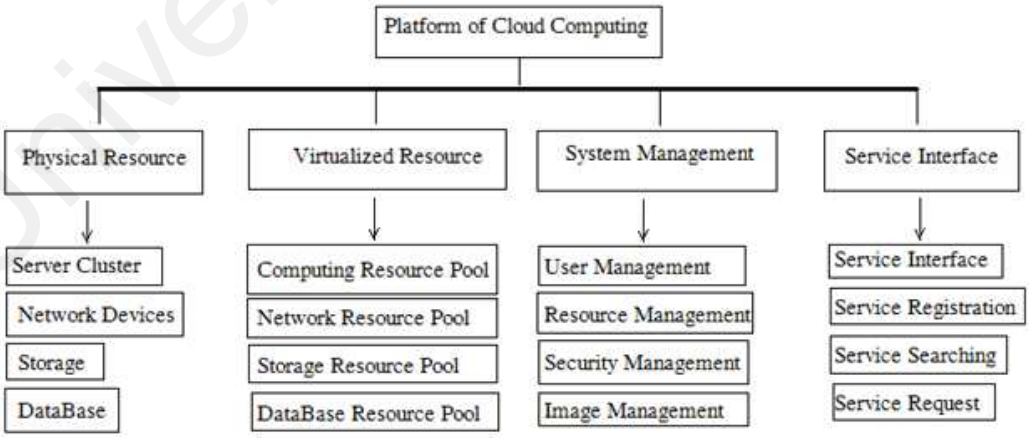

Figure 2.4: The Division of Cloud Computing in Technology

Physical resource is the real resource includes hardware and other equipment. In cloud computing, a group of low-cost PCs can be used in the infrastructure. Those com-

puting resource in real devices is integrated by networking technology, parallel and distributed system for providing higher processing capacity and mass data storage. Hence, users in cloud computing do not need to purchase any high capacity devices but just necessary to enjoy the benefits from cloud computing, which saves more costs.

The virtualized resource is not real but a virtualized physical resource pool by virtualization software like Virtual Machine. At present, the virtualized resource includes resource pools of computing, data storage, network, database and so on.

System management locates between provided services and server cluster, it is the major management system in cloud computing. The management includes authentication and authorization of clients, user catalog and security management as well as image establishment, deployment and so on.

Cloud computing makes a standard for provided services at the service interface, such as Service Level Agreement (SLA). Service interface is used for user registration, service searching and response user's request. It is the interface between cloud platform and clients.

For supporting the above mentioned cloud-oriented services and important applications (such as scientific computations, financial analysis, massive data processing and warehousing, as well as utility computing), Microsoft, Amazon, Google, Salesforce.com and other corporations have established large data centers around the world (Buyya et al., 2008). Microsoft, for example, established a 707,000-square-foot DCN building in Chicago, 2009 (Vahdat et al., 2010). There are 162 containers of 2,500 servers each with total 60 Megawatts of electricity in the building which costs 500 million US dollars. Apple data center in Maiden is established in 2010 with 500,000-square-foot and costs 1 billion dollars (Tarantino, 2012).

At present, the main approach for resource utilization is that service providers virtualize physical machines to multiple VMs (one-to-many), and integrate those VMs to a

resource pool via load balancing clusters (many-to-one). From the perspective of computing, we believe that the cloud should be a 'Super Computer' with thousands of cores, which is virtualized by a range of physical machines. Users then utilize several cores on demand.

2.1.2 Data Center Network

Data Center Network (DCN) is an important component of data centers that consists of a large number of hosted servers and switches connected with high speed communication links. A DCN enables the deployment of resources centralization and on-demand access of the information and services of data centers to users.

Since the advent of cloud computing era, data volume has been increased tremendously in the Internet. As cloud-oriented services and applications are data and communication intensive (a simple Web search request may need a cooperation among more than 1,000 servers), the information exchanging among remote nodes and local servers to proceed computation is increasing rapidly. Thereby, a great data traffic flow stress is taken to the data center and the limited inter-node communication bandwidth among servers is becoming serious bottleneck to DCN.

The motivations for building such cloud-oriented DCN are both economic and technical (Greenberg et al., 2009). Reasonable cost and elastic utilization according to the business requirements are considered for Information Technology (IT) investment of enterprises for DCNs. A cloud computing provider offers a large pool of high performance computing and storage resources that are shared among the end users. Users subscribe to the cloud computing services and receive computing and storage resources allocated on-demand from the pool. A number of enterprises still have concerns about the cloud computing service models, such as the network part of data center has not seen much commoditization and still uses enterprise-class networking equipment. The cost of us-

ing enterprise-class network equipment is large (upwards of $12 million per month for a 100,000 server data center) and is not suitable to accommodate internet-scale services in data centers. To be profitable, these data centers better use some lower-cost network equipment to achieve high utilization with agile end-to-end network capacity assignment and un-fragmented server pools (Greenberg, Hamilton, et al., 2008).

Table 2.1: A Comparison of Traditional DCN and Cloud-oriented DCN

Traditional DCN	Cloud-oriented DCN
Servers and software belongs to users, and infrastructure belogs to DCN provider	All equipments belong to DCN provider
Multiple management tools	Standardized management tools
Hosts a large number of relatively small/medium-sized applications which run on a dedicated hardware	Runs a smaller number of very large applications
Limited fault-tolerance or graceful degradation	Needs fault-tolerance or graceful degradation
Mixed hardware environment	Homogeneous hardware environment
Complex workloads for server installation	Simple workloads for server installation

Cloud-oriented Data Centers (CDCs) offer shared computing resource model with higher quality of service at the lower possible total cost of ownership. The main difference in CDC and traditional DC is “virtualization“ that allows for massive scalability, virtualized resources, as well as on-demand utility computing. Table 2.1 shows the comparison of Traditional DCN and Cloud-oriented DCN in features. Looking at the table, it becomes clear that the cloud-oriented DCN is simple to organize, operate, and it is more scalable. In a traditional DCN, servers are fixed in hardware and additional budget (such as hardware and the installation and maintenance) is required for upgrading and scaling up to more applications and users. In cloud-oriented DCN, by contrast, multiple servers are already in place. The virtualization is used to provide only the resources that a spe-

cific user demands, which gives cloud-oriented DCN a great scalability. In other words, the "cost" of cloud-oriented DCN is lower as compared to the traditional DCN. Whereas, the traditional DCN lacks of network bandwidth, scalability, and cost for coping with the increasing demands of tenants in accessing the services of CDCs.

It is common for a CDC to contain hundreds to thousands of servers in an economy of scale (Beloglazov & Buyya, 2010). A fundamental question for the DCN is how to effectively interconnect the number of exponentially increasing servers with a fault-tolerance, high available and significant aggregate bandwidth. The architectural design of DCN significantly affects its total performance. Therefore the design of a novel DCN architecture with the characters of scalability, low cost, robustness, and energy conservation is required.

2.2 Review on Data Center Network Architectures

In the recent years, the scale of the DCN is constantly increasing with the widespread use of cloud-oriented services and the unprecedented amount of data delivery in/between data centers. As the architectural design of DCN significantly affects its total performance, the DCN architecture has rapidly become research focus issue. In recent years, many DCN architecture related papers are published in the leading international journals in IEEE and ACM such as IEEE Computing in Science and Engineering, IEEE/ACM Transactions on Networking. Hence, it is necessary to take a review of the existing architectures for further research.

2.2.1 Taxonomy of Data Center Network Architecture

Current DCN architectures are classified into Clos/Tree, Variant Load Balancing (VLB), Hierarchical Recursive Architecture, and Optical/Wireless. Figure 2.5 shows the taxonomy of current DCN architectures.

The classification of the taxonomy is considered by the features of current DCN

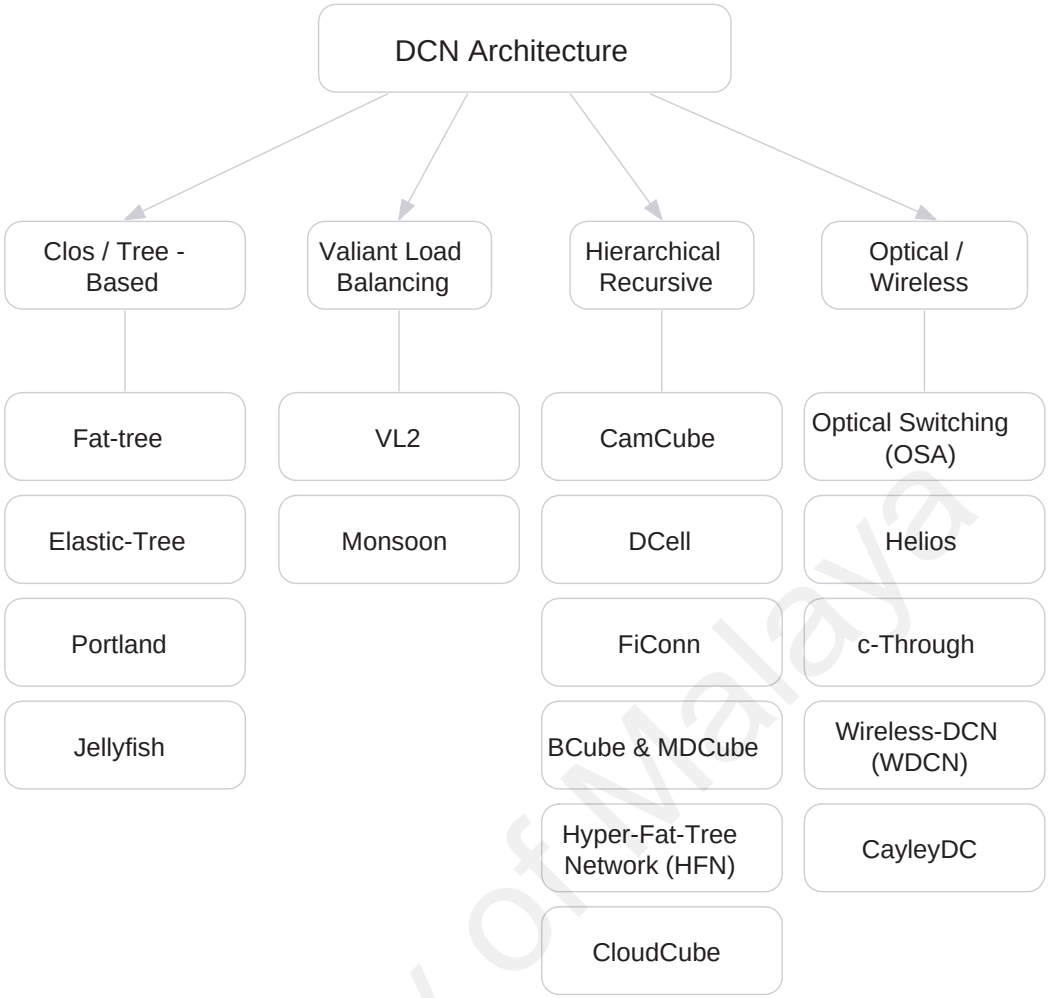

Figure 2.5: Taxonomy of Data Center Network Architectures

architectures. The Clos/Tree is to achieve high performance and high resource utilization by using commodity hardware in tree structure. VLB is distributes traffic across a set of intermediate nodes and leverages the random distribution of traffic into equal cost multi paths. The Hierarchical Recursive Architecture is proposed to avoid the existence of a single point of failure as well as to increase the network capacity. The Optical/Wireless architecture is established to include optical and/or wireless network.

In accordance with this taxonomy, each research trend with selected research perspective is introduced in the following sections. We note that although each research relates to multiple perspectives listed in Figure 2.5, we categorized them by selecting a key feature that shows the initial design motivations for each architecture.

2.2.2 Review on Data Center Network Architectures Using Taxonomy

This section reviews current DCN architectures on the basis of framework nature presented in figure 2.5. It also investigates the implications and critical features of the specific architectures.

2.2.2.1 Clos/Tree-Based

The Clos/Tree-Based related architectures are widely used in current DCN as the features of simple deployment and easy extension by increasing the rack and related switches.

Tree-Based

The traditional data center network is typical multi-root tree architecture, commonly composed by 3 layers switches (Infrastructure, 2007). In the architecture, the top layer as a root is called core layer, middle layer is aggregation layer, and the bottom layer is named as access layer. The higher layer devices own a higher performance and value. The core layer typically is comprised of several routers with redundancies accessing the external network in one side, implementing External Border Gateway Protocol (EBGP) or static routing protocol, and accessing to the internal network in another side, implementing Interior Gateway Protocol (IGP). The accessing layer switches commonly provides 1Gbps and 10Gbps downlink and uplink interface, respectively. The aggregation layer switches normally have 10Gbps interfaces and allow aggregating between access layer switches and forwarding data. The traditional hierarchical tree architecture may look similar to Figure 2.6.

In DCN, requests from the Internet are received by core layer router and forwarded to load balancing server in aggregation layer. The load balancing servers maintain a mapping table which includes Virtual IP address (VIP, for requests acceptance) and Direct IP address (DIP, for requests processing). According to the table, load balancing server forwards the Internet requests to application pool in accessing layer for processing.

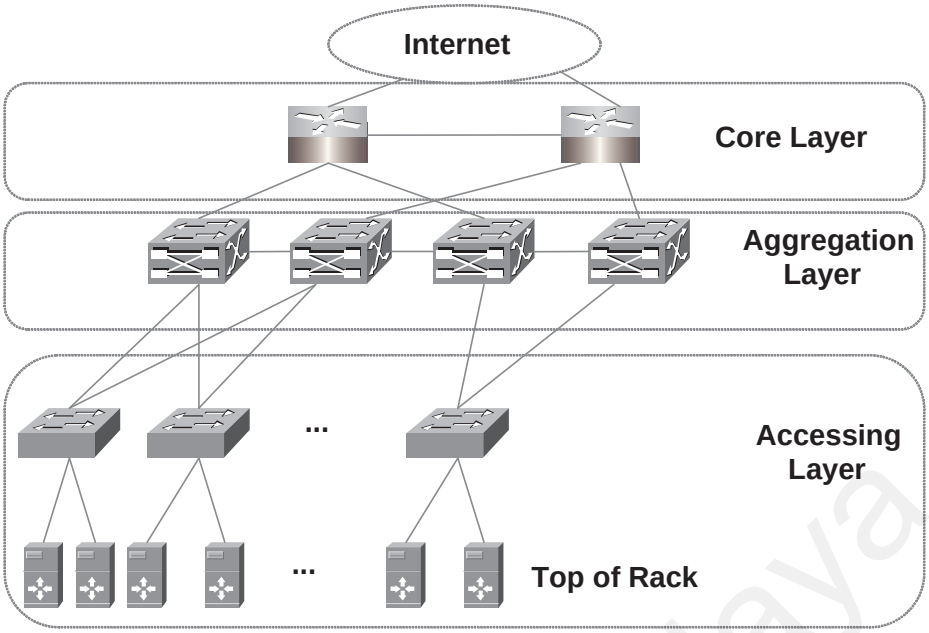

Figure 2.6: A sample of the tree hierarchical DCN architecture

Some shortcomings exist in the traditional tree architecture (Infrastructure, 2007). First of all, the bandwidth increases significantly near the root of the tree and deploying high performance network device in required, which may increase the cost. Secondly, network scale is severely limited by the switch port. Thirdly, the lower layer nodes will lose connection with others once the upper layer switch failure happens. Last but not the least, with the increasing of device processing capacity, there is not much doubt about data center power consumption will increase as well. Hence, researchers start to design alternative architectures for DCN.

Clos

Clos is an enhanced architecture on the basis of tree, and is widely used in many enterprise-class data center nowadays (Dally & Towles, 2004). The mathematical theory of clos was introduced by Charles Clos from bell labs in 1953 for creating a non-blocking, multi-stage topology, which provides higher bandwidth than what a single switch is capable of supplying (Clos, 1953). A main feature of the architecture is multi-layers switching wherein each switching unit connects to all units in the lower layer for reducing the num-

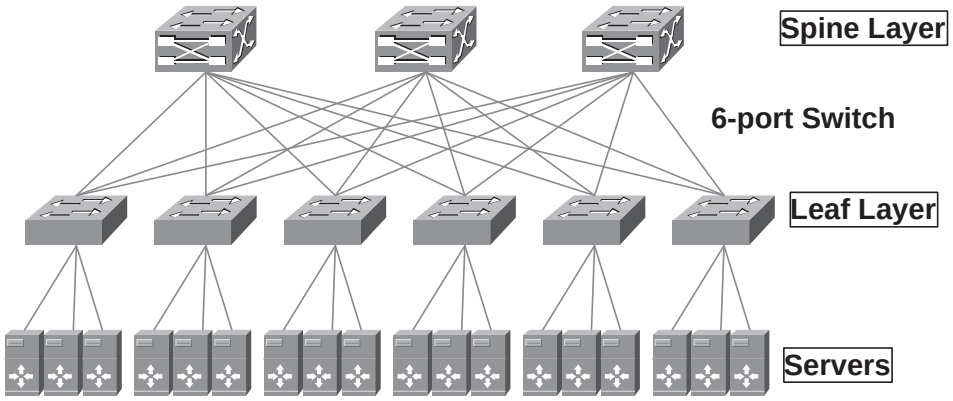

Figure 2.7: 3-Stage Folded Clos Topology (Dally & Towles, 2004)

ber of intersecting nodes since input and output streaming is increasing. Figure 2.7 shows an example of a 3-stage folded clos architecture.

In clos, the Leaf Layer is responsible for advertising server subnets into the network fabric. The Leaf layer determines oversubscription ratios, and thus the size of the Spine. The Spine layer is responsible for interconnecting all Leafs. As the clos is using a similar tree hierarchical data transmission mechanism, description is not necessary here. Though the multi-layers switching in clos effectively reduces the stress of bandwidth restriction in aggregation layer than the tree hierarchical, the same features and problems still exist between the two architectures.

The above Tree and Clos architectures are initially designed for small or medium scale networks. In the era of cloud computing, however, CDC is different from traditional enterprise class data center as new requirements are desired for large-scale distributed computing since the number of data center network device is growing rapidly. The problem of the widely used tree-based hierarchical architecture will be discussed in detail in chapter 3.

Fat-tree

To resolve the problems of network bottleneck and upper layer single node failure, Al-Fares introduced a Clos-based DCN architecture called Fat-tree (Al-Fares et al., 2008).

Similar with tree architecture, the switches in Fat-tree are also categorized into 3 layers, core layer, aggregation layer, and edge layer. Figure2.8 shows a classic Fat-tree architecture. In this architecture, a range of switches in a square are called Pod. In this diagram, there are k=4 switches in each pod and half of them belong to edge switches and half are aggregation switches. Similarly, aggregation switch uses each of k/2 ports while connecting to edge and core switches. Therefore, the maximum number of server in Fat-tree is K3/4, and 5K2/4 switches.

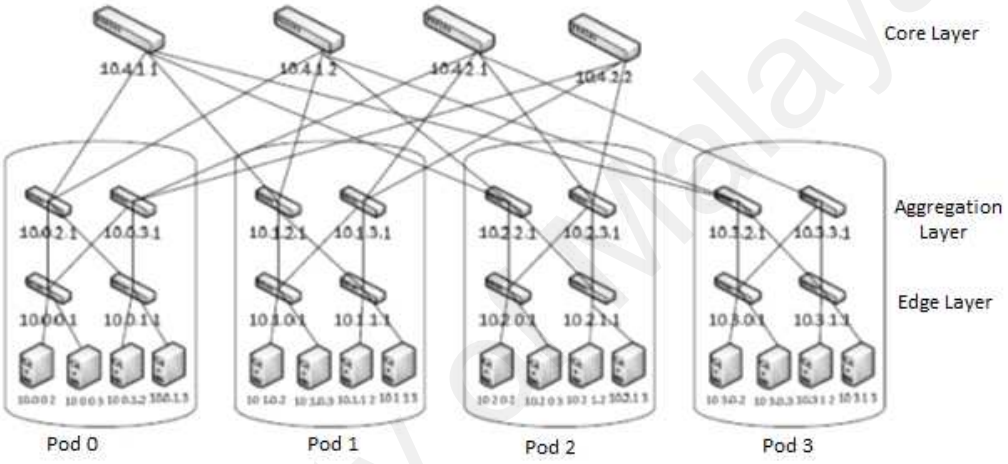

Figure 2.8: A sample topology of Fat-tree architecture(Al-Fares et al., 2008)

Fat-tree uses the 10.0.0.0/8 private range setting the interior DCN address and the format for pod switch is 10.pod.switch.1. The pod indicates a pod number ([0, K-1]) switch which means the position of switch in pod ([0, k-1], from left to right and bottom to top). The IP format for core switch is 10.k.j.i, where j and i show the coordinates of switches between core switches and aggregation switches (start from top-left). The host IP format is described as 10.pod.switch.id, where id means the host position in its own subnet.

Fat-tree architecture improves the cost-effectiveness by deploying a large number of low-cost switches with complex connections to replace the expensive and higher advanced switches in DCN. The equal number of links in different layers achieve non-

blocking communication among servers, which reduce the network bandwidth bottleneck. However, the scale of Fat-tree architecture is restricted by the number of device port. For example, a range of 48-port switches support maximum 27,648 servers only. Greenberg from Microsoft research points out that the Fat-tree architecture is very sensitive about the low-layer switch failure and it will impact the forwarding performance of DCN as Fat-tree is still a tree-based structure (Greenberg, Lahiri, et al., 2008).

Elastic Tree

Due to the uncertainty of data traffic in DCN, Heller points out that providing full bandwidth connection among all edge switches is not necessary. Hence, ElasticTree architecture is proposed from the perspective of power saving based on the Fat-tree architecture (Heller et al., 2010). The main feature of ElasticTree is on-demand turn on or off switches and connections.

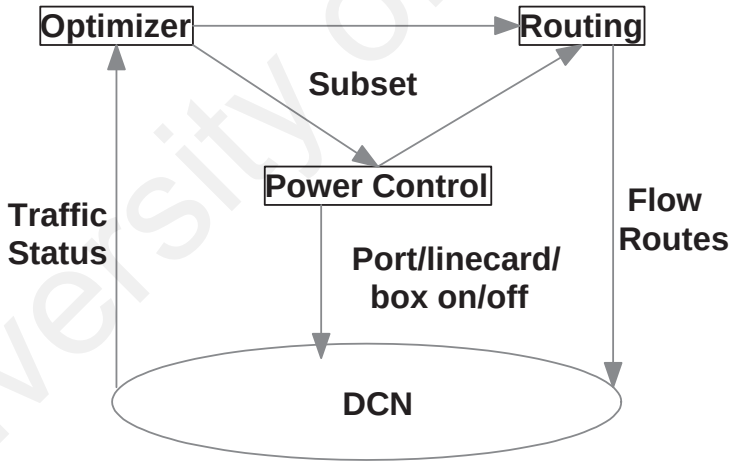

Figure 2.9: A sample of Elastic Tree Network architecture(Heller et al., 2010)

Elastic Tree consists of three logical modules, optimizer, routing, and power control. As shown in Figure 2.9, the Optimizer responds to find the minimum power network subnet which satisfies current data flow conditions. Its input are network topology, data flow matrix, power model for each switch and the desired fault-tolerance properties. The optimizer outputs a set of active components to power control and routing modules. Power control module toggles the power states of ports, adapters, and entire switches. The rout-

ing module provides route to data flow.

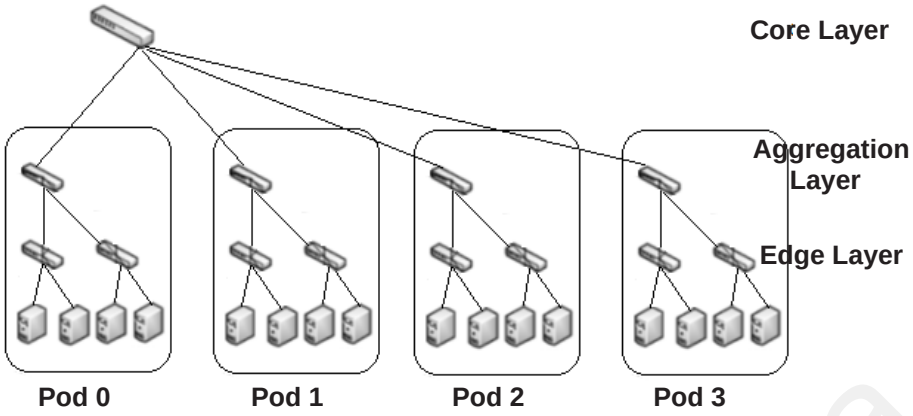

Figure 2.10: A sample topology of Elastic Tree DCN architecture(Heller et al., 2010)

ElasticTree is designed for power saving in DCN, which effectively reduces the maintaining cost for data center. However, due to the ElasticTree is deployed based on Fat-tree, it still inherits the same problem. Figure 2.10 shows the architecture of Elastic-Tree.

Portland

Based on the Fat-tree architecture, Mysore proposed a scalable and fault-tolerance 2-layer routing fabric (Niranjan Mysore et al., 2009). PortLand employs a fabric manager and Pseudo MAC address (PMAC) to forwarding data packet, and MAC to PMAC mapping to avoid modification in servers.

PortLand edge switches learn a unique pod number and a unique position number in each pod. Location Discovery Protocol is employed to assign these values. For all directly connected hosts, edge switches assign a 48-bit PMAC. The format of the PMAC is pod.position.port.vmid, whereas pod (16 bits) indicates the pod number of the edge switch, position (8 bits) reflects the switch position in the pod, port (8 bits) and vmid (16 bits) describe the port number of the host connects to and multiple VMs on the same physical machine (PM), respectively.

Whenever a source host desires to communicate with another, it searches the target

PMAC through the fabric manager. Once data packets research to the destination node, the ingress switch modifies the PMAC to actual MAC (AMAC) of the target. Upon completing VMs migration from one PM to another, the fabric manager maintains the new PMAC to AMAC mapping and broadcasts to the previous PM which VMs located before.

PortLand deployed a new routing mechanism in 2-layer based on the Fat-tree architecture, which supports a better fault-tolerance routing and forwarding, VMs migration, as well as network scalability. However, modification of the existing switches is required to meet the above features. In addition, as the fabric manager plays a major role in PortLand, the risk of single node failure still exist in this architecture.

Jellyfish

The architectures introduced before (Al-Fares et al., 2008)(Heller et al., 2010)(Niranjan Mysore et al., 2009) are the improvement of the tree hierarchical structure, and some common disadvantages exist among all of them. For example, network scale is restricted by the number of core routers, weakness in switch failure recovery, one-to-many and many-to-many communications, as well as cloud computing. The DCN architecture therefore trends to a flat structure to modify the network structure from 3 layers to 2 layers or even 1 layer – Mesh structure, such as Jellyfish (Singla et al., 2012).

Jellyfish constructs a random graph topology at the ToR switch layer, and each ToR switch i has number of k_i ports, of which using r_i to connect to other ToR switches and the remaining $k_i - r_i$ ports to servers. In the simplest case, each switch has same number of ports and servers, thus $k = k_i$, $r = r_i$. When N is the number of ToR Switches, total N_{kr} servers can be supported in DCN. Figure 2.11 shows a topology of Jellyfish architecture.

The authors in (Singla et al., 2012) point out that when the number of servers is less than 900, Jellyfish supports more than 27% servers than Fat-tree, performance improves with the network scale increasing. Jellyfish also has a shorter average path length and

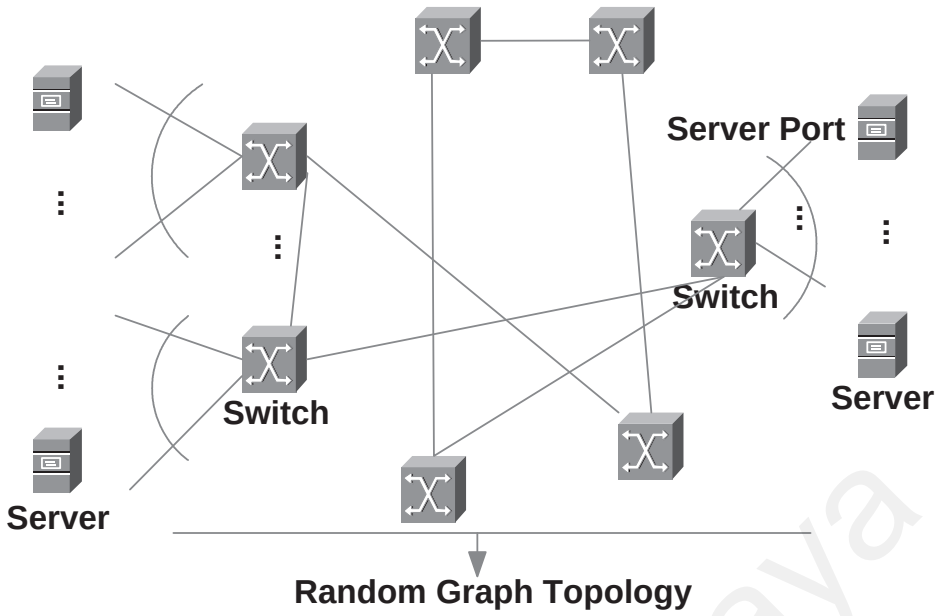

Figure 2.11: A sample topology of Jellyfish architecture(Singla et al., 2012)

higher bandwidth capacity than Fat-tree, and a better performance in power saving. However, as Jellyfish is a random regular graph structure, the cabling layout issue is a big challenge which limits the positions among the ToRs, and the implementing of optimal routing is also a challenge as well.

2.2.2.2 Valiant Load Balancing

The Valiant load balancing(VLB) architecture was initially introduced by L. G. Valiant for processor interconnection networks (Valiant, 1990), which is approved with capacity for handling traffic variation. VLB can achieve a hotspot free fabric for DCN when random traffic into multi paths.

VL2

VL2 is another tree-based architecture introduced by Greenberg in 2009 for resource allocation dynamically in DCN (Greenberg et al., 2009). The difference with Fat-tree is that VL2 connects all severs through a virtual 2-layer Ethernet, which is located in a same LAN with servers. In this case, as shown in Figure2.12 all servers are assignable to upper layer applications as no resource fragmentation happens. VL2 uses the Clos

topology to increase connections, and VLB mechanism to assign routing for load balancing. Moreover, VL2 implements Equal-Cost Multi-Path routing (ECMP) to forward data over multiple optimal paths and resolve the problem of address redistribution in VMs migration. Therefore, the VL2 is considered in the VLB category.

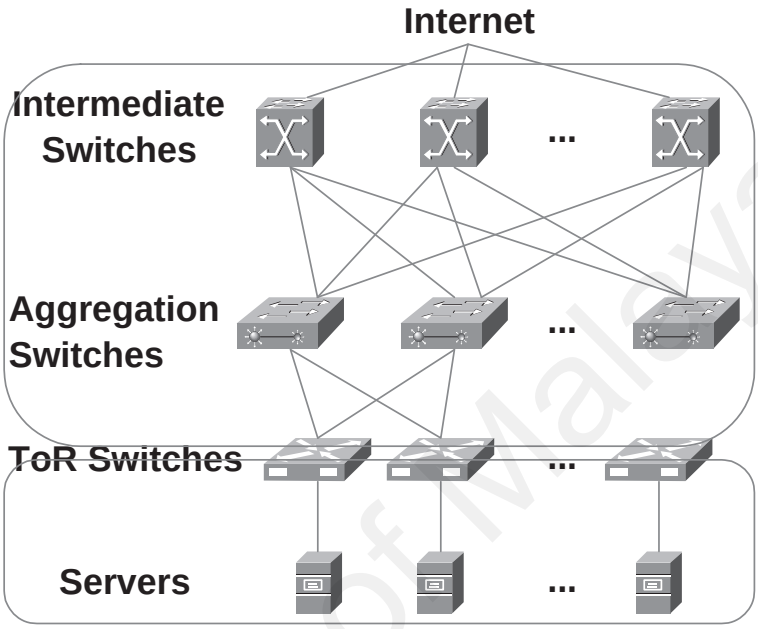

Figure 2.12: A sample topology of VL2 DCN Architecture(Greenberg et al., 2009)

Since VL2 follows the traditional tree architecture in connection, it is widely used for enhancing the existing DCN. However, its network reliability is not improved and still has problems in scalability and single node failure.

Monsoon

The architecture of Monsoon (Greenberg, Lahiri, et al., 2008) is described in Figure 2.13, where over 100,000 servers are linked in a 2-layer network without over subscription. The core border router and accessing router in layer 3 uses ECMP for multi-path transmission, and VLB mechanism for load balancing like VL2.

Monsoon uses a MAC-in-MAC technology to create MAC layer tunnel, modifies the traditional Address Resolution Protocol (ARP) to a user mode process, and allows a new mac interface to forward encrypted Ethernet frames. These mechanisms and solutions,

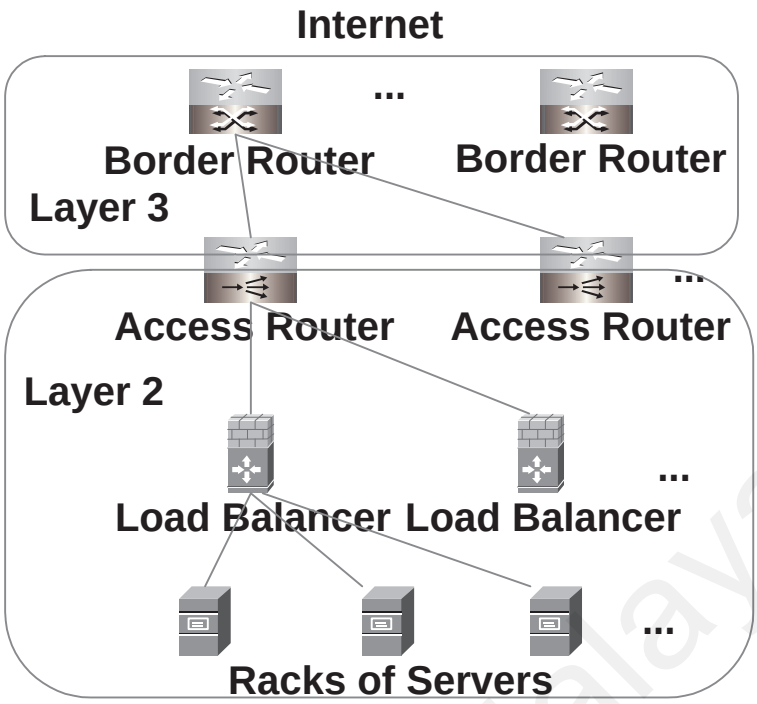

Figure 2.13: A Sample Topology of Monsoon DCN Architecture(Greenberg, Lahiri, et al., 2008)

however, are not compatible with the existing Ethernet architecture.

2.2.2.3 Hierarchical Recursive

Hierarchical recursive architecture is generally appropriate in order to avoid the bottle-neck of single point failure and increase network capacity.

CamCube

CamCube is a non-switch architecture presented by Libdeh, which constructs network with a 3D torus topology by each server directly and connects with two neighbor servers in 3D directions (Abu-Libdeh et al., 2010). The topology of CamCube is shown in Figure 2.14.

CamCube assigns an (x, y, z) coordinate to indicate the position for each server in the topology, and provides functionality to send or receive packets to and from one-hop neighbors. CamCube provides a platform for developers to create a more efficient routing algorithm for API according to the requirement, which decreases the additional network

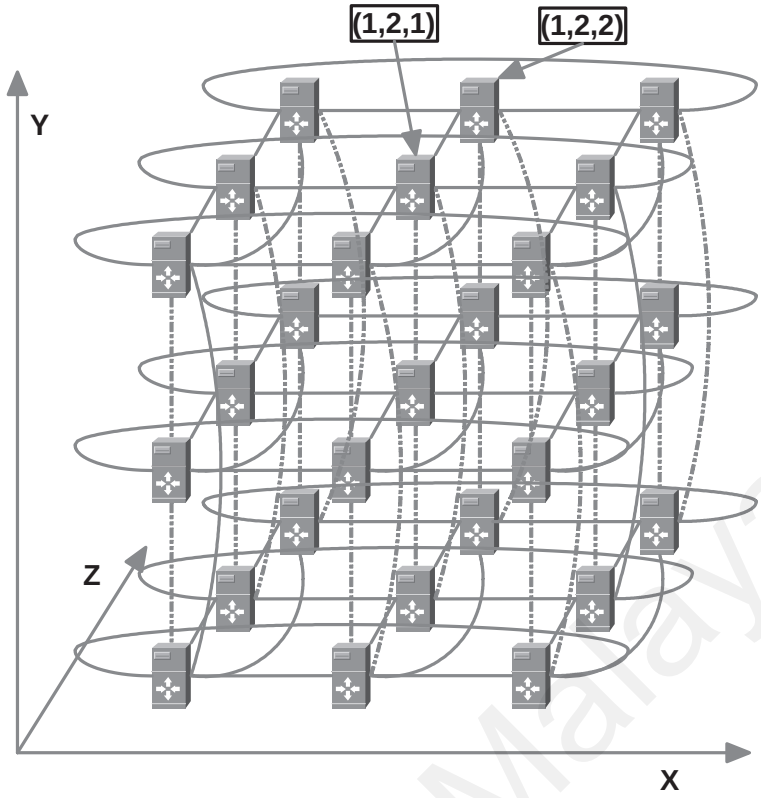

Figure 2.14: A sample topology of CamCube architecture(Abu-Libdeh et al., 2010)

performance overhead and verified the efficiency of this design.

CamCube has a simple structure and connection, as well as a high link redundancy. There is no bandwidth bottleneck in specific node as this is not a tree-based structure. However, those servers act a role of switch to forward data, which consume part of the server's computing resources and reduce the computing efficiency of servers. In addition, the number of network adapters installed in each server is limited (2 adapters for each server commonly), which means the size of CamCube network is also limited.

As CamCube has a relatively long routing path in a torus (O*$N^{1/3}$ hops), which causes decrease of performance and increase of cost of DCN, Popa introduced a De Bruijn-based DCN architectures in (Popa et al., 2010) that servers within a rack are labeled and connected as a De Bruijn graph structure. Those servers with same label but in different racks are also connected as a De Bruijn structure. As the diameter of De Bruijn is (log N), the result shows that it has a better routing performance and lower cost contrast

with CamCube structure.

The approach of using recursive structure in DCN architectures design is widely achieved in DCell (Guo et al., 2008), FiConn (D. Li et al., 2009), BCube (Guo et al., 2009), MDCube (H. Wu et al., 2009), and HFN (Ding et al., 2012), which reduces the bottleneck in core layer routers, and provids multiple paths in pair of servers.

DCell

DCell (Guo et al., 2008) is a recursively defined network architecture shown in Figure 2.15. $DCell_0$ is a basic unit to construct larger DCells, which consists of n servers and a mini-switch. If there are m servers in $DCell_k$ network, the $DCell_{k+1}$ is considered as a compound graph structure consists of the number of m+1 $DCell_k$.

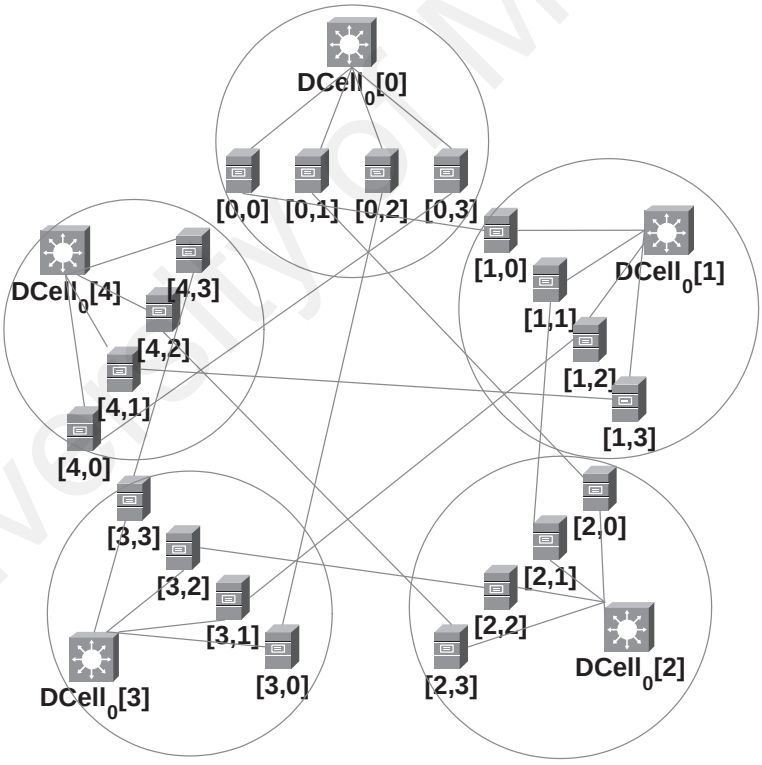

Figure 2.15: A sample topology of DCell architecture(Guo et al., 2008)

DCell uses a distributed routing algorithm called DCellRouting for data forwarding. According to the destination node and the relationship of server and virtual nodes, data packet is forwarded to next hop automatically without routing table searching in server. The mass redundancy links in DCell make a higher bandwidth than in tree-based structure

and have a better performance at one-to-all and all-to-all communication model in data-intensive computing. The situation of server, link, and rack failure has been considered in DCell design. Data packet can also be delivered to destination node through fault-tolerance path when a failure is detected by server or switch. In addition, DCell uses local reroute, local link-state, and jump-up mechanism to address the above failures. As the routing algorithm in DCell network is running between layer 2 and layer 3, the exciting TCP/IP protocol and based applications can be deployed in the structure seamlessly and effectively.

One of shortcomings of DCell architecture is more interfaces and ports are desired to extend network size. Furthermore, the lower levels servers undertake more forwarding tasks and this load balancing is challenging issue to deal with in the future. Nevertheless, the proposed DCell architecture indicates a novel thinking in DCN, which has been a milestone in the relevant research.

FiConn

A common commercial server typically has two network adapters, one for data receiving and forwarding, and another one for redundancy. To reduce the additional overhead caused by the mass redundancy links, Li introduced a modified DCell structure, called FiConn in 2009 (D. Li et al., 2009). Similar with DCell, FiConn uses compound graph creating its FiConn structure. In a 4 levels FiConn with 16 ports switch, the number of servers can reach to 3,553,776. Figure 2.16 shows the 1 level FiConn architecture.

FiConn decreases the overhead of network establishing by decreasing some performance. Contrast with DCell, the fully connections among virtual nodes in same level are not required in FiConn, it only uses idle ports of servers and switches to connect with other devices, which decreases the number of redundancy links and the network adaptors on server. Therefore, multi network adaptors are no longer to be installed in server and the number of port requirement for higher level switch is reduced, which means the cost

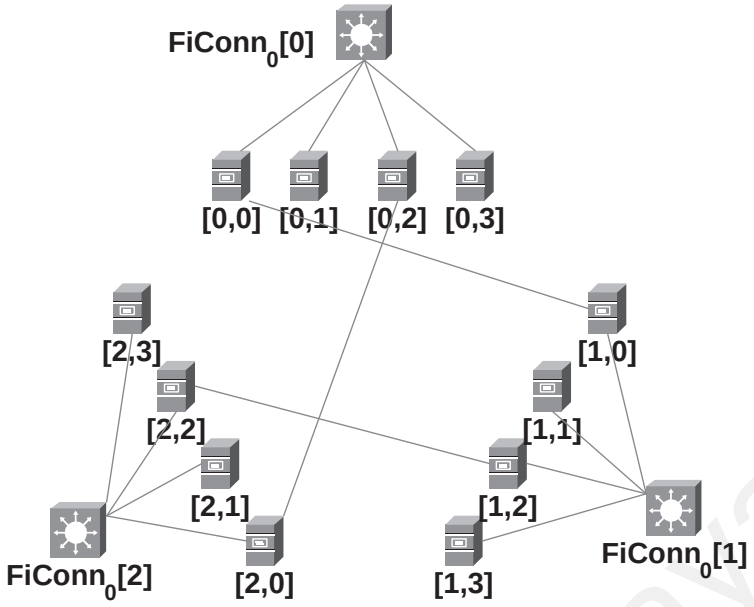

Figure 2.16: A sample topology of FiConn architecture with level 1(D. Li et al., 2009)

for DCN establishment is decreased as well.

BCube and MDCube

In 2009, (Guo et al., 2009) proposed a hypercube related structure of data center network, named as BCube and the structure is shown in Figure 2.17. Similar with the recursive defined character of DCell structure, $BCube_0$ is constructed by n servers connecting to an $n-port$ switch, and $BCube_1$ is constructed from $nBCube_0$ connecting to n switches. More generically, a $BCube_k$ is constructed from n $BCube_{k-1}$ connecting to n^k $n-port$ switches. Each host has $k+1$ parallel paths with different lengths. It is easy to see that a $k-level$ BCube structure, $BCube_k$ has n^{k+1} servers and $n^k(k+1)$ mini-switches. Each host has k+1 parallel paths in BCube but the lengths are different. BCube also makes one-to-X speedup for data replication, and such speedup depends on the number of network adapters.

One of the design goals for BCube is to establish shipping-containers DC called Modular Data Center (MDC), but how to connect these data centers and create a larger data center are the main goal for MDCube (H. Wu et al., 2009). MDCube is proposed as an interconnection structure among shipping-containers by fiber to construct larger size

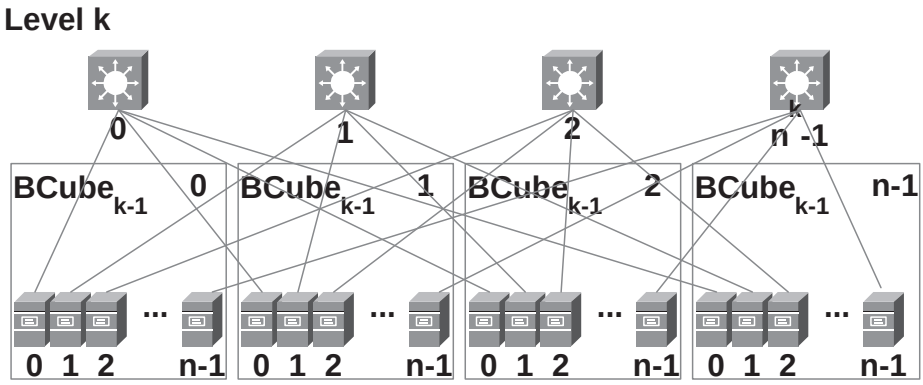

Figure 2.17: A sample topology of BCube architecture(Guo et al., 2009)

of DCN. In MDCube, each BCube container is assumed as a virtual node, and connecting with other node to create a HyperCube network structure.

In BCube, servers have multiple ports to support selectable routing, high fault-tolerance and high throughput. Therefore, BCube has better performance in one-to-many, and many-to-many communication and resolves load balancing issues in lower lever servers. However, the number of switches in BCubek is k times than DCellk for connecting a certain number of servers, and thereby, more cost in cabling layout and others in construction than DCell.

Hyper-Fat-Tree Network (HFN)

To optimal DCN for some specific requirements are desired in cloud computing, such as the following architectures HFN (Ding et al., 2012) and CloudCube (Model et al., 2009) are proposed for MapReduce optimization.

$HFN_{0(N,M)}$ is the basic building block of the entire network topology, which consists of n master servers, $n \times m$ worker servers, and number of n "m-port" switches. Each switch connects m worker servers, n master servers and n "m-port" switches to create two-vertex sets of the bipartite graph. More generically, the level $k+1$ HFN, HFN_{k+1} consists of n HFN_k and $n \times k+1$ "n-port" switches. If all the HFN_0 are considered as virtual servers, it is obviously that the basic architecture of HFN is from BCube. A sample

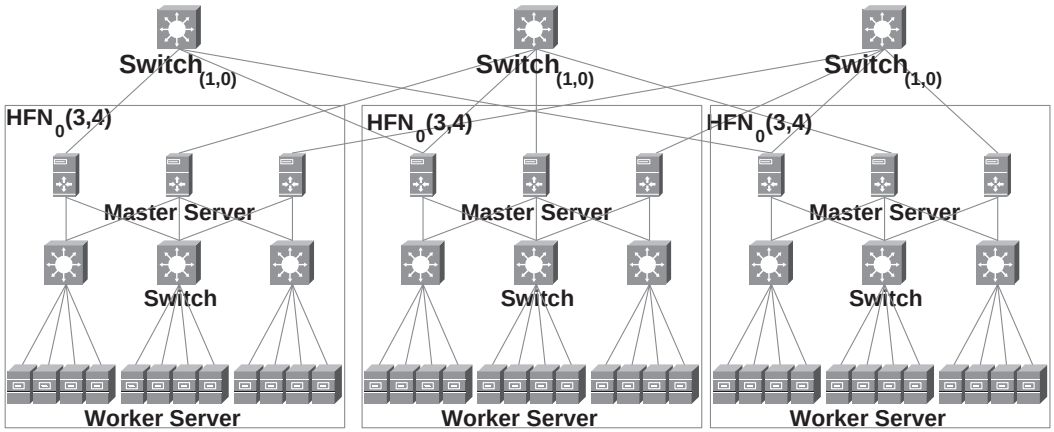

Figure 2.18: A sample topology of HFN architecture with n=3, m=4(Model et al., 2009)

topology of HFN is shown in Figure 2.18.

In HFN, master servers control the entire procedure of MapReduce and receive tenant's requests. The server assigns a task to multiple master servers and further forward to worker servers to execute under the master servers' control. The worker server sends the number of tasks to be performed to its master servers once the worker server completes its job, and the masters may assign a new job according to the schedule. The experiment shows that MapReduce has a better performance in HFN than the tree hierarchical architecture.

CloudCube

As the basic structure of HFN is generated on the basis of BCube, a large number of switches are required in network establishing. To resolve this problem, CloudCube (Formu, 2009) is proposed based on the architectures of HFN and BCube, which owns a same structure with $HFN_{0(n,n)}$ if considering the $CloudCube_{0(n)}$ as virtual servers. By contrast, CloudCube interchanges the positions of switches and servers in BCube to create a $CloudCube_{k(m,CloudCube_{0(n)})}$ architecture, where m denotes the number of switches connected by $CloudCube_{0(n)}$, and commonly, $m = n$. The number of potential server in CloudCube is much more than HFN, which effectively reduces the cost and enhance the scalability of DCN.

2.2.2.4 Optical/Wireless

The approach of using optical and wireless network in DCN is introduced in this section.

Optical Switching Architecture (OSA)

(K. Chen et al., 2012) believe that if the network is able to dynamically change its topology and link bandwidth, then unprecedented flexible architecture can be supported in DCN. Thereby, they introduced a novel Optical Switching Architecture for DCN called OSA, that uses Optical Switching Matrix (OSM), Wavelength Selective switch (WSS) and Wavelength Division Multiplexing (WDM). Figure 2.19 shows the OSA architecture.

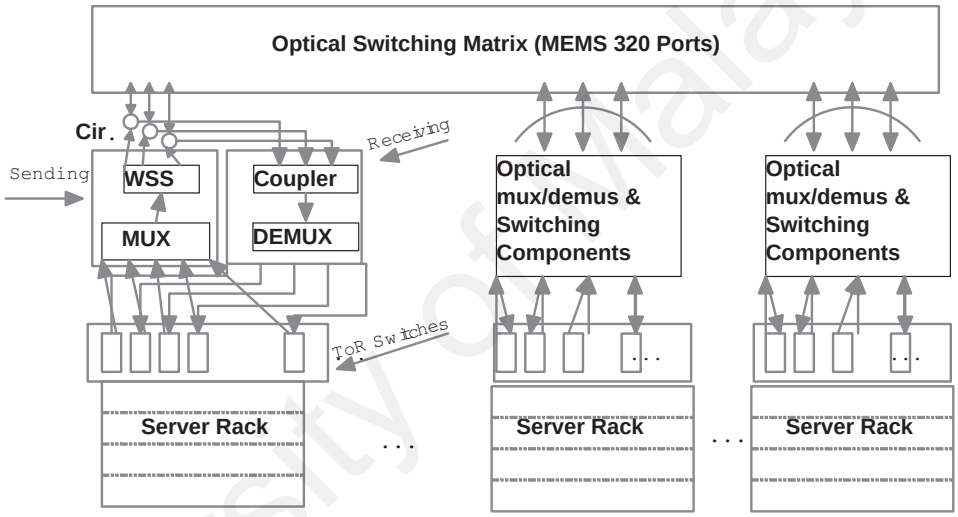

Figure 2.19: A sample topology of OSA architecture(K. Chen et al., 2012)

Most OSM modules are bipartite $N \times N$ matrix where any input port connects to any one of output ports. Nowadays, the Micro-Electro-Mechanical Switch (MEMS) is widely used in OSM to reconfigure a new input/output matching and connection within 10ms by mechanically adjusting a microscopic array of mirrors. The WSS is a $1 \times N$ switch consisting of one common port and N wavelength ports for partition the set of wavelengths coming through the common port among the N wavelength ports.

OSA employs shortest path routing scheme and hop-by-hop switching to ensure the network wide connectivity in DCN. Each single hop converts the forwarding data in fiber from optics to electrical signals and then back to optics (O-E-O) and switching at the

ToR. In addition, a central OSA manager responses for topology management, traffic and routing estimation and configurations.

Helios (Farrington et al., 2011) and c-Through (G. Wang et al., 2010) are well-known hybrid electrical-optical structures. In this hybrid model, each ToR connects to an electrical and an optical network at same time. The electrical network is a 2 or 3 layers hierarchical tree structure with a certain oversubscription ratio. In the optical network, each ToR maintains a single optical connection to other ToRs, and this optical connection is unrestricted capacity.

Optical switching has a better potential performance than node switching in data transmission speed, flexible topology, power-saving and bits ratio in long distance forwarding (Ikeda & Tsutsumi, 1995). Moreover, optical switching generates less heat to reduce the maintenance cost in cooling and radiating. Therefore, optical switching in DCN is an important research topic in DCN.

Wireless-DCN (WDCN)

Wireless technology can flexibly change network topology without re-cabling layout, thereby, (Ranachandran et al., 2008) operate wireless technology into DCN in 2008. Later on, (Kandula et al., 2009) describe the Flyways architecture to de-congest and reduce data forwarding time between ToR switches in DCN. However, the separated wireless network is hard to meet all the requirements about DCN such as scalability, capacity and fault-tolerance. For example, the bandwidth of wireless network is commonly limited due to high traffic load and interference. (Cui et al., 2011) proposed hybrid Ethernet/wireless architecture in DCN called WDCN.

To avoid excess antenna using and interference, (Cui et al., 2011) consider each ToR as a wireless transmission unit (WTU) in WDCN that is shown in Figure 2.20. Using 60Hz wireless communication technology, (Shin et al., 2012) proposed a fully wireless connection DCN, integrating switching fabric into server nodes to reduce the actual dis-

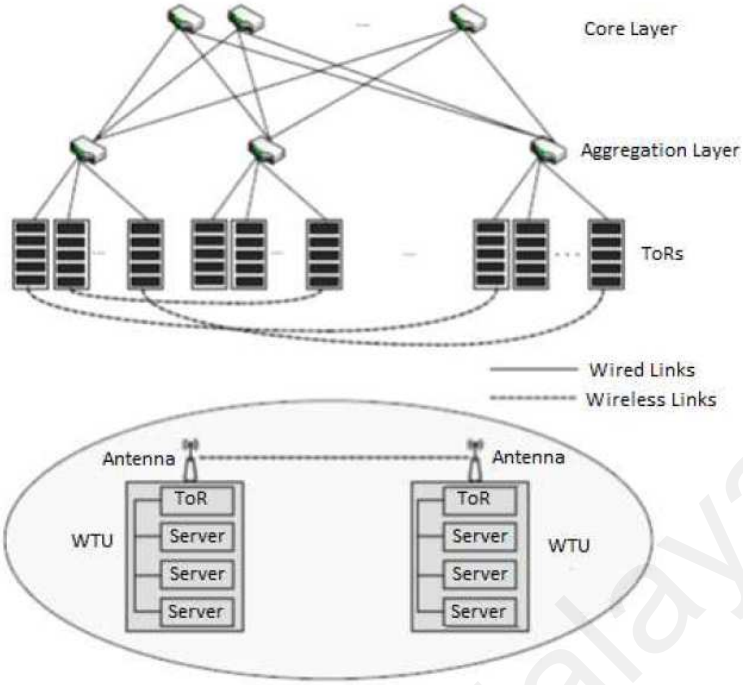

Figure 2.20: A sample topology of WDCN architecture(Ranachandran et al., 2008)

tance between ToRs and support fault-tolerance. They also replace the network interface card (NIC) of a server to a Y-switch, and deploy these servers to circular structure racks. The above approaches can easily establish communication channels between and interior of racks, and create a mesh network structure. As this mesh network is a kind of Cayley graphs (Alon & Roichman, 1994), thereby it also called Cayley Data Center (CayleyDC).

Deploying wireless connection makes the network topology flexible and decreases the complex cabling layout. However, with a certain bandwidth, the forwarding distance in wireless network is limited and more overhead is generated due to broadcasting.

2.3 Comparison of Data Center Network Architectures

This section presents a comparison and analysis of these architectures from the six criteria: scale, bandwidth, fault tolerance, scalability, overhead, and cost of deployment. Scale, fault tolerance, and scalability are important design concerns for the reason that DCN consists of large amount and continuous increasing number of servers and network equipment. The scale of cloud-oriented DCN is much bigger than that of traditional

DCN. For managing such number of devices, scheduling decentralized resources, as well as supporting different range of cloud-based applications and services, the DCN architecture with fault tolerance and scalability are highly desired.

In this section, scale refers to the number of servers that are supported by the existing architecture. Fault-tolerance refers to whether the proposed architecture can effectively deal with the problems of server, switch, and link failure. Scalability refers to whether the selected architecture exists centralized nodes, single point of failure, and easy to expand by deploying more devices. In this comparison, bandwidth indicates a proportion of actual bandwidth to maximum theoretical bandwidth between servers. Normally, a higher oversubscription refers to less parallel paths (lower bandwidth) and vice versa. Overhead and cost of deployment are also important facts in DCN design as reasonable cost according to the business requirements are considered for IT investment. In this section, overhead refers to the number of switches and links, as well as their cost. Cost of deployment refers to the workload of switch and server configuration, and the construction of basic equipment.

Table 2.2 gives the comparison of the proposed DCN architectures from the above aspects.

Network scale. Clos/Tree-based, VLB and Optical/wireless architectures connect servers and switches to establish a simple and easy connecting hierarchical tree-based topology. Servers take full part in data processing only. In Hierarchical Recursive architectures, by contrast, each server is installed with one or more extra network adapters to establish flexible, complicated and specific network architecture. Servers not only process data but also participate in data transmission. In DCell, for example, servers are fully connected in identical layers that makes it more scalable than tree-based architecture. The deployment, however, is a complicated mission for DCell as the significant cabling layout. It is worth to mention that, the network scales of OSA and WDCN are

Table 2.2: A comparison of the proposed DCN architectures

Architectures	Scale	Bandwidth	Fault-tolerance	Scalability	Overhead	Cost of deployment
Tree hierarchical	small	low	bad	bad	very high	very high
Fat-tree	medium	medium	medium	medium	high	high
ElasticTree	medium	medium	medium	medium	high	very high
PortLand	medium	very high	good	medium	high	high
Jellyfish	large	very high	good	good	medium	high
VL2	large	very high	medium	medium	high	high
Monsoon	large	very high	medium	medium	high	high
CamCube	large	high	good	good	very high	high
DCell	large	high	good	good	high	high
FiConn	large	high	good	good	medium	high
BCube	small	very high	very good	high	high	high
MDCube	large	high	good	good	very high	high
HFN	small	medium	medium	good	low	medium
CloudCube	small	medium	medium	good	low	medium
OSA	small	very high	bad	medium	high	medium
WDCN	small	very high	good	medium	medium	medium

smaller than others due to the higher deployment cost and limited wireless transmission range in optical/wireless architectures.

Bandwidth: The initial goal of the proposed architectures design is to resolve the bandwidth bottleneck in DCN. Therefore, no matter VLB, hierarchical recursive, optical/wireless, or even the improved tree-based (such as Fat-tree and Jellyfish) architectures, all are higher in bandwidth as compared to the original clos/tree-based architectures. In addition, optical switches are advanced at bandwidth, loss rate, as well as the ultra-higher data forwarding speed by comparing with electrical switches.

Scalability: Clos/tree-based architectures expand the scale of DCN by adding a number of ports and levels on switches. They presents the advantages of ease-of-wire but is limited by poor scalability and fault-tolerance. The improved tree-based architectures, such as Fat-tree and VL2 solve the problems by increasing the number of switches in aggregation layer but the cabling layout become much more complex. By contrast, hierarchical recursive architectures have a limited scale of network size as the number of network adapters installed on a server is limited.

Overhead: Clos/tree-based and VLB architectures utilizes switch and router to forward data, and server is only concerned with data processing and storage. For effectively using lower level resources, those high level switches and routers are desired to have a better data processing capacity and higher bandwidth. As servers participate data forwarding in hierarchical recursive architectures, part of CPU and memory resources of servers are consumed.

Cost of deployment: As hierarchical recursive architectures employ server to transfer data, it has a lower cost than the tree-based architectures in the same cost/performance conditions. In addition, the cost of hierarchical recursive deployment may reduce with the development of CPU and network adapter technical, such as integrating a module in CPU to process networking related task, or improving the autonomy process capacity of network adaptor to forward data without CPU. By contrast, in optical/wireless and tree-based architectures, higher cost is generated as additional fiber optic or electro-optical transmitters/receiver, and high advanced routers and switches are required for supporting higher network performance in DCN.

Existing DCN architectures are all fixed (K. Wu et al., 2012). They are advanced in one or more network evaluation metrics but may not support sufficiently in others. It is still difficult to decide which architecture will perform the best and whether it is suitable in a specific DCN as features.

2.4 Open Issues and Challenges for Cloud-Oriented Data Center Network Architecture Design

In the recent years, the scale of the DCN is constantly increasing with the widespread use of cloud-oriented services and the unprecedented amount of data delivery in/between data centers, whereas the traditional DCN architecture lacks of aggregate bandwidth, scalability, and cost for coping with the increasing demands of tenants in accessing the services of cloud data centers. Therefore, the design of a novel DCN architecture with the features of scalability, low cost, robustness, and energy conservation is required. With the analysis of the existing variable DCN architecture designs, we identify the following open issues and challenges that can be the subject of future cloud-oriented DCN architecture design.

2.4.1 Deployment Cost and Energy Consumption

Data center with different network architectures can accommodate different number of servers and switches. When the number of servers in DCN reaches tens of thousands or more higher, different network architectures result in a huge data center deployment costs. For this reason, reducing the DCN deployment cost is seen by operators as a key driver for reaching high cost/performance ratio and maximizing DCN profits.

In addition, the energy conservation is emerging as an increasingly important global consensus issue which is pointed out as an amortized cost in (Greenberg, Hamilton, et al., 2008). The energy cost reaches 15% of the total cost of a data center. Based on a report submitted to Congress by the U.S. Environmental Protection Agency as part of the Energy Star program, networking devices in data centers in the United States accounted for 6.5 billion kWh/year in 2012 (USEPA, 2012). How to build green and low consumption of data centers become a serious research issue. The consumers of energy in data center include servers, networking equipments, power distribution and cooling facilities. Most approaches (Beloglazov & Buyya, 2010)(Y. Chen et al., 2012)(Lee & Zomaya, 2012)(Boru et al., 2013) effort focus on making servers and cooling infrastructures more

energy efficient. In contrast, the energy consumed by networking equipments is rarely considered, because networking equipments take up a relatively smaller proportion of data center's energy budget. As servers and cooling within data centers become more energy efficient, the percentage of data center power consumed by networking equipments is expected to grow. Moreover, reliable power supply providing for a large-scale DCN needs more budget.

2.4.2 Network Optimization

Bandwidth utilization and cabling complexity are becoming significant factors in the novel network architecture design. For example the metric of bisection bandwidth is widely used in DCN performance evaluation (Al-Fares et al., 2008)(Guo et al., 2008)(Guo et al., 2009)(Katayama et al., 2011), and the throughput in aggregate layer measures the sum of the aggregated data flows when network broadcast is conducted in tree-based DCN architectures. In traditional DCN, the cabling layout is simple. In cloud-oriented DCN, however, cabling is a critical issue as massive number of nodes that has an impact on connecting efforts and maintenance. In addition, the issue of designing optimized network structure for particular applications to increase its competitiveness in the era of cloud computing is also yet to be explored.

2.4.3 The Novel Network Architecture Studies

Network architecture in distributed system has been studied extensively, and researchers propose a number of network structures (Frécon & Stenius, 1998)(Tennenhouse & Wetherall, 2002)(Foster et al., 2002)(Lian et al., 2002). In DCN, the deployment of the existing mature network architectures need to be analyzed and validated, especially those in the model of the server-centric (Guo et al., 2009). Novel network architectures for cloud-oriented DCN are also expected in further research.

2.4.4 Quality of Service in Upper Layer

DCN architecture indicates the relation of connection between the servers in the data center, which is the order of intermediate nodes links. The systems mentioned earlier such as GFS and HDFS are achieved as the form of parallel and distributed through collaborative communication between a large numbers of servers in DCN. The quality of implementing these systems directly affects the quality of service to end user.

2.4.5 Congestion Control

Cloud-oriented DCNs adopt TCP and Ethernet as their layer-4 and layer-2 transmission technologies. However, the broadcast nature of data transmissions in Ethernet causes significant traffic congestion which makes the TCP retransmission mechanism unworkable, especially when a large number of lost packets occurs.

TCP-Incast is an unique phenomenon observed in some cloud computing applications, such as MapReduce and cluster-based storage system (Y. Chen et al., 2009). For instance, consider the example multiple servers simultaneously communicate with a single client as a scenario, large number of packets are dropped as the switch buffer overflow. Then the application throughput decreases rapidly due to the packets loss and TCP retransmission timeout (RTO). RTO may degrade application throughput up to 90%.

2.4.6 Load Balancing/Flow Scheduling

The purpose of load balancing in cloud-oriented DCN is to distribute workload to network equipments fairly by routing traffic across multiple paths. As mentioned in Section 2.2, the CDC architectures, such as Fat-tree (Al-Fares et al., 2008) and Clos (Dally & Towles, 2004) network, often use densely multi-path topologies to provide large bandwidth for internal data exchange. In such networks, it is critical to employ effective load balancing schemes for fairly utilizing network resources.

In private or traditional data centers, workload patterns are relatively predictable.

Typically, routing in such environment is based on the shortest path algorithms, for example, open shortest path first (OSPF). The shortest path from one node to the other is calculated in advance without considering load balancing over multiple paths, and all the corresponding traffic are directed through this shortest path.

For cloud-oriented DCNs, several properties of cloud applications make the load balancing highly complex than the traditional (Singh et al., 2008). Prefix-routing is insufficient since workload patterns in cloud-oriented DCN are priori unknown and variable to the network designer. Enterprises prefer to run their applications on commodity hardwares, so the network can meet Quality of Service (QoS) without requiring software or protocol changes. Cloud computing providers use virtualization technology to efficiently multiplex customer's applications and processes across physical machines. It is difficult for customers to deal with inter-VM communication in traditional application manners.

2.4.7 Compatibility

In the actual deployment and upgrading of cloud-oriented DCN, purchasing devices with different capacity in different batch time are often being considered for cost savings and habits. Therefore, how to interconnect large-scale heterogeneous devices whereas ensuring the new DCN and existing networks cooperate efficiently is a major issue to be addressed.

2.4.8 Research and Improvement of DCN Protocol

The management of architecture of DCN are significantly different from the existing Internet architecture. DCN's management is often accomplished by an instance. Thus, its global topology, data flow, failure and various log information can be obtained to assist in protocol design and network architecture design. To propose novel protocols which is suitable for a specific DCN architecture can improve the efficiency of execution.

2.4.9 Automatic IP Address Assignment

Information of location and network topology in PortLand and BCube is stored at the server or switch which improves the performance of routing. Therefore, traditional protocols such as Dynamic Host Configuration Protocol (DHCP) (Droms, 1997) cannot be deployed in this condition. In addition, An automatic IP address assignment mechanism is required to reduce labor costs and the risk of configuration errors, due to the manually configuring such a large number switches or servers is a time consuming and tedious task. Therefore, proposing low-cost, high reliability and manageability automatic address configuration methods regardless of known or unknown DCN architecture is a challenging research perspective.

2.4.10 Future Applications of Optical Switching and Wireless Transmission

The hybrid structure of optical/electrical is superior to traditional electrical switching architectures in term of cabling layout as well as the design complexity and energy consumption. However, optical equipment is still relatively expensive and is not yet deployed in DCN. Therefore, in addition to the architecture design, reducing the cost is also one of the important research perspective. Even though the architecture of fully wireless layout has minimum complexity, however designing reliable and high-performance multi-hop network architecture is still a great challenge. In hybrid architecture of wireless/wired, wireless technology can effectively alleviate the loading of hotspot, and efficient wireless routing and demand traffic aware are the challenging research perspectives.

2.5 Conclusion

This chapter discusses the concept of cloud computing, presents a review on the recent research findings and technologies of cloud computing and cloud-oriented DCN architectures. Motivated by a better support for data intensive applications, how to optimize the interconnection of CDC becomes a fundamental issue. It reviews the existing proposed

architectures by using thematic taxonomy and gives a brief comparison from different aspects, including network scale, bandwidth, scalability, overhead, and cost of deployment. The open issues and challenges of the domain are also highlighted.

We believe that, as a backbone of data center, the architecture of DCN significantly affects its total performance. Our investigation results show that the tree-based architecture lacks of scalability, network throughput, resource fragmentation, and cost for deployment with the increasing demands of tenants in accessing the services of CDCs. Furthermore, we also found that although using additional higher performance and advanced network equipment can effectively improve the whole network performance in a short time period, this approach will cause lower price-performance ratio and take additional cost to service providers and finally shift the burden onto the consumers. However, we only gave a simple comparison among the tree-based to the selected architectures from the above perspectives in this chapter. We did not know that how the tree-based architecture lacks in those areas yet and what the performance is in CDCs. In next chapter, we will discuss the properties and network performance of the tree-based architecture in details by using experiments.

CHAPTER 3

PERFORMANCE ANALYSIS OF THE TREE-BASED NETWORK ARCHITECTURE IN CLOUD-ORIENTED DATA CENTER

This chapter presents the performance analysis of current widely used tree-based architecture in DCN from the perspectives of bandwidth and throughput restriction, network scalability and reliability, resource fragmentation, as well as cost for establishing. The aim of this chapter is to investigate the behavior and performance of the tree-based architecture under different workloads, benchmark the performance of the above mentioned issues, and point out the problems of this architecture deploy in cloud-oriented DCN for further verification. The network scalability and deploying cost are analyzed by using mathematical calculations, and that of network throughput is implemented by benchmarking experiment.

The chapter is organized into three sections. Section 3.1 discusses the analytical analysis of the tree-based architecture in DCN. The benchmarking experiment is presented in section 3.2 and a conclusion is given in the last section.

3.1 Analysis of Traditional Tree-Based Architecture

We conducted an analysis to determine the current best practice and weakness for tree-based architecture in DCNs.

3.1.1 Topology

The tree-based architecture is initially designed for small or medium scale networks. In the era of cloud computing, however, cloud-oriented data center is different from that of traditional enterprise-class because the new requirements (such as higher rate of server utilization and throughput in aggregation, and better price-performance ratio) are desired

for large-scale distributed computing since the number of data center network device is growing rapidly.

As we briefly introduced in Section 2.2.2.1, the tree-based architectures simulates a hierarchical tree structure with a set of linked nodes (server and switch). This structure is divided into three layers: Core layer, Aggregation layer, and Access (Edge) layer, whereas the core layer in the root of the tree, the aggregation layer in the middle and the access layer at the leaves of the tree. For increasing network scale, saving bandwidth, and improving data forwarding rate in CDC, normally the deployment of any security or optimize equipment in the aggregation layer is "omitted". However, a two layers network only support between 5k to 8k hosts due to the limited port number in switch. This scale is available for private cloud computing, but since we target to public (large-scale) cloud-oriented DCN, where the number of host commonly reach to ten thousand and more, we restrict our more attention to the three layer tree architecture in this thesis.

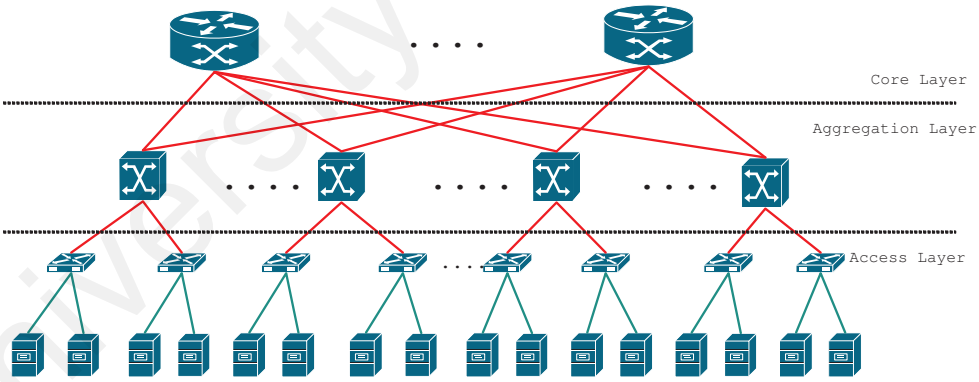

Figure 3.1: Three-layer Tree-based Structure

As shown in Figure 3.1, a rack server (20-80 servers normally) is connected to a Top-of-Rack switch (ToR), and establish an Ethernet by connecting with some End-of-row switches to create the infrastructure of access layer. The switches in access layer connect with that in aggregation layer for bandwidth binding and load balancing. At last, those aggregation switches connect with core switch or router in core layer to provide route for client accessing from the Internet. Typically, the switches at the access layer

have some number of GigE ports (48-256) downlinks and several 10GigE uplinks to one or more aggregation layer switches. Switches with 32-128 10GigE up/downlinks ports commonly deployed in aggregation layer to aggregate traffic between the lower layer of the topology.

The tree-based structure is simple and can be easily deployed and extended by increasing the rack and related switches. However, there are still many problems in this structure that are stated as follows.

3.1.2 Bandwidth and Throughput Restriction

The bandwidth requirement is more important at core layer comparing to the lower layers as the oversubscription by a significant factor (Popa et al., 2010). The definition of the term oversubscription is the ratio of the existing achievable aggregate bandwidth among the end hosts to the total bisection bandwidth of a topology (Infrastructure, 2007). For example in a 1Gbps for Ethernet bandwidth topology, a 1:1 oversubscription indicates that all hosts be able to communicate with others at the full bandwidth of their network. An oversubscription value of 1:4 means that only 25%(250 Mbps) of bandwidth is available for communications. In the tree-based architecture, the bandwidth is convergented from lower to upper layers. That is, two hosts connect to a same physical switch may communicate at full bandwidth, but once data transmission through multiple levels in the tree, the available bandwidth and throughput may restricted. This is due to the bandwidth in core layer is much less than the sum of that in aggregation and accessing layers. The links between aggregation and core layers are normally oversubscription by factors of 1:5 or more due to the equipment cost concerns, which limits the communication among the servers in different branches of the tree, and leads to congestion even the network capacity is still available elsewhere (Greenberg et al., 2009).

Advanced switches are also required besides higher bandwidth and data forwarding

performance in the core layer. The core switch or router is still unable to meet the bandwidth requirements from lower links when the throughput is higher. In this case, though some idle servers exist in the access layer, they may still be assigned any task due to the insufficient bandwidth. In addition, the MapReduce (Dean & Ghemawat, 2008) application, for example, a Reduce worker needs to copy intermediate files from a range of servers in its Reduce operation phase. Moreover, Virtual Machine (VM) migrating and deployment of other bandwidth-intensive applications increase the data traffic in DCN (almost reach to 80% of whole traffic), thus requesting for high network bandwidth and throughput from DCN.

3.1.3 Network Scalability and Reliability

The treebased architecture does not scale well for two reasons. First, in tree-based hierarchical architecture (Infrastructure, 2007), the maximum number of supported hosts is limited by available port density and the requirement of fast failure recovery mechanism. Therefore, it is difficult for the architectures to support the scale of large number of servers in the data center for cloud computing. Second, typical single path routing along trees means overall hosts communication is restricted by some "key" switches, an aggregation or core switch failure may tear down thousands of servers and impact the whole network performance reduced.

Figure 3.2 indicates the maximum possible number of servers that can be supported by different port-number switches. We employ 10 GigE switch with 28, 32, 64 and 128 ports respectively, in the core and aggregation layers , 1 GigE switch with 48 ports in access layer and assume a 1:1 oversubscription ratio in this calculation. As we can see from the figure, one instance of the tree-based architecture employs 128-port switches capable of providing full bandwidth to up 20,480 servers. Even deploying the most advanced switch with 256 ports in aggregation layer become available to increase the scale of net-

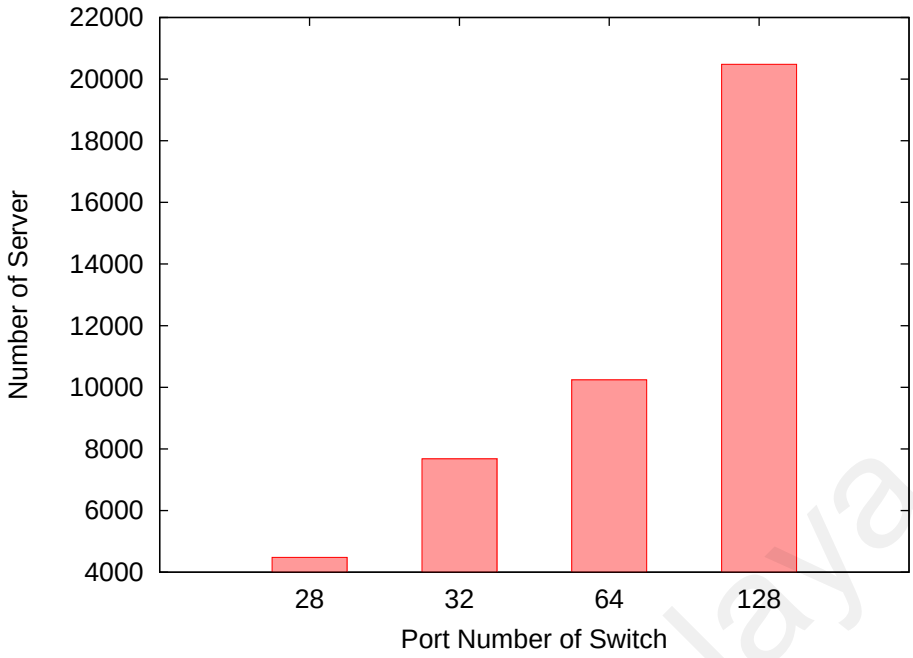

Figure 3.2: The maximum possible number of servers with 1:1 oversubscription ratio

work, it will definitely take a significant extra cost and impact the price-performance ratio to service providers.

3.1.4 Resource Fragmentation

Restricted bandwidth limits the performance of data center for the reason that idle resources cannot be effectively assigned to the place they are needed. The plentiful spare resources capacity is often reserved by individual services or specific devices without sharing for quick responding to nearby servers once network failure or demand request happens. Moreover, the existing network scale in tree-based hierarchical DCN is IP addresses assigning and dividing servers by VLANs. Such IP address fragmentation limits the utility of virtual machines (VMs) migration among servers (IP address has to be reconfigured with VM) and it is inconvenient to reconfigure the IP address for all devices for rapidly growing network. Hence, normally a number of standby servers and IP resource have to be reserved for ensuring the well running of DCN in case of node or link failure happens. In addition, redundancy is a common approach to improve the reliability and availability of a system. The scheme of 1:1 equipment redundancy on the switches

in upper layers is widely used in current tree-based structure. However, to ensure the performance of DCN when switch failure happens, a range of bandwidth is reserved for the redundancy switches and routers.

3.1.5 Cost

The cost for building a large-scale network greatly affects topology design decisions. As we introduced above, the oversubscription is typically deployed for decreasing the total cost of network establishment. Once the oversubscription ratio changing happens in the aggregation and core layers, the only way to enhance the network performance is upgrading high capacity devices (Popa et al., 2010). However, for the reason of large price difference between high-end devices and commonly employed switches and routers, the upgrading cost can be very high. Here we give a cost comparison for different number of hosts and oversubscription by using current practices. We assume that there are two types of switches in the tree-based architecture. The first, used at the access layer, is a 48 ports GigE downlink and 4 ports 10 GigE uplinks switch. In higher layer, we select 128 ports 10 GigE switches. Both types of switches can directly connect to neighbor nodes with a 1:1 oversubscription ratio which is fully bandwidth of their network interface. In this comparison, we select Cisco WS-C3560G-48TS with 48 ports as access layer switch which costs $1,500, and H3C S12518 switch with 128 port 10 GigE switches in the aggregation and core layers. The prices of the two types of switch are collected from Ebay.com. The cost of cabling is not considered in the calculations. Figure 3.3 indicates the cost in millions of US dollars to the total number of hosts in different oversubscription ratio.

As we can see from the figure, each line presents a target oversubscription ratio. The reason of 2.5:1 (400 Mbps) and 8:1 (125 Mbps) factors are selected is that typical oversubscription is designed from 2.5:1 to 8:1 at present (Infrastructure, 2007). The fig-

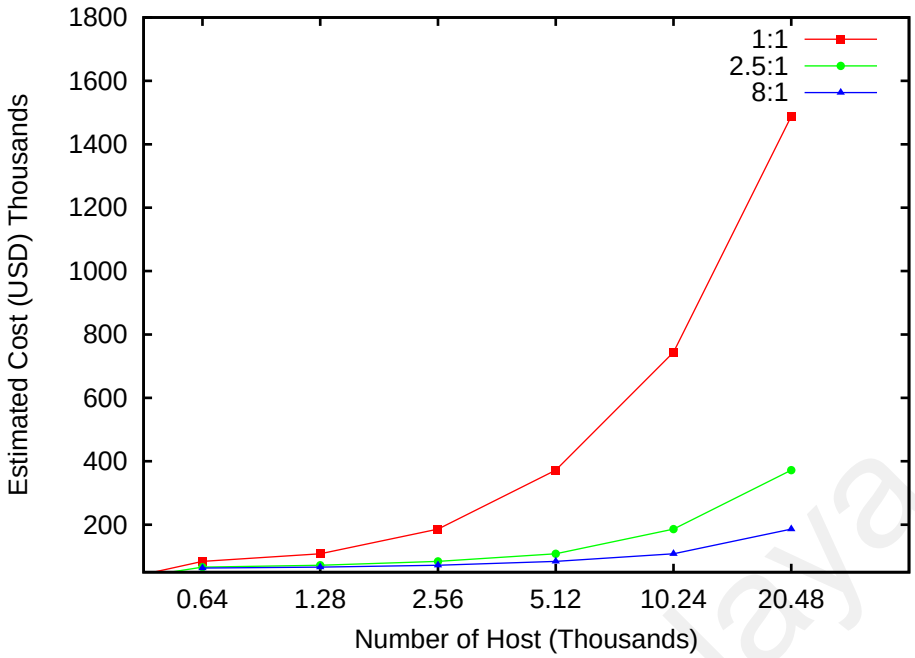

Figure 3.3: The estimated cost vs. maximum possible number of hosts with different oversubscription ratio

ure shows that with the increasing number of host in tree architecture, the expense of deploying the high-end network devices is increased with significant rate under different oversubscription ratio. For instance, supporting 20,480 and 640 hosts with full bandwidth (1:1) among all hosts, the switching equipment costs to $1.4M and $84,000, respectively. To support the bandwidth of 400 Mbps and 125 Mbps among 20480 hosts, the costs for switches are $372,000 and $186,000, respectively.

3.2 Benchmarking Experiments

One simple experiments implemented in this section allows to benchmark the behavior and throughput performance of the tree-based architectures under different workloads and network conditions. The experiment is implemented by using real physical devices.

3.2.1 Throughput Analysis

The throughput measures the maximal sustainable rate at which a node can expect data to transfer. Given a bandwidth of link, it is shared by multiple data flows transfer over this link. The maximal throughput between any pair of servers is restricted to the bandwidth

and flows of the corresponding route. The formula of throughput calculating is:

$$\tau = \frac{\sum_{i=1}^{n}(P_i) \times \delta}{\sum_{i=1}^{n} D_i} \tag{3.1}$$

(Bianchi, 1998)

where τ is the throughput, P_i denotes the i^{th} received packet, δ is the size of the packet in bits, and D_i is the delay of packet i.

We assume that there are total number of i packets transfer over a path between source node s to destination node d. The size of receiving data by d is $\delta \times i$, and the total data packet delay is $\sum_{i=1}^{n} D_i$. Thus, we can calculate the average throughput between two server nodes according to the above equation. The aggregated throughput is the sum of the average throughput, thus the sum of all data rates delivered among all terminals in a network.

3.2.2 Implementation

As our research is focusing on cloud-oriented data center network, the MapReduce model is selected in the implementation to measure the performance of different DCN architectures. MapReduce (Dean & Ghemawat, 2008) is one of the most widely used data processing mechanisms in cloud computing. It provides a programming and execution model for processing and generating large data sets. The main purpose of using MapReduce in the implementation is trying to close to the real situation. Because this model is successful deployed in cloud computing by Google, Yahoo, Amazon and some other Internet service providers, if our proposed STB model get a better throughput performance in a small cloud DCN environment, this model may also improve throughput for large DCN in the future.

3.2.2.1 Test-bed

For the implementation of our research, a test bed is built. It consists of twelve Dell Optiplex GX520 desktops and one Lenovo Ideapad Z470 laptop as servers, whereas the one desktop server acts as master and the rest eleven desktops act as workers, the laptop connects to core switch to monitor the network performance. Each work server has an Intel 3.2GHz dual-core CPU, 4 GB DRAM, 80 GB hard disk, and Linux Ubuntu 11.10 32-bit operating system. The laptop has a 2.3GHz CPU, 4GB RAM, 640GB hard disk with Ubuntu 11.10-32 bit as well. These servers are interconnected by six Cisco WRT54GL 4-port switches and one Cisco Catalyst 2950 24-port switch to create the structure shown in Fig 3.4, whereas the 4-port switches deployed in access and aggregation layers and the 24-port switch deployed in core layer. The bandwidth of server to switch is setting to 100 Mbps and that of switch to switch is setting to 1Gbps. Java with JDK version 1.7.0, Hadoop MapReduce 1.2.1, VMware Workstation 9.0.2, Python 2.7, and Ntop 3.3.10 are pre-installed in the physical servers. At the initial stage of the implementation, we use the 11-worker servers to complete the MapReduce task. In order to act a real DCN environment, in the following 3 stages, we deploy 2, 4 and 8 virtual machines (VMs) in each of server, respectively. Therefore, the total number of workers in the architecture increases from 12 to 24, 48 and 96, respectively, which is enough to act a real small DCN.

To show that the throughput performance of the tree-based network, we employ the word count (Condie et al., 2010) as an example. Word count is a kind of MapReduce applications used for enumerating the number of specific words repeated in a block of text. In this implementation, we randomly generate 100 million ip addresses from the range of “192.168.0.0” to “192.168.255.255” by python, to enumerate the top ten ip addresses repeated most often in the ip pool. The details of MapReduce deployment and data collection are presented in chapter 5.

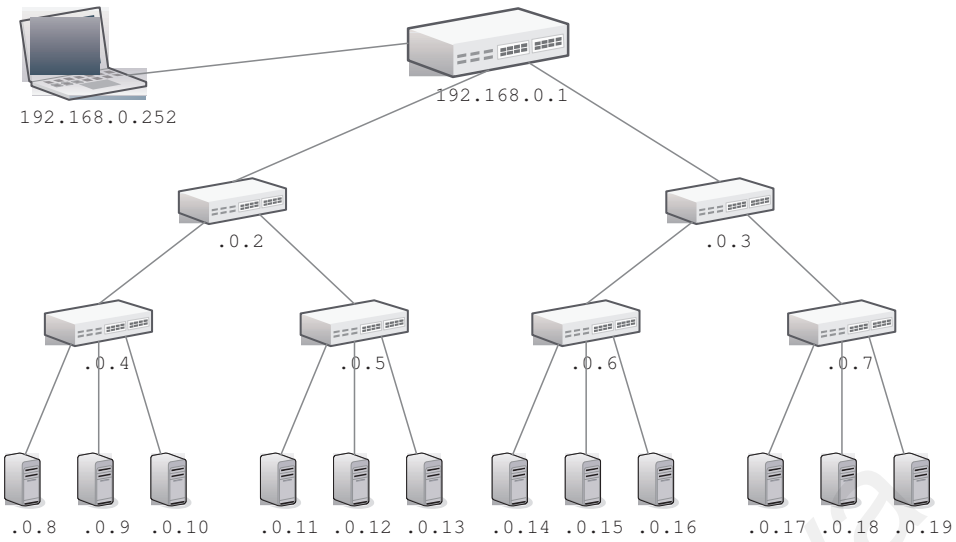

Figure 3.4: Tree-based network topology

3.2.2.2 Results

In this implementation, master server assign a 1.5GB text file to worker servers. Normally, the text file is splited to several blocks with 3 duplicates, and stored in different mapping worker servers. Figure 3.5 illustrates the data forwarding throughput result of the tree-based architecture, where the total number of servers varies from 12 to 96. We can find that the throughput of architecture is slightly degraded when the number of servers is increased. The average network throughput for 12 to 96 nodes was observed in a range from 44.87Mbps to 8.38Mbps, respectively. With the number of server has increased, the amount of network bandwidth available for per server is decreased, more and more data delivery flows share 1 link. For ensuring the quality of service (QoS) and avoiding data losing, the TCP transmission rate needs to be reduced. Therefore, more high-end network devices are needed in order to acquire higher throughput performance.

Figure 3.6 plots the aggregated throughput of the tree-based architecture with different number of servers. The throughput with 96 servers has the best performance which completed at 144 seconds, and the transmission with 48, 24 and 12 servers completed their tasks at 434, 616, and 934 seconds, respectively. We observed that, even the server number of 96 are about 2 times than 48, but the aggregated throughput is less than 2 times

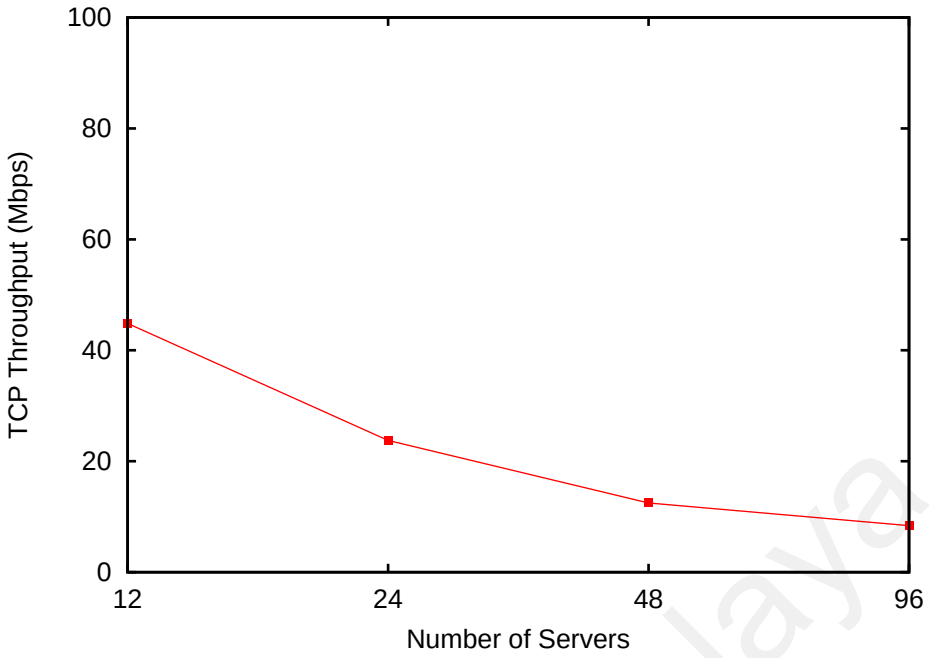

Figure 3.5: TCP throughput with different number of servers

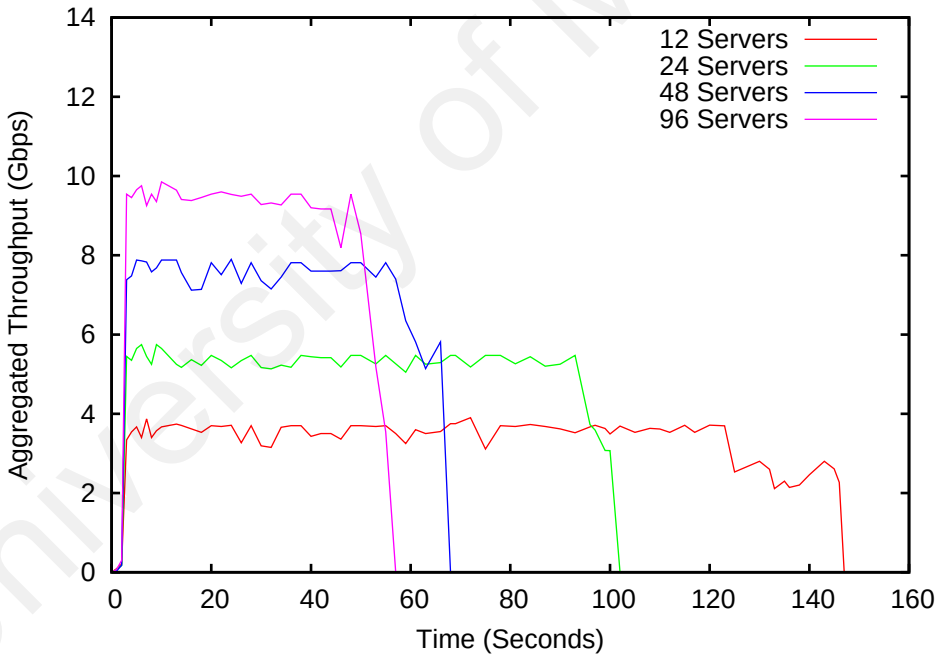

Figure 3.6: Aggregated throughput with different number of servers

(around 9.544 : 7.813 Gbps) due to the bandwidth and throughput restriction, which affects the speed of data processing in the tree-based network. Therefore, simply increasing the number of servers and network equipment cannot increase the throughput performance as much as we desired in the tree-based DCN.

3.3 Conclusion

This chapter analytically and experimentally analyzed the problems in current widely used tree-based architecture in cloud-oriented DCN. We analyzed the problem of the architecture from the perspectives of bandwidth and throughput, network scalability and reliability, resource fragmentation, as well as cost of deployment. Using series of calculation we verify the limitations of the network architecture that only 20,000 around servers can be supported even using the most advanced network devices, and costs $1.4M for deploying such network devices in the tree-based architecture. We also using a benchmarking experiment in real physical environment to validate our findings in throughput performance. We benchmark the results of TCP and aggregated throughput by using MapReduce model to analyze the impact with different number of servers in the tree-based architecture.

Our benchmarking results advocates that network throughput performance is directly impacted by the number of servers in tree-based architecture, thus more number of servers, lower TCP throughput. Moreover, with the restriction of bandwidth and the existing switching technology, simply increasing the number of servers cannot improve the aggregated throughput as much as we thought, but also decrease the price-performance ratio. Therefore, with the coming era of cloud computing and the scale of DCN is constantly increasing, we desperately look for a novel DCN architecture with the features of scalability, low cost, higher throughput and fault-tolerance routing without redundancy devices, to replace the traditional, small/medium scale oriented tree-based architecture. In next chapter, a novel architecture called Sierpinski Triangle Based (STB) architecture will be introduced.

CHAPTER 4

SIERPINSKI TRIANGLE BASED DATA CENTER NETWORK ARCHITECTURE

This chapter presents the proposed network architecture in this research. Starting from the sierpinski triangle, we present the physical structure and the construction methodology of the proposed architecture. Moreover, the routing mechanism of the proposed architecture, which provides a fault-tolerant approaches to nodes is introduced in detail. The topological properties of the architecture is also presented from the perspective of network size, network diameter, and bisection width.

The remainder of this chapter is as follows. Section 4.1 covers the classic deterministic fractal, Sierpinski triangle (or Sierpinski gasket). Section 4.2 presents the physical structure of Sierpinski Triangle Based (STB) and its building procedure. Section 4.3 describes the routing and node identification mechanism for STB. Section 4.4 points out the distinctive topological properties of STB. The chapter is concluded in section 4.5.

4.1 Sierpinski Triangle

The well-known Sierpinski triangle (Sierpinski, 1916), shown in Figure 4.1, is constructed using iterative approach. We assume that the S_t is the Sierpinski triangle after t-time iteration, and the construction is as follows: starting with an equilateral triangle and S_0 is the initial configuration with $t = 0$. In the first generation S_1, we select the midpoint of the three sides of the equilateral triangle and link them to construct four small triangles, and then remove the central triangle. In the second generation S_2, linking three midpoints of sides in each small triangles in the same way as in first generation. And more generally, we repeat the procedure for all new small triangles and get the classic deterministic fractal, sierpinski triangle S_t, when $t = \infty$. One of major properties of the sierpinski triangle

is unlimited number of triangles (S_t) in a limited area (S_0), which can be used for network infrastructure deployment.

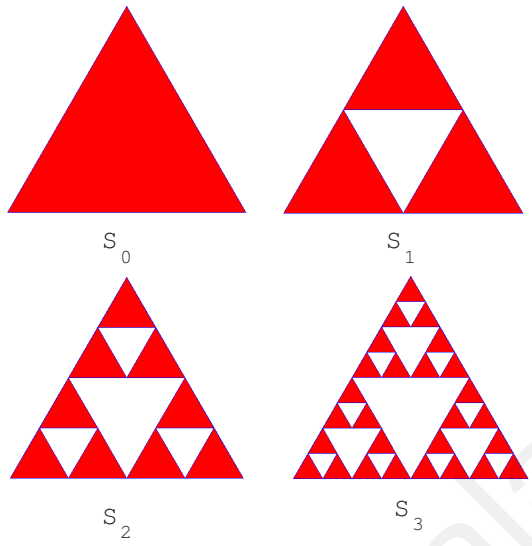

Figure 4.1: Structure of Sierpinski Triangle

4.2 Sierpinski Architecture

One of the major research objectives is to increase the network scalability by deploying more devices in DCN, whereas the property of unlimited number of triangles in sierpinski triangle can be used to fulfill the architecture design. The physical structure of Sierpinski Triangle Based (STB) architecture is introduced in terms of the initial recursive unit and the recursive rule.

4.2.1 Physical Structure

STB architecture is recursively defined in order to be scalable and to meet the design goals of a DCN. The initial recursive unit of STB is represented first and followed by the rule of higher-level STB construction.

4.2.1.1 Initial Recursive Unit

The initial recursive unit, denoted as $STB_0(K)$, is the basic building block of the whole network topology, where K denotes the number of servers in a STB_0. Each server with

multiple network adapters connects to one switch but none of servers connect to each other directly. According to the diagram S_0 in figure 4.1, we assume that each line is a server and connecting to a switch located in the central of the triangle to establish a structure of $STB_0(3)$. The vertical view of $STB_0(3)$ is shown in figure 4.2(a), where $K = 3$ servers are defined by connecting $n - port$ mini-switch. Then each line of the equilateral triangle is selected as the position of servers in S_0, and switch is added in the central of the triangle, figure 4.2(b) shows the side view of the architecture.

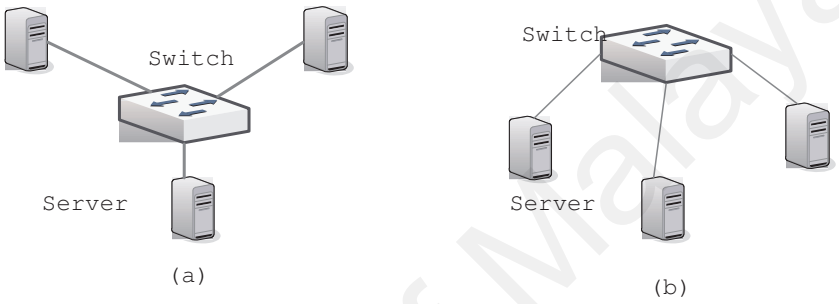

Figure 4.2: Example of $STB_0(3)$

The switch also can be connected by 4, 5, 6 or more servers to establish $STB_0(4)$, $STB_0(5)$, $STB_0(6)$ and $STB_0(K)$, respectively, which are shown in figure 4.3.

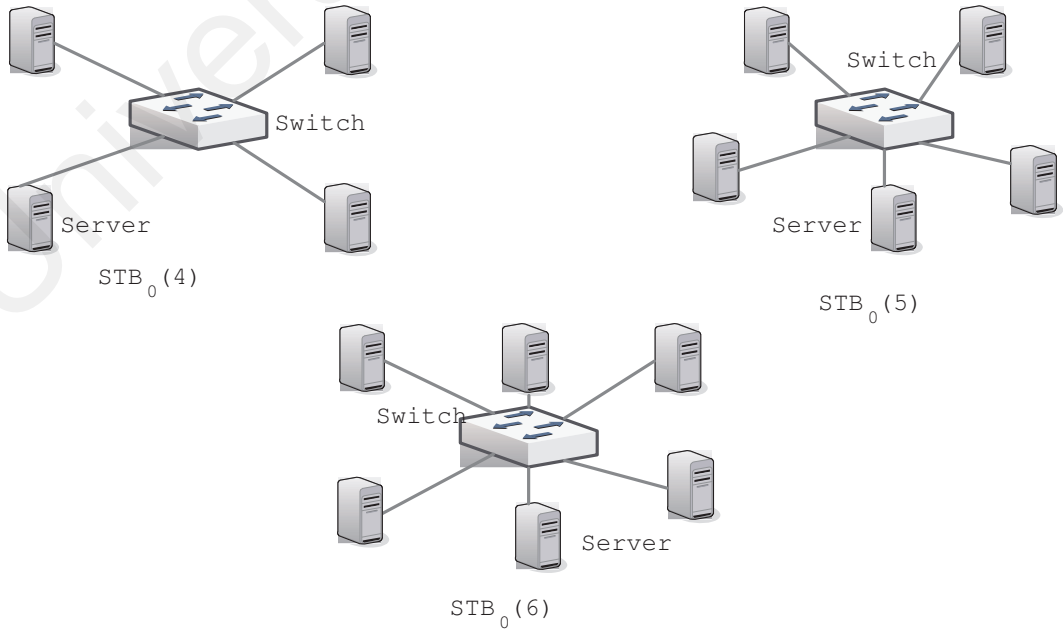

Figure 4.3: Example Topology of $STB_0(4)$, $STB_0(5)$ and $STB_0(6)$

4.2.1.2 Recursive Rule

The recursive rule determines the interconnection mode of the recursive units. Let $STB_n(K)$ denotes a level n recursive unit $(n \geqslant 1)$, which employs $STB_0(K)$ as the initial recursive unit. A number of low-level units make up a high-level structure through interconnecting the nodes in different level of structures. In this research, an $STB_n(K)$ is constructed by $K^n + STB_{n-1}$ units, which are interconnected through the K^{n+1} servers at level n. To generate $STB_n(K)$ structure, a number of M initial recursive units are added between switches in level n to servers in level $n-1$, and the rest $K-M$ units are added between each pair of servers in level n units. The pseudocode for constructing a STB network is shown in Algorithm 1. An example of $STB_n(K)$ construction method will be introduced in next section.

Algorithm 1 Pseudocode of STB Construction

/* construct Sierpinski Triangle Based architecture $STB_n(K)$, K is the number of servers in STB_0, and n denotes the level of the structure. */
if$(n == 0)$
$for(inti = 0$;i<k; $j++)$
connect the i^{th} server to switch according to the position in Serpinski triangle;
establish virtual links between server;
end
else /* $(n \neq 0)$, construct $STB_{1\to\infty}$ */
if$(n = 1)$
{
add Unit between servers;
connect mini-switch of new Unit to its neighbor servers in $n = 0$ level;
}
else while $(n > 1)$
{
add Units between the two vertexes of servers in $n-1$ level;
add Units between the mini-switch of $(n-1)$ level and server of $(n-2)$ level;
}

4.2.2 Construction Method

To generate a high-level STB, we need to construct the initial recursive unit and then expand the unit to any higher-level STB recursively. We propose the construction method for an $STB_0(3)$ and an $STB_n(3)$ as examples and introduce as follows.

Based on the physical structure of an $STB_0(3)$ in figure 4.2(a), we add new initial units between each pair of servers to create the STB_1 structure. As shown in Figure 4.4, the switches of the new initial units connects to their neighbor STB_0 servers.

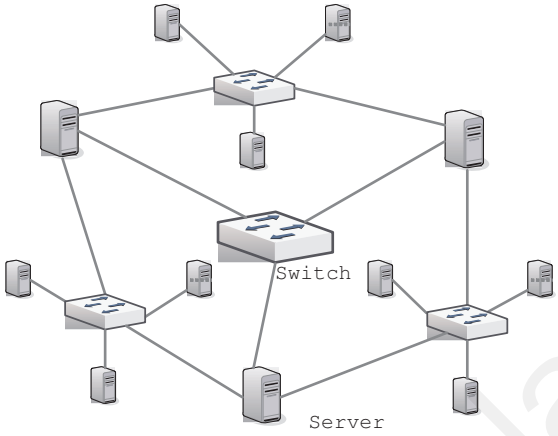

Figure 4.4: Example Topology of $STB_1(3)$

Then in the subsequent generations, a new initial recursive unit is added between one pair of servers in S_{n-1}, and two more units are added between the switch of $(n-1)-level$ and server of $(n-2)-level$. Figure 4.5 shows a part of $STB_2(3)$ architecture, and the new adding units are circled.

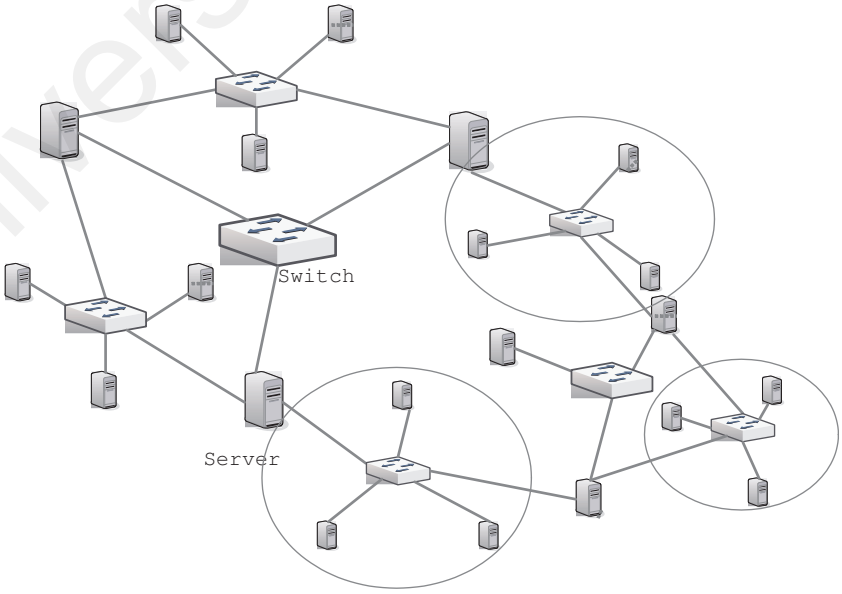

Figure 4.5: Part Example Topology of $STB_2(3)$

The above STB architecture is defined when $K = 3$ (3 servers connect to 1 switch in

the initial recursive unit). Similarly, according to the features of Sierpinski triangle, the example topology of $STB_1(4)$ is shown in Figure 4.6.

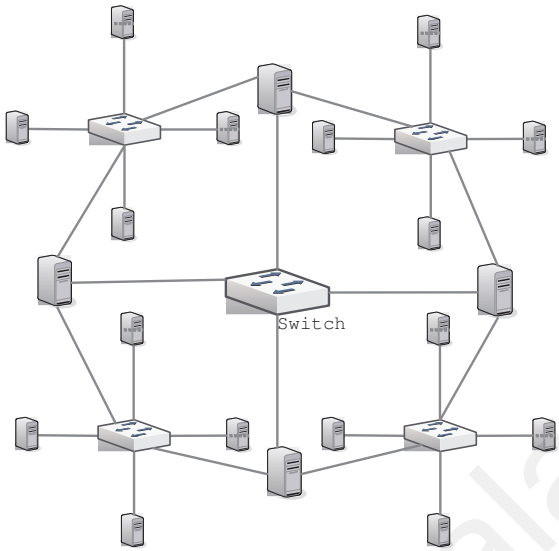

Figure 4.6: Example Topology of $STB_1(4)$

The advantages of this mechanism are that (a) the network structure is designed by adding fixed module to achieve network expansion, (b) we are able to establish the $(n+1)$ level though the n level structure is not fully constructed. In such situation, the structure of lower levels can be modified without changing any links or nodes in $(n+1)$ level.

4.3 Node Identification and Routing schemes in STB Architecture

The number of nodes in STB is increasing rapidly as exponential (K^n)growth. Traditional routing schemes, such as global link-state routing (LSR) (Clausen et al., 2003) and open shortest-path first (OSPF) (Pioro et al., 2002) routing are unsuitable using in STB since the STB's goal is to interconnect up to millions of servers. In this section, we first introduce the node identification scheme in STB and then represent the STB routing to provide fault-tolerant scheme for DCN.

4.3.1 Node Identification Scheme

In STB, we use a new node identification scheme to replace the traditional IP address scheme in DCN. We assign each server a 2-tuple $[L, D]$, whereas L is the level IDs and D is the degree of node in the architecture. For switch assignment, we add a tuple "w" at the end of the array as $[L, D, w]$.

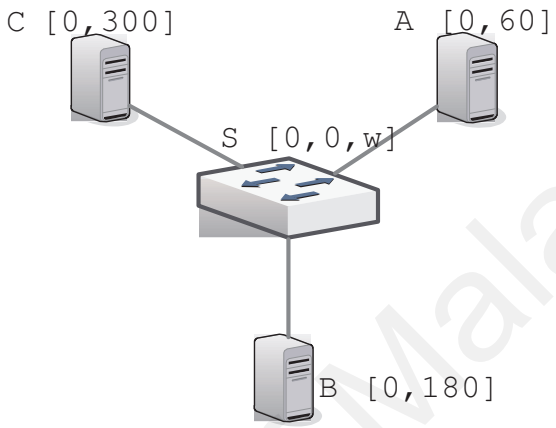

Figure 4.7: Example of Node Identify

As shown in Figure 4.7, the servers in $STB_0(3)$ are assigned the address as $A[0, 60]$, $B[0, 180]$, and $C[0, 300]$, and the switch is assigned as $S[0, 0, w]$. The first value 0 indicates that those servers locate at 0*level* of STB(3), and the second value shows the degree for each node in the architecture. Assuming node S is the center of clock, such above degree value is a relative value from 0° to 359° compare with the position of switch in clockwise rotation starting from 12 o'clock. As S is 0° in this architecture, we have server A is 60°, B is 180°, and C is 300°.

Node identification in $STB_1(3)$ is shown as follows: D is a 1*level* switch and locates between A and C, so its address is $D[1, 0, w]$. Then the three servers in $Unit_D$ trisect the range of degree between server A and C, which is from $300^\circ - 60^\circ$. Hence we get the addresses for those three servers are $E[1, 30]$, $F[1, 0]$, and $G[1, 330]$. The rest addresses of nodes are assigned in the same way, which is shown in Figure 4.8.

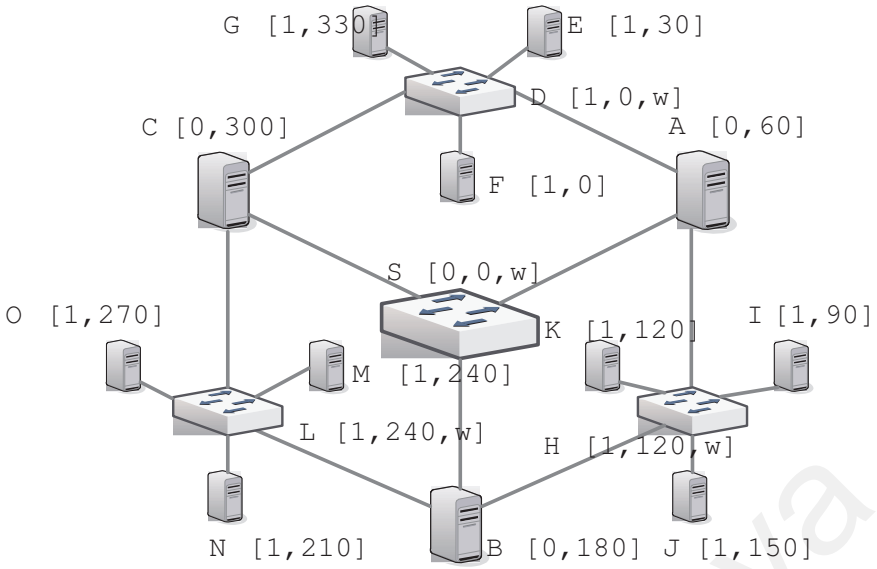

Figure 4.8: Node Identify in S_1

4.3.2 Routing scheme

The STB rouing serves as a network layer for STB-based DCN. It includes packet header format and protocols.

4.3.2.1 Packet header

We use a 24-bit uid to identify a specific node in STB. As shown in figure 4.7, the level number and degree value use the first 16 bits and the following 2 bits is used to identify the nodes' type. The rest 6-bit is reserved for further development. Figure 4.9 shows the format of the packet header with size of 24 bytes. The design of the STB header borrows from IP packet. We set a 32-bit in *identification* field as same as that of IP packet is because the bandwidth in DCN is very high. Using a long bit field can effectively reduce the identification recycles in a very short period of time (Deering, 1998). Setting 4 in *Retry* field denotes the maximum number of local link error recovery allowed. Once a packet losing happens, the *retry* count becomes 1 in the resending packet. If the retry value reaches to 4 means the original link or destination node is failed. The *Flag* field denotes the type of received data packet. Commonly there are 4 types of packet can be identified by *Flag*, which are ACK, PSH, SYN and FIN. The ACK is acknowledgment

reply that indicates the destination node has already received a packet. PSH flag, short for Push, denotes that the receiving packet contains buffered data. When a node desires to send data to destination node, a packet with *SYN* flag will be sent first for asking communication connection. The *FIN* denotes the data transmission is over, and the connection can be closed.

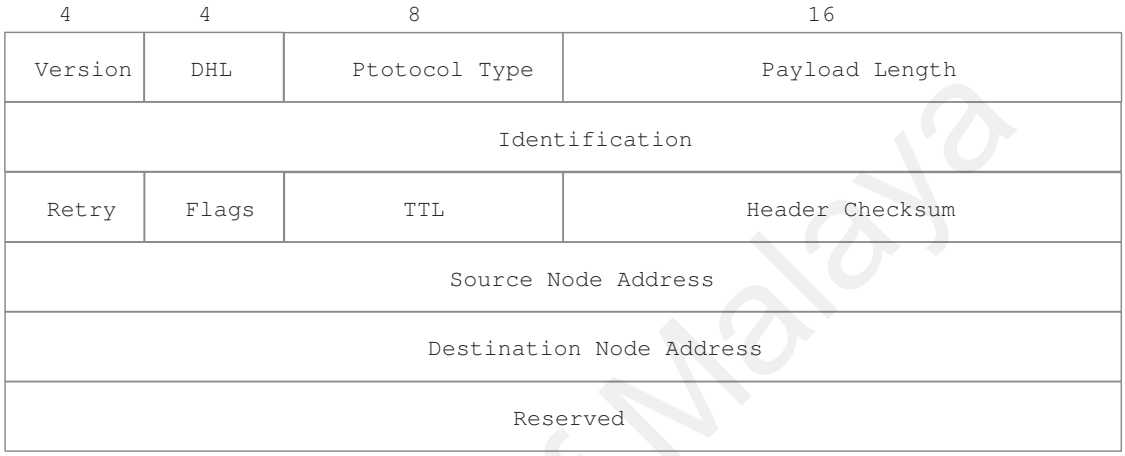

Figure 4.9: The STB packet header

We denote that each data packet in STB has its own ID, source node's ID, and destination node's ID, and intermediate nodes add their IDs into the head of packet in data forwarding. Once an intermediate node find its own ID in a data packet (means this packet has been transferred by itself before), the packet is discarded to avoid looping.

4.3.2.2 Routing without failure

STB uses a simple and efficient single-path routing algorithm for unicast through the recursive STB architecture. The routing algorithm is named as $STBRouting$. In STB, each node maintains a $2hops$ routing table including the address of its neighbors. Assuming that a source node *s* desires to send data to a destination node *d* that are in the same $nlevel$ but different recursive units. When computing the path from *s* to *d* in a STB_n, we first calculate the intermediate link (s,i) and (i,d) that interconnects the two recursive units through lower level. Routing is then divided into how to find the intermediate node *i*

which directly interconnects the two recursive units and locates in the highest level. The final path of *STBRouting* is the combination of the two sub-paths, (s,i) and (i,d).

In routing selection, node *s* first checks its routing table. Once the path of *d* is recorded in the table, means the two nodes are 2*hops* neighbor nodes. Then the switch in the same recursive unit is considered as node *i*. So data packet can be transferred through the path to *d*. If there is no routing information about the *d* in the table, node *s* selects data forwarding direction according to the *degree value* of *d*, and estimate path length depending *level value*. A fundamental rule of routing selection is that once a node has more than one paths for data delivery, choosing the next hop whose *degree value* is closer to that of node *d*. If the *degree* absolute values of the multiple paths are same, selecting the path with smallest *degree value* to delivery.

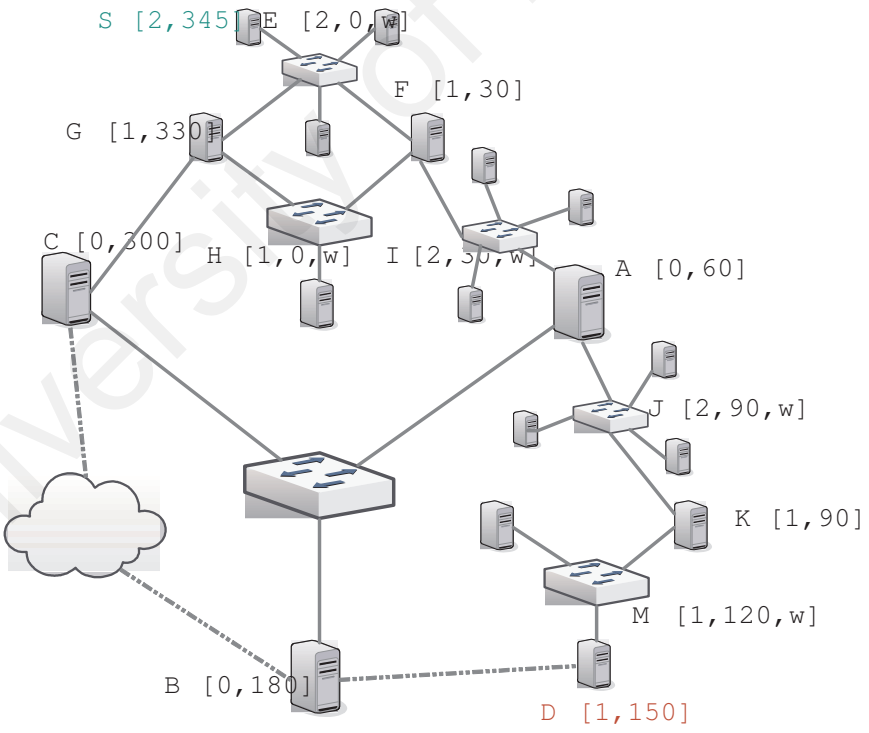

Figure 4.10: Routing Selection in $STB_2(3)$

We take an example to introduce the proposed STBRouting for convenience. Figure 4.10 shows a part of $STB_2(3)$ structure. We assume that source node *S* wants to send data to a destination node *D*. First of all, node *s* checks the routing table, but no

routing information as the length of path is obviously more than two hops. In next step, $S[2,345]$ forwards data to switch $E[2,0,w]$ according to the unique link, and E sends data to $F[1,30]$. In this forwarding procedure, E actually has two options, server F and server G. The *absolute degree value* of F to D is closer to that of G to D which is 120, hence the data packet is forwarded to F. Similarly, node F forwards data through node I is because the *absolute degree value* of I to D is 120, that is smaller than the value (150) of H to D. In this way, the data is forwarded through, $I[2,30,w]$, $A[0,60]$, $J[2,90,w]$, $K[1,90]$, $M[1,120,w]$ and finally arrives to destination node $D[1,150]$.

The pseudo code for path selection procedure without failure of STB is shown in Algorithm 2.

Algorithm 2 Pseudocode for Path Selection without Link Failure

```
/* src is source node, dst is destination node, i is the degree value of node, Path(s,d)
indicates the path from s to d */
STB_rtable :: rt_lookup(addr_dst id) /*look up dst(id) in routing table */
if(rt → rt_dst == id) /* if the dst id is in routing table */
  append addr_dst to path(s,d);
  goto send;
elseif(|i_dst − i| ≤ 180) /* the forwarding path will not through i = 0 node */
  sendto(i_nextHop → (i,i_dst));
else{
  if(i_dst > i){ /* i_dst ∈ (180,360) & i_intermediate ∈ (0,180)*/
    send to(i_nextHop → 0);
    when (i == 0) received;
    send to(i_nextHop → i_dst);
  }
  else{
    send to(i_nextHop → 360);
    when (i == 360/0) received;
    send to(i_nextHop → i_dst);
}
```

4.3.2.3 Fault-tolerant routing

Node and link failure are very common in large-scale DCNs. Because a link disconnection due to the physical problem of network cable few happens, we regard the link failure as a failure of a pair of nodes. In this research, the proposed STBRouting approach fo-

cuses on the node failures to provide a distributed, fault-tolerant routing protocol for STB networks without redundancy equipment.

As introduced above, each data packet sets a threshold in *Retry* field for determining link state. If a node does not respond with its state information within the threshold time, it will be regarded as a failed node. Because *STB* is a recursive structure, more than 1 parallel paths are provided between each pair of two nodes (will be proved in next section). When the original link failed, the data forwarding node will select another path according STBRouting selection scheme to transfer data. We take an example to explain the proposed fault-tolerant routing scheme which is shown in figure 4.11.

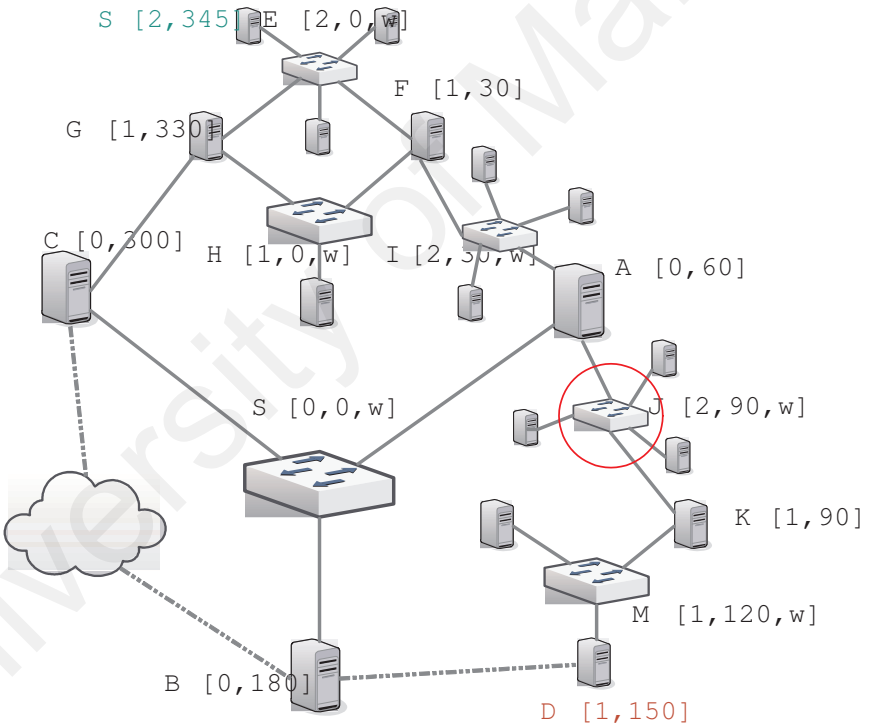

Figure 4.11: Routing Selection in $STB_2(3)$

Node *J* locates the routing path between source node *S* to destination node *D*, and assuming that node *J* failed during data transmission period. Node *A* forwards data to *J* but no response. After 4 retry times, the STBRouting selection scheme is activated in *A*. Based on the *absolute degree value* of neighbor nodes, *A* selects node *S* as its next hop to transfer data. The data packet then is delivered through $S[0,0,w]$, $B[0,180]$ to *D*. It is

worth to mention that, the link $[A,S]$ is the only path after J failed in this case. In a real network, however, the scheme of *absolute degree value* and *level* should be executing restrictively when node has more than 1 path options.

Each switch in an $STB(K)$ DCN connects a number of K servers. A failed switch may affect all the servers that connect to it. Fortunately, in STB, when a switch out of the STB_0 recursive unit failed, at most only K servers connecting to it might be unavailable. If there are some higher level recursive units in the failure switch's unit, then the number of unavailable server might be 1 only. Let us see figure 4.11, when node J goes wrong, the $K = 3$ servers in J's recursive unit is unavailable. But if the recursive unit has higher level units (such as in node H's unit), even though the switch H is failed, server G and F still can be connected through the parallel paths, and only 1 server is unavailable.

The pseudo code for fault-tolerant path selection procedure of STB is shown in Algorithm 3.

Algorithm 3 Pseudocode for Path Selection with Link Failure

/* *src* is source node, *dst* is destination node, i is the degree value of node, $Path(s,d)$ indicates the path from s to d */
for $i = 0$; $i++$ /*record retry times */
return;
if $i < 4$ /*local route recovery is successful*/
  goto send;
else if
i == 4 /*local route recovery is failed, activate STBRouting selection*/
{
$if(|i\_dst - i| \leq 180)$ /* Compare the absolute degree value */
  $sendto(i\_nextHop \rightarrow (i, i\_dst))$;
else{
  $if(i\_dst > i)$\{ /* $i\_dst \in (180,360)$ & $i\_intermediate \in (0,180)$*/
    $send\ to(i\_nextHop \rightarrow 0)$;
    $when\ (i == 0)\ received$;
    $send\ to(i\_nextHop \rightarrow i\_dst)$;
  }
  else{
    $send\ to(i\_nextHop \rightarrow 360)$;
    $when\ (i == 360/0)\ received$;
    $send\ to(i\_nextHop \rightarrow i\_dst)$;
}
}

4.4 Topological Properties of STB Architecture

In this section, we analyze the topological properties of STB, including network size, bisection width, and network diameter.

4.4.1 Network Size

The size of a DCN depends on the amount of servers it is able to accommodate. We have discussed in chapter 3 that the network size is an important consideration in cloud-oriented DCN architecture design. In STB architecture, we assume that there are k servers in a $0level$ STB architecture $STB_0(K)$, and the number of servers and mini-switches in $nlevel$ of STB is denoted as SV_n and SW_n, respectively. Taking the value $n \geq 0$, we have:

THEOREM 1

$$SV_n = k \times SW_n \tag{4.1}$$

and

$$SW_n = 1 + \sum_{i=1}^{n} k^i \tag{4.2}$$

and

$$SW_n = SV_{n-1} + 1 \tag{4.3}$$

PROOF

(1) As STB_0 has k servers, thus the proportion of servers to switches is: $SV_n : SW_n = k : 1$, similarly, the proportion in the recursive unit is also true. As the STB architecture is constructed by adding multiple $Units$ based on previous level, so it is obvious that the proportion $SV_n : SW_n = k : 1$.

(2) As the STB is a recursive architecture, When $n = 0$, we can easily get $SW_0 =$

$1 = k^0$; similarly, when $n = 1$, $SW_1 = k + SW_0 = k^0 + k^1$, and $SW_2 = k + SW_0 + SW_1 = k^0 + k^1 + k^2$. Hence we can get $SW_n = k^0 + k^1 + k^2 + ... + k^{n-1} + k^n$, $n\varepsilon(0,\infty)$, that is $SW_n = 1 + \sum_{i=1}^{n} k^i$.

(3) From the equation (1) and (2), we get $SV_{n-1} = k \times SW_{n-1} = k(1 + \sum_{i=1}^{n-1} k^i) = k + k \times \sum_{i=1}^{n-1} k^i = k + \sum_{i=2}^{n} k^i = 1 + \sum_{i=1}^{n} k^i$.

Theorem 1 shows that the number of servers and server proportion in a STB as the level increases. A small level can lead to a large network size. Taking the value of n=6, we calculate from equation (4.1), (4.2) and (4.3) that the number of servers in the $STB_6(3)$, $STB_6(4)$, $STB_6(5)$ and $STB_6(6)$ are 3,279, 21,844, 97,655 and 335,922, respectively.

4.4.2 Bisection Width

The bisection width of a DCN is the minimal number of links can be removed to partition a full DCN into two approximately equal-size sub-networks. A larger bisection width denotes a higher network capacity.

THEOREM 2

The lower bound of bisection width of a STB network is $\frac{SV}{K \times \log_n SV}$, (4.4)

where

$$SV = K \times (1 + \sum_{i=1}^{n} k^i) \quad (4.5)$$

PROOF

According to the structure of STB and *theorem* 1, one $STB_n(K)$ is constructed by $1 + \sum_{i=1}^{n} k^i$ switches. Each of the switches has K links connecting to their own recursive units, so the number of servers is $K \times (1 + \sum_{i=1}^{n} k^i)$. When $n > 1$, one $STB_n(K)$ can be

divided into two equal parts by removing $\frac{K}{2}$ links from each of the switches. Each part consists of $\frac{K}{2} \times STB_{n-1}(K)$. Based on the (Leighton, 1992), the lower bound of the bisection width of an $STB_n(K)$ is $\frac{SV}{K \times \log_n SV}$, where $SV = K \times (1 + \sum_{i=1}^{n} k^i)$, denotes the number of servers in STB.

4.4.3 Network Diameter

Network diameter denotes the maximum number of hops in the shortest path between each pair of nodes. A shorter network diameter indicates a potential higher data exchange rates.

THEOREM 3

The diameter of a STB network is at most (2n+1) (4.6)

PROOF

We assume that a source node S and a destination node D locates at $M-level$ and $N-level$ of STB architecture, S_M and S_N, respectively, where $(M, N \leq n)$. The maximum path for S to a S_0 server is $S_M, S_{M-1}, S_{M-2}, ..., S_{M-n}, ..., S_1, S_0$. Similarly, the maximum path for D to the a random S_0 server is $S_N, S_{N-1}, S_{N-2}, ..., S_{N-n}, ..., S_1, S_0$. As only 1 hop between each S_0 server, so the maximum path length from S to D is $M+N+1$. When $M = N = n$, the maximum path length in STB is $2n+1$.

Theorem 3 indicates that the network diameter in S_n. Network diameter indicates the number of nodes between source and destination nodes. That is also called the maximum path length between pairs of nodes. Normally, network with a longer diameter means slower routing convergence and lower network stability performance as long distance transmission. Therefore, the diameter decreasing can effectively improve the speed and quality of data transmission.

THEOREM 4

Each pair of two servers have 2 *parallel paths at least, and* 2^{2n} *at most* (4.7)

PROOF

As we break the virtual link(s) between servers of $(n-1)-level$ and add *Cell* to construct $n-level$ STB network, hence we have at least 2 parallel paths for each of two servers. According to the *Theorem*2, the maximum path length from source node S to destination node D is $M+N+1$, thus we have 2^M parallel path between S and S_0, and 2^N for D node. Therefore, when both S and D locate in same level $M=N=n$, we have the number of most parallel paths for two nodes is $2^M \cdot 2^N \cdot 1 = 2^{2n}$.

The parallel paths are the links between pairs of servers, which is independent to others. Those pair of two servers can be selected from any position, any levels in STB, and no matter if they connected to the same switch or not.

According to the Theorem 1, it is easy to get the size of network has an exponential growth and it may interconnect up to millions of nodes, some traditional global link-state routing algorithms, such as OSPF, are no longer suitable in STB architecture. In the following section, we use STB-3 as an example to propose the STB routing scheme starting from node identify mechanism.

4.5 Conclusion

This chapter presents a new architecture, call Sierpinski Triangle Based (STB) for cloud-oriented DCN. One of the distinctive features of the proposed STB architecture is that STB is established on the basis of sierpinski triangle fractal, where higher level STBs are built recursively from many lower lever STBs to provide scalable network. Moreover, STB uses an angle-based node identify mechanism is introduced for replacing the IP-

based address allocation for supporting the size of DCN with exponential growth. Due to the existing global link-state routing and OSPF schemes are unsuitable to be used in millions of servers interconnecting networks, we proposed our STBRouting to provide a fault-tolerant routing scheme without redundancy equipment.

The best application scenario for STB is large DCNs. The ultimate goal for DCN research is to support all-to-all traffic patterns without throughput bottleneck in a scalable and fault-tolerant network. In next chapter, we use experiment in real cloud environment and simulations to evaluate the performance of STB and compare with the tree-based and another recursive architecture, namely DCell.

CHAPTER 5

EVALUATION

This chapter presents the experimentation and data gathering method for evaluating the proposed STB architecture. It explains the tools used for implementation, data generation and the method used for processing data. The experimentation in our research is divided into two components. First, we deploy Hadoop MapReduce model in the testing cloud DCN to emulate a real cloud computing environment. This is used to evaluate the throughput performance of STB with other selected architectures. Secondly, we use NS2 to simulate our proposed STBRouting in a large scale DCN environment to evaluate the performance of fault-tolerant as well as throughput.

The remainder of this chapter is as follows. Section 5.1 presents the test-bed of this implementation. Section 5.2 presents the experimental scenarios from implementation and simulation. Section 5.3 presents the data collected in evaluating the selected performance metrics and the conclusion is given in the last section.

5.1 Test-Bed

This research focuses on network performance in cloud-oriented DCN. As it is difficult to deploy an actual large-scale data center infrastructure, the test-bed is established in 3 steps. First, we establish a small-scale private cloud environment for the real experimentation in this research. Second, virtual machines (VMs) are deployed in these real physical servers to establish a medium-scale DCN. At last, Network Simulator 2 (NS2) is used to simulate a relative large-scale DCN. By doing this, if the performance of selected network metric is better and better with the increasing number of servers in both real and simulation testing situation, then we can deduce that our proposed network architecture

still can achieve the expected target in an actual large-scale DCN.

The small-scale network is established by using totally twelve desktops, one laptop and seven switches. Selected server is a DELL OPTIPLEX GX520 desktop with Intel 3.2GHz dual-core CPU, 4GB RAM, and 80G hard disk. Each server also installs 3 Broadcom NetXtreme 57xx Gigabit network adapters. The operation system (OS) used in servers is Ubuntu 11.10 32-bit. The laptop is Lenovo Ideapad Z470 with 2.3GHz CPU, 4GB RAM, 640GB hard disk, and Ubuntu 11.10 32-bit OS. The switches used in this experimentation are two different models. Six mini-switches are Cisco Linksys WRT54GL Wireless-G broadband router with 4-port, and one Cisco Catalyst 2950 switch with 24-port. The latest released software/firmware was loaded on all equipment. Java with JDK version 1.7.0, Hadoop MapReduce 1.2.1, VMware Workstation 9.0.2, Python 2.7, and Ntop 3.3.10 are pre-installed in the physical servers. The bandwidth of server to switch is setting to 100 Mbps and that of switch to switch is setting to 1Gbps.

In order to test the performance of proposed STB architecture in medium-scale DCN, VMware Workstation 8 (Ward, 2002) is installed in the servers to emulate multiple Virtual Machines (VMs) acting servers in our test-bed. VM is a virtualized machine in physical devices. Such VM has its own virtualized CPU, RAM, hard disk, network adaptor, as well as OS and applications. The applications and hosts in the network are unable to distinguish the VM that looks like a real host exists in the network. VMware Workstation provides multiple virtual network establishing approaches that can be used to deploy the proposed architectures in the test-bed. In this research, 2 VMs are created in each server and each VM is assigned with 1GHz CPU, 1GB RAM, 10G hard disk. VM uses bridged model connecting to its host with 100Mbps virtual network adapter. In this case, all VMs act as real servers in the proposed DCN architecture.

Network Simulator 2 (NS2) (McCanne et al., 1997) is used to simulate the proposed STBRouting mechanism and the relative large-scale DCN. NS2 is an object-oriented,

open source network simulation tool which offers many modules embedded in host and supports routing protocols simulation for wired and wireless networks. NS2 runs on Unix-based OS, but also can runs on Cygwin platform in Windows-based OS. In this test-bed, NS2 with 2.33 version is selected.

Ubuntu Ntop is a open source network traffic tool running in linux-based OS, which shows the current network usage in real time. It monitors and reports information concerning the traffic generated by a list of hosts in network. A major benefits of using Ntop is a webpage based management user interface (UI). We can use a web browser to manage and navigate the throughput of each node to analyze the network performance.

In order to verify the data generated by the implementation and simulation, the t-Test analysis tool was used. The T-Test is a statistical data analysis and hypothesis test tool. It is used to test if the null hypothesis is true or not. In this research, the aim was to know whether there is a significant difference among the throughput performance of STB and other selected architectures. Thus, the Paired Two-Sample for Means T-Test is selected to analyze the sample data generated from the test.

5.2 Scenarios

Two kinds of experimental scenarios, implementation and simulation, are carried out to study the throughput of STB and selected architectures.

5.2.1 Implementation

MapReduce is one of the most widely used data processing models in cloud computing, which provides a programming and execution model for processing and generating large data sets using distributed computing mechanism. From the name of MapReduce we can see that it can be divided into two operations, Map and Reduce. In Map operation, the master node divides input value into smaller sub-values and distributes them to worker nodes. In Reduce operation, the master node collects the results from all workers and

combines them in some way to form the output. In this research, we use the execution of MapReduce to evaluate the throughput performance of the proposed STB architecture.

Word count is a typical example in MapReduce, which reads text files and counts how often words occur. Taking a simple example in figure 5.1, we assume that a sample text file including a list of words contents. After running the MapReduce word count application, an output file with the contents of word and its occur times are generated which shown in figure 5.2. From the figure we can see that the word “hadoop” and “python” occur 5 and 4 times, respectively.

Figure 5.1: A Simple test file of Word Count for MapReduce

In this research, readers do not need to understand the procedure of word count running in detail, but have to know that an all-to-all session happens among servers during the execution of MapReduce. The all-to-all session indicates that each source node forwards data packet to all destination nodes like a broadcast within a DCN, which also shows the essential part of cloud computing that is all servers (no idle one) participate in data processing and forwarding. In this experiment, we execute the word count application in

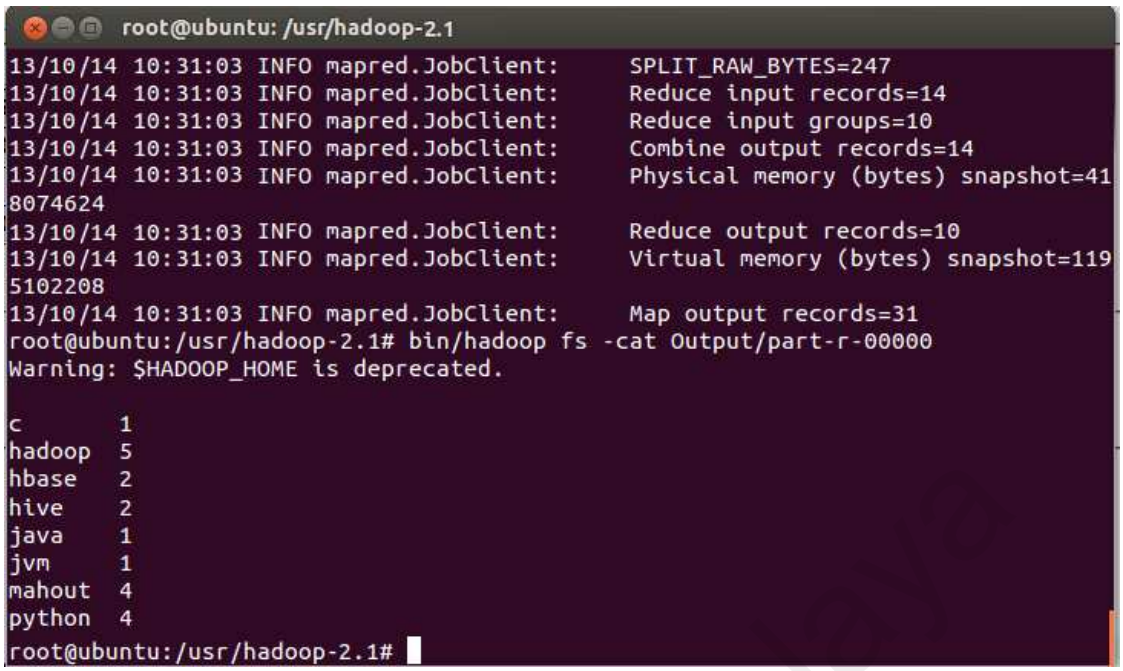

Figure 5.2: A Simple test file of Word Count for MapReduce

a fixed number of servers but different network architectures to evaluate the throughput performance of our proposed STB, tree-based, and another recursive DCell architectures.

5.2.1.1 Testing File Generating

As a major purpose of MapReduce is processing big data, we first, need a text file with large size to be tested in this experiment. We use python to generate a text file including a list of 100 million IP addresses as shown in figure 5.3. Each of the IP addresses is "192.168.x.x", and the last two bytes are selected from 0 – 255 randomly. The source code of sample file generation is attached in appendix A.

From the python shell in figure 5.4 we can see that generating the 100 million IP addresses needs about 15 minutes. The size of text file is about 1.5 GB. Before executing the MapReduce word count application in the experiment, we first write a python script to calculate the top ten IP addresses that have the most occur times in the list. It is worth to mention that the result shown in figure 5.5 is not generated by MapReduce but python as a benchmarking to check whether the MapReduce results in each architectures are correct.

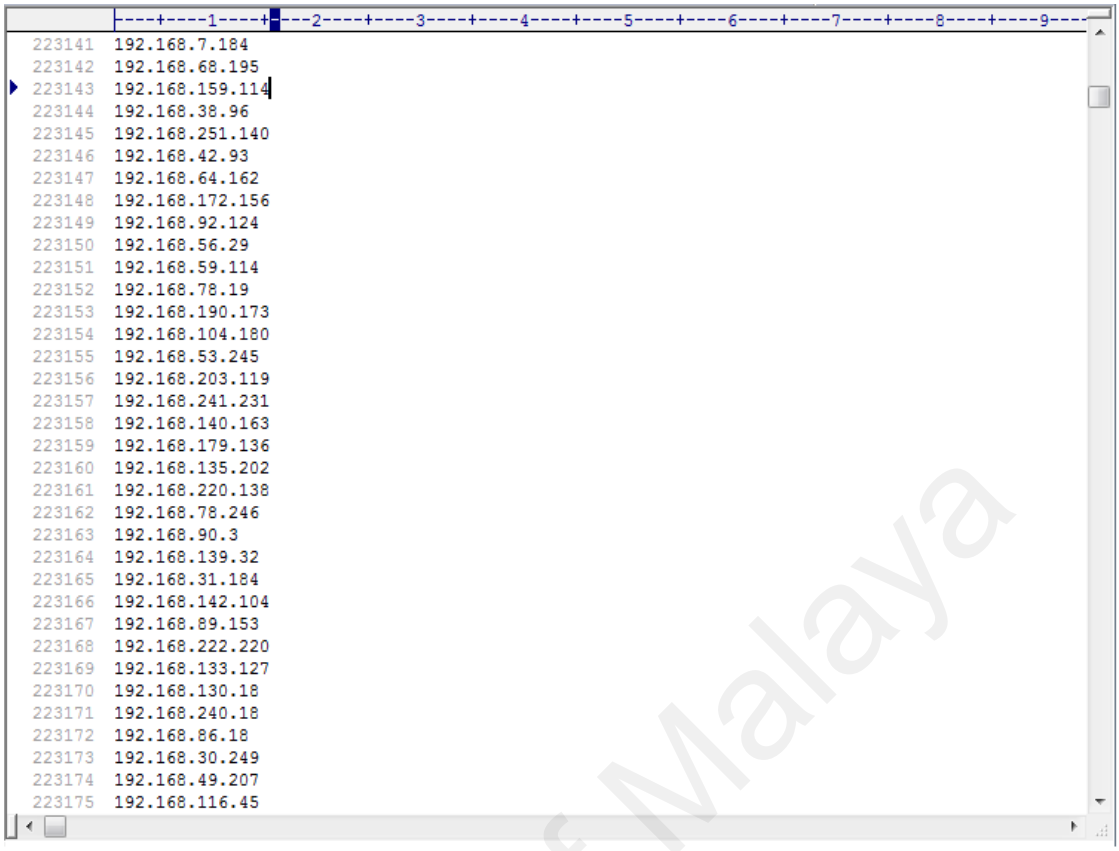

Figure 5.3: The IP Addresses List in Sample File

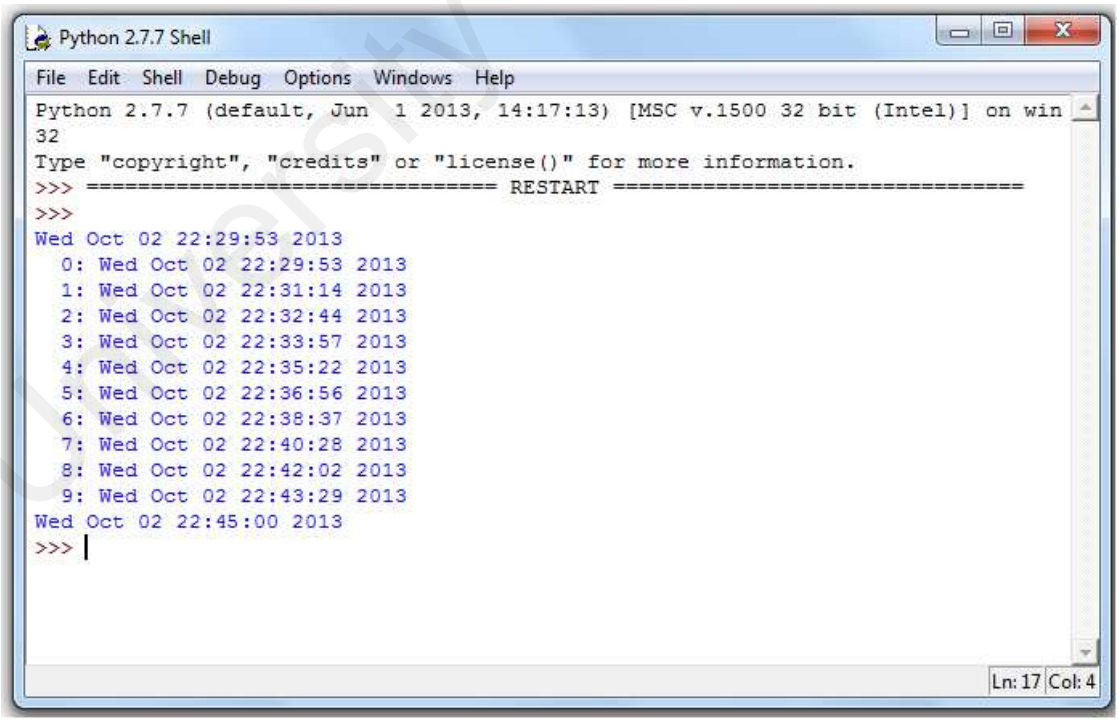

Figure 5.4: Python Shell

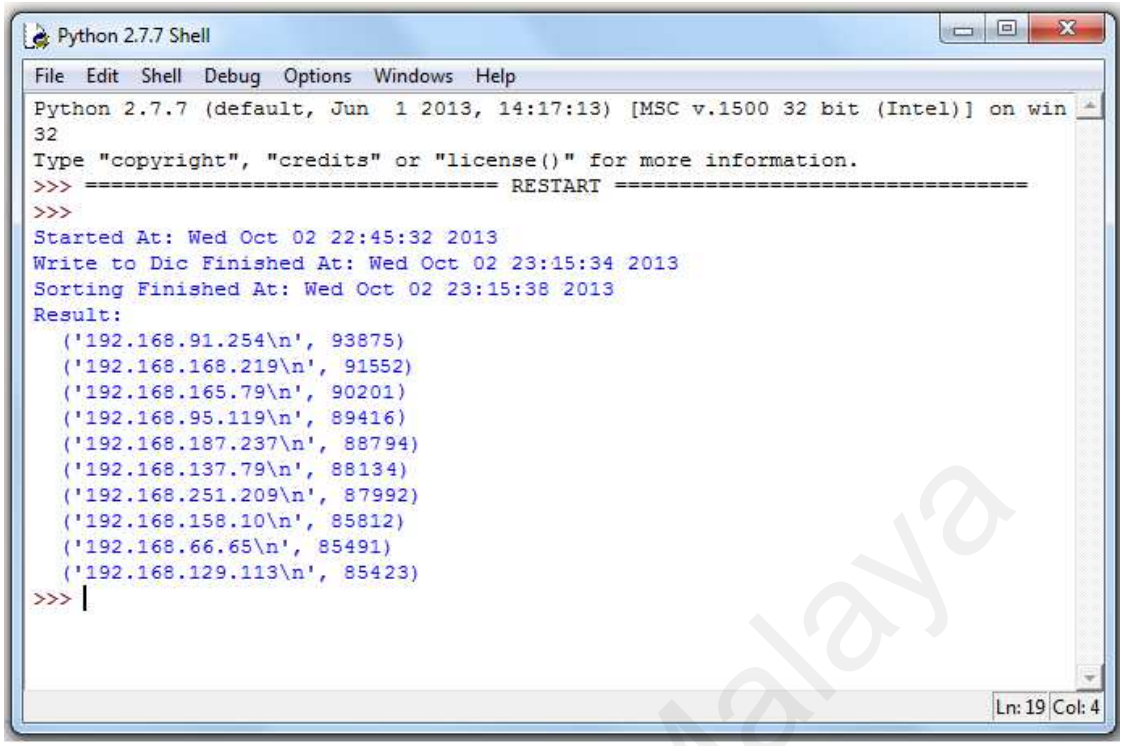

Figure 5.5: Top Ten IP Addresses with Most Occur Times

The python script for calculating the word count is enclosed in appendix B.

5.2.1.2 Hadoop MapReduce Deployment

Since we already have a text file as sample, we need to establish a MapReduce cluster based on our proposed STB network architecture. Figure 5.6 shows the topology of the proposed STB in this experiment, we first select the server with IP address 192.168.0.2 as a master server and the rests 11 servers are slaves.

Then we add the IP addresses of slave servers in master node to establish our MapReduce cluster. Figure 5.7 shows the commands on Master server for adding slaves' IP addresses to the cluster. The command includes the IP address and role of each server, as well as their hosts' names. The configuration file for establishing the MapReduce Cluster is attached in the appendix C.

We can access to the master server by using web browser to check the status of our MapReduce cluster. From figure 5.8 we can see that the first ten slave servers and a cluster

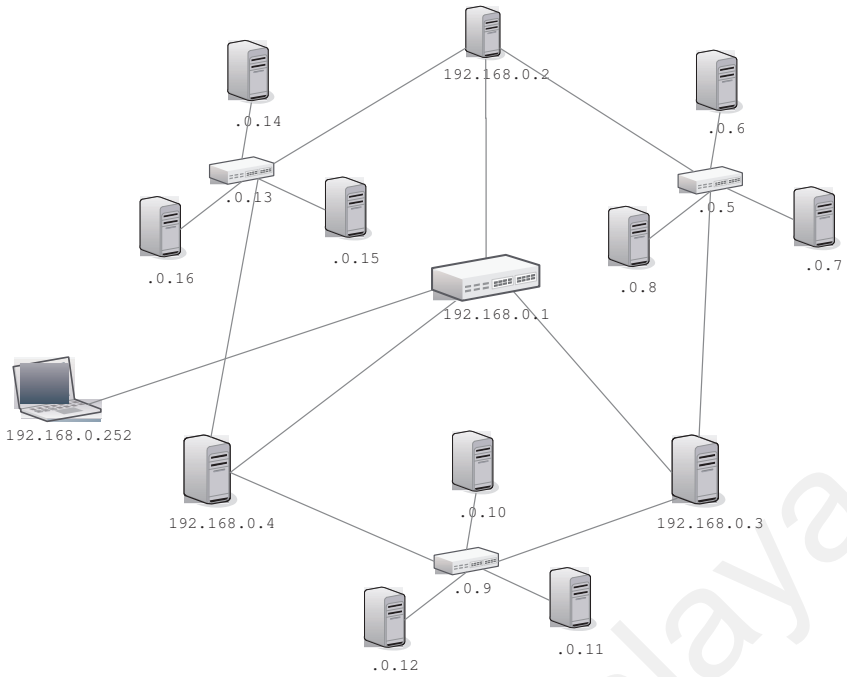

Figure 5.6: Network Topology of STB

```
1
2  192.168.0.2-master,namonode,jobtracker-master
3  192.168.0.3-slave,datanode,tasktracker-slave0
4  192.168.0.4-slave,datanode,tasktracker-slave1
5  .
6  .
7  .
8  192.168.0.18-slave,datanode,tasktracker-slave10
9
```

Figure 5.7: Adding IP Addresses of Slaves in Master Server

with total twelve number of servers is established in our network. It is worth to mention that as our physical server has multiple network adaptors that introduced in section 5.1, we have already assigned the IP address for each adaptor in the same subnet and gateway in our experimentation. However, we only use one IP address to identify a physical server is for clear expression concern in this section.

After establishing the MapReduce cluster in the proposed STB network, we create a new MapReduce project called “WordCountqh” in master server for uploading and testing the previously generated text file. Figure 5.9 shows that the new “WordCountqh” Java class is already successfully created in the master server.

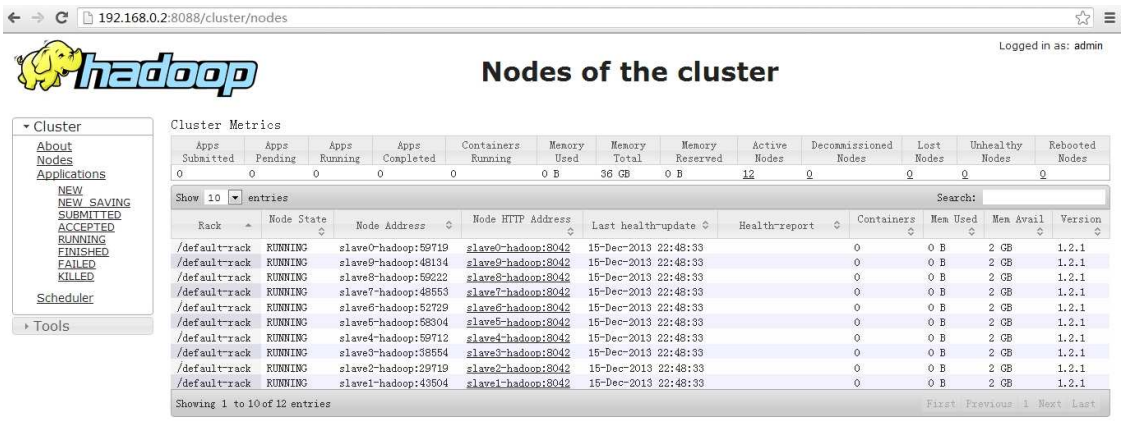

Figure 5.8: The Status of MapReduce Cluster

Figure 5.9: Creating WordCountqh Project

We then upload the previous generated sample text file to the Hadoop distributed file system (HDFS) in the master server. Figure 5.10 indicates that the sample file is successful uploaded in the system.

5.2.1.3 Execution

We run the WordCountqh project in the MapReduce cluster based on our proposed STB architecture. Figure 5.11 shows the execution procedure of word count application, which indicates the related information such as how many data processing groups are generated, how many input/output records happened. The execution time of this experiment is 117

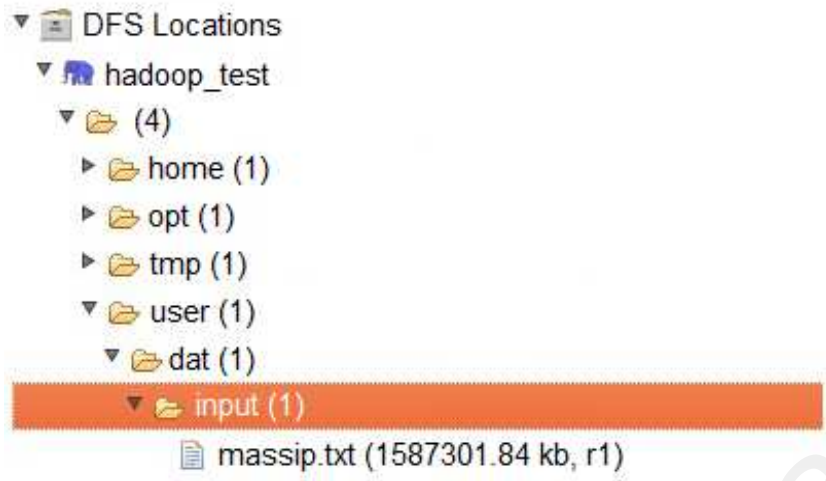

Figure 5.10: Uploading Sample File to HDFS

seconds.

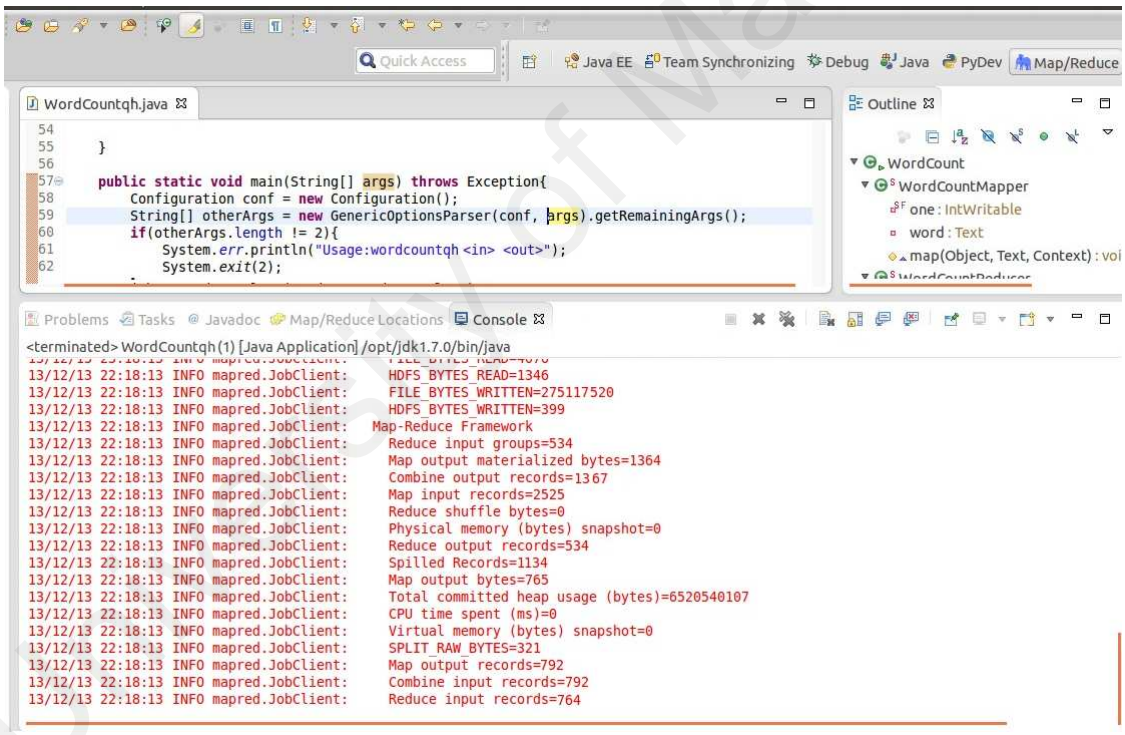

Figure 5.11: Execution of WordCountqh Project in STB Network

After running the word count application, we gather an output file named as “part-r-0000”, which is shown in figure 5.12. It is worth to mention that the result of this execution is same with the benchmark result collected by using python script in section 5.2.1(a).

Since the purpose of this experiment is focusing on throughput performance of the

Figure 5.12: Network Topology

STB network, we do not pay too much attention on the details of MapReduce processing and the result. The DCN infrastructure from all twelve servers was deployed in accordance with published specifications. However, the DCN scale with only twelve servers is too "small". For evaluating our proposed STB architecture in a "medium" scale DCN, based on the previous network topologies, each server creates 2 virtual machines acting servers to deploy a medium-scale DCN with 24-server. We did not create more than two VMs in a physical server is because the extra VMs will impact the execution efficiency of system in the server, which may bring an inaccurate results to the experiments.

Each VM in the network has a 1 GHz CPU, 1G ROM and 30G hard disk with Ubuntu 11.10 32-bit OS. VM uses bridged model connecting to its host with 100Mbps virtual network adapter. In this case, all VMs act as real servers in the proposed STB network. Then we re-execute the experiment as before to evaluate the throughput performance of STB architecture in a 24-server medium network.

Until now, the experiments under the scenarios of 12 and 24 servers in STB net-

work architecture are already done. We used word count application in MapReduce to generation data flow, and Ntop to monitor and analyze the throughput performance in the experiment. The two selected architectures DCell and Tree-based for comparison are also evaluated in the same way. The reason of selecting DCell to compare in the experiment is that DCell is a hierarchical recursive structure. As we mentioned in chapter 2, optical/wireless architecture normally needs more deploying cost for their high-advanced network devices. VL2 and Monsoon are focusing on load balancing solutions. As a typical architecture in the category of hierarchical recursive structure, DCell attracts a lot of attentions from different research organizations in recent years (X. Wang et al., 2013)(X. Wang et al., 2014)(Kumar et al., 2012).

The topologies of the two architectures are shown in figure 5.13 and 5.14. The servers with IP addresses 192.168.0.5 and 192.168.0.8 are selected as master servers in each of topology respectively.

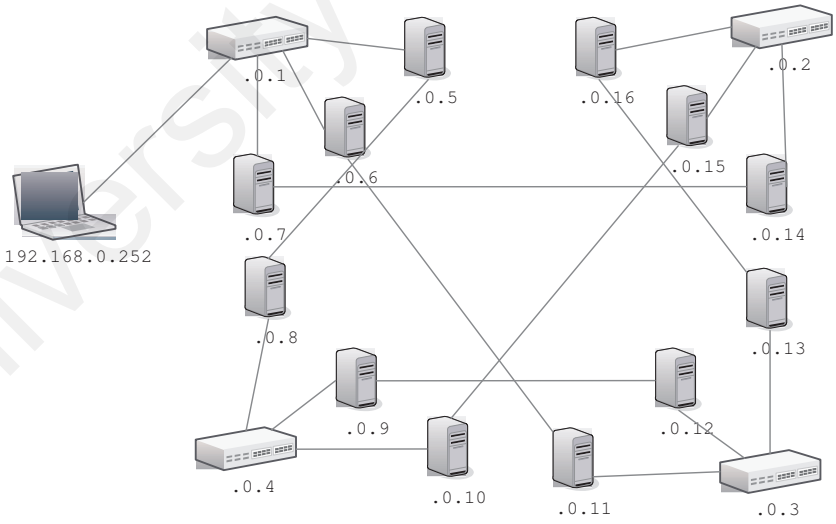

Figure 5.13: Network Topology of DCell Architecture

5.2.2 Simulation

Due to the restriction of current number of physical servers, we are not able to evaluate the performance of our proposed STB architecture in a real large-scale DCN. In addi-

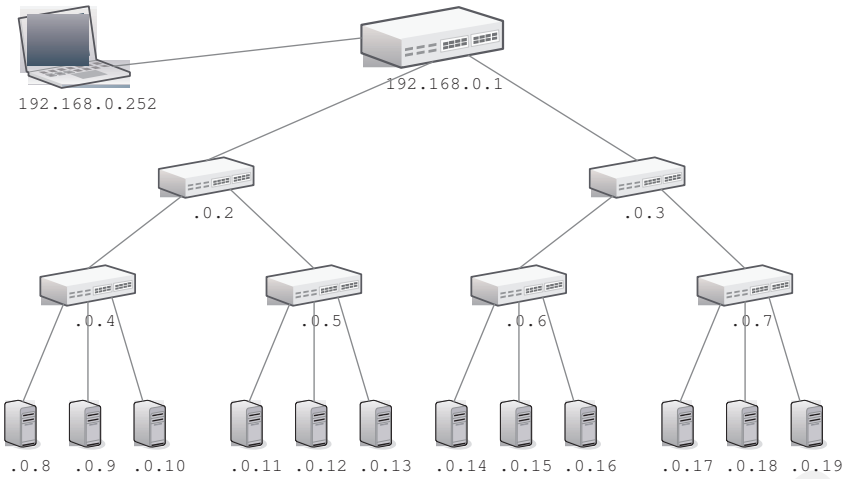

Figure 5.14: Network Topology of Tree-based Architecture

tion, the existing network devices cannot be used to implement the proposed STBRouting scheme. Therefore, we performed our simulation for evaluating network throughput and angle-based fault-tolerant routing scheme performance in a large-scale network. In this simulation, we use Network Simulator 2 to simulate the proposed STB, DCell and tree-based network architectures.

5.2.2.1 STBRouting in NS2

For implementing our proposed STBRouting scheme, we add the angel-based node identification, and route discovery and recovery schemes in NS2. We first create a folder named as stb including *stb.h*, *stb.cc*, *stb_pkt.h*, *stb_rtable.h*, and *stb_rtable.cc* five files under ns2.33. The main roles of each files are defining timer and route agent, executing timer, route agent and Tcl script, declaring routing protocol in NS2, declaring stb routing table, and routing table execution. Data forwarding procedure of STBRouting is similar with that of Routing Information Protocol (RIP), the difference is route discovery in STBRouting is based on angel and level of node in the structure. According to the route discovery scheme of STB introduced in chapter 4, node check the angle and level values of potential next hops before data forwarding. Figure 5.15 shows the source code of packet forwarding and route discovery procedure in *stb.cc* file. We simply modified the

rttable.h file under *routing* folder in NS2.33 by limited the neighbor hops to 2, used in our STBRouting.

```
if (ch->direction() == hdr_cmn::UP &&
    ((u_int32_t)ih->daddr() == ID_ANGELLEVEL || ih->daddr() == ra_addr())) {
    dmux_->recv(p, 0);
    return;
}
else {
   float mina, float hdr_nha, float hdr_nhb, float hdr_dsta;
    {   mina=(abs(hdr_dsta-hdr_nha)? abs(hdr_dsta-hdr_nhb));
        return;}
    ID_ANGELLEVEL = mina;
    ch->direction() = hdr_cmn::DOWN;
    ch->addr_type() = NS_AF_INET;
    if ((u_int32_t)ih->daddr() == ID_ANGELLEVEL || ih->angel())
        ch->next_hop() = ID_ANGELLEVEL;
    else {
        nsaddr_t next_hop = rtable_.lookup(ih->daddr());
        if (next_hop == ID_ANGELLEVEL) {
            debug("%f: NO ROUTE TO FORWARD %d\n",
                CURRENT_TIME,
                ra_addr(),
                ih->daddr());
            drop(p, DROP_RTR_NO_ROUTE);
            return;
        }
        else
            ch->next_hop() = next_hop;
```

Figure 5.15: Source Code of Packet Forwarding and Route Discovery

We modified the *node.h* file in NS2 by adding two columns to indicate the value of angle and level for each node. The original node identification in NS2 is using *number* starting from 0. In STB, this column is used for indicate the role of node is server or switch, by using *null* or 1. We also modified the *trace.cc* and *cmu − trace.cc* files under *trace* folder of NS2.33 to make our proposed STBRouting can be recognized by NS2 in final simulation. Figure 5.16 shows the modified source code in *trace.cc* and *cmu − trace.cc* files.

5.2.2.2 Simulation

We evaluate our proposed $STB(3)$ architectures in three different scenarios from level-2 to level-4 in area of $1000m \times 1000m$. We set the bandwidth between each pair of nodes (no matter server or swtich) is duplex $1Gbps$ link with $10ms$ propagation delay. Figure

```
//trace.cc
void Trace::recv(Packet* p, Handler* h)
    {
    format(type_, src_, dst_, p);
    hdr_cmn *th = hdr_cmn::access(p);
    th->nodid=src_;
    pt_->dump();
    callback();
    pt_->namdump();
    if (target_ == 0)
        Packet::free(p);
    else
        send(p, h);
    }

//cmu-trace.cc
    CMUTrace::recv(Packet *p, Handler *h)
    hdr_cmn *th = hdr_cmn::access(p);
    th->nodid=src_;
```

Figure 5.16: Source Code of Modified Trace File

5.2 shows the values of parameters between two nodes, and the rest nodes and links in the architectures have the same parameters as well. Each node uses a *DropTail* queue, of which the maximum size is 10. An *UDP* agent and a *NULL* agent are attached to both of the nodes to establish UDP connections. Moreover, a *constant bit rate* (*CBR*) traffic generator is attached to *UDP* agents to generate 1*Kbyte* packets and 100*Mbps* transmission rate. The *Cbrgen* tool of NS2 are used to create the traffic overload. We also use a random perturbation scheme in data forwarding process to simulate expected node and link failure. In our proposed STB architecture, servers are used for not only processing data but also forwarding as the extra installed network adaptors. However, the data forwarding capacity of server is normally lower than that of switches. Since we are focusing on the throughput performance of the proposed architecture in DCN, such difference can be omitted. In this work, we assume that switches and servers can provide same bandwidth and data forwarding capacity.

In the first scenario of the simulation, each server node of $STB_2(3)$ establishes an UDP connection to each of remaining servers, and send 1GB data in each of UDP con-

nection. Total 1.482TB data will be forwarded on the 1482 connections. In next simulations, we keep the total aggregated throughput as 1.482TB. Because the number of UDP connections in $STB_3(3)$ and $STB_4(3)$ are 14,280 and 129,240,respectively, we set 100MB and 10MB in each UDP connections in the scenarios of $STB_3(3)$ and $STB_4(3)$, respectively. There is no disk access in this simulation for separating network performance from disk IO. We study the aggregate throughput of STB architecture and its fault-tolerant performance with different link failure ratios from 0-20 %. This simulation is running on a Dell Optiplex 990 desktop with Intel core i5-2500 3.3GHz CPU, 4GB Memory, 500GB hard disk in Windows 7 professional OS. We repeat our simulation 30 times to obtain the average results.

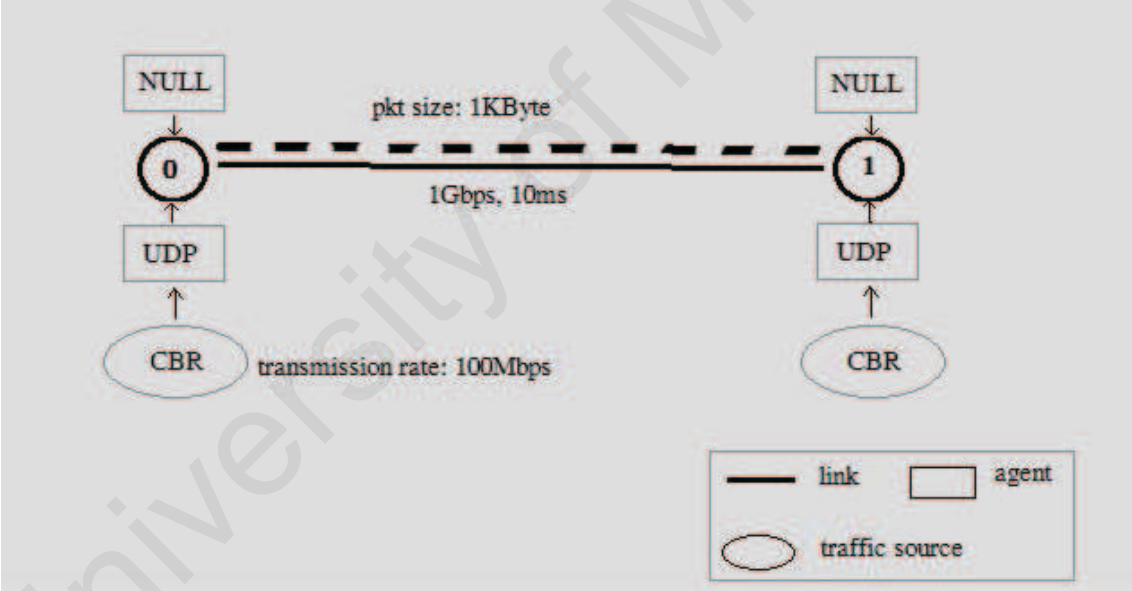

Figure 5.17: Network Topology

We first run the simulation in the scenario of $STB_2(3)$, that is 39 servers and 13 switches, without link failure. Then we set the link failure ratios from 2% to 20% to see the aggregated throughput performance in the scenario. Similarly, we implement the simulation in the scenarios of $STB_3(3)$ and $STB_4(3)$ to see the throughput with different link failure ratios.

In order to further compare the aggregated throughput of STB with that of DCell

and Tree-based when the three architectures hold the same number of servers, we run the simulations as previous steps.

5.3 Data Collection and Performance Metrics

Three performance metrics are selected to evaluate the performance of proposed STB and other architectures in this experiment.

5.3.1 Throughput

Throughput is always a major parameter in network evaluation, especially for data center network. A higher throughput means a greater data transmission and faster information exchanging. In this experiment, we evaluate throughput of the proposed STB architecture by calculating the proportion of sending and receiving packets with a period of time in all nodes.

5.3.1.1 Implementation

We use Ntop to monitor the network throughput during the execution of experimentation. Figure 5.18 shows the dashboard of Ntop running on the laptop (192.168.0.252) indicating the throughput and traffic information of part of servers at the beginning stage the implementation.

Figure 5.18: Dashboard of Ntop

At the end of this experiment, we use “Network Load Statistics” command to see

the last ten minutes aggregated throughput of the STB network that is shown in figure 5.19. The figure indicates us that the maximum and minimum aggregated throughput are 7.9Gbps and 449.1Mbps, respectively, and the Average value is 4.0Gbps. This diagram is generated by Ntop automatically, and the result will be used for further comparison with selected DCell and Tree-based architectures.

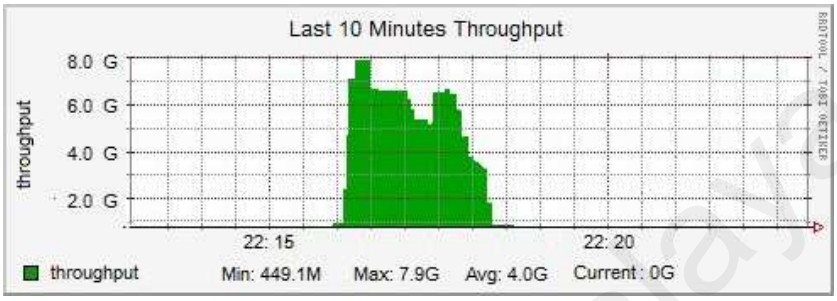

Figure 5.19: Aggregateion Throughput

Ntop supports saving the information of original data flow in the experiment with different formats, such as text, Perl, PHP, and Python, for further analyzing by using other tools. Figure 5.20 shows the information about data flow in master server with Python format.

```
'192.168.0.2' => array(
'hostResolvedName' => '192.168.0.2',
'pktSent' => 26748379,
'pktRcvd' => 36277336,
'ipv4BytesSent' => 3851818480,
'ipv4BytesRcvd' => 5223936426,
'bytesMulticastSent' => 0,
'pktMulticastSent' => 0,
'bytesMulticastRcvd' => 0,
'pktMulticastRcvd' => 0,
'bytesSent' => 3851818480,
'bytesRcvd' => 5223936426,
'ipv4BytesSent' => 3851818480,
'ipv4BytesRcvd' => 5223936426,
'ipv6BytesSent' => 0,
'ipv6BytesRcvd' => 0,
'tcpBytesSent' => 3851818480,
'tcpBytesRcvd' => 5223936426,
'udpBytesSent' => 43852,
'udpBytesRcvd' => 0,
'icmpSent' => 0,
'icmpRcvd' => 0,
),
```

Figure 5.20: Data Flow Recorded in Master Server

5.3.1.2 Simulation

After the end of simulation in each scenario, a trace file is automatically generated by NS2. This trace file is used to record the whole simulation process by indicating the sending and receiving time of each data packet, source and destination nodes of transmission and so on. By analyzing the trace file, it is possible to obtain details of the simulation. Figure 5.21 gives an example of trace file in our first simulation.

```
+ 0.1 1,30 2,0,w cbr 1000 ------- 2 1,30.0 0,60.1 0 0
- 0.1 1,30 2,0,w cbr 1000 ------- 2 1,30.0 3.1 0 0
+ 0.109 1,30 2,0,w cbr 1000 ------- 2 1,30.0 3.1 1 1
- 0.109 1,30 2,0,w cbr 1000 ------- 2 1,30.0 3.1 1 1
r 0.113 1,30 2,0,w cbr 1000 ------- 2 1,30.0 3.1 0 0
+ 0.113 2,0,w 0,60 cbr 1000 ------- 2 1,30.0 3.1 0 0
- 0.113 2,0,w 0,60 cbr 1000 ------- 2 1,30.0 3.1 0 0
+ 0.117 1,30 2,0,w cbr 1000 ------- 2 1,30.0 3.1 2 2
- 0.117 1,30 2,0,w cbr 1000 ------- 2 1,30.0 3.1 2 2
s 0.128 1,30 2,0,w cbr 1000 ------- 2 1,30.0 3.1 1 1
+ 0.128 2,0,w 0,60 cbr 1000 ------- 2 1,30.0 1,330.1 3 3
- 0.128 2,0,w 0,60 cbr 1000 ------- 2 1,30.0 1,330.1 3 3
+ 0.129 1,30 2,0,w cbr 1000 ------- 2 1,30.0 1,330.1 3 3
- 0.129 1,30 2,0,w cbr 1000 ------- 2 1,30.0 1,330.1 3 3
r 0.129 2,0,w 0,60 cbr 1000 ------- 2 1,30.0 1,330.1 3 3
r 0.131 1,30 2,0,w cbr 1000 ------- 2 1,30.0 1,330.1 3 3
+ 0.131 2,0,w 0,60 cbr 1000 ------- 2 1,30.0 1,330.1 3 3
- 0.131 2,0,w 0,60 cbr 1000 ------- 2 1,30.0 0,300.1 4 4
+ 0.134 1,30 2,0,w cbr 1000 ------- 2 1,30.0 0,300.1 4 4
- 0.134 1,30 2,0,w cbr 1000 ------- 2 1,30.0 0,300.1 4 4
r 0.135 2,0,w 0,60 cbr 1000 ------- 2 1,30.0 0,300.1 4 4
r 0.135 1,30 2,0,w cbr 1000 ------- 2 1,30.0 0,300.1 4 4
+ 0.137 2,0,w 0,60 cbr 1000 ------- 2 1,30.0 1,124.2 3 3
- 0.137 2,0,w 0,60 cbr 1000 ------- 2 1,30.0 1,124.2 3 3
+ 0.142 1,30 2,0,w cbr 1000 ------- 2 1,30.0 1,124.2 5 5
s 0.142 1,30 2,0,w cbr 1000 ------- 2 1,30.0 1,124.2 5 5
r 0.146 2,0,w 0,60 cbr 1000 ------- 2 1,30.0 1,124.2 6 6
r 0.146 1,30 2,0,w cbr 1000 ------- 2 1,30.0 1,124.2 7 7
+ 0.147 2,0,w 0,60 cbr 1000 ------- 2 1,30.0 1,124.2 7 7
- 0.147 2,0,w 0,60 cbr 1000 ------- 2 1,30.0 2,345.1 9 9
+ 0.148 1,30 2,0,w cbr 1000 ------- 2 1,30.0 2,345.1 9 9
- 0.149 1,30 2,0,w cbr 1000 ------- 2 1,30.0 2,345.1 9 9
r 0.155 2,0,w 0,60 cbr 1000 ------- 2 1,30.0 2,345.1 8 8
```

Figure 5.21: Data Flow Recorded in Master Server

The fist column indicates the status of packet, where $+/-$ means the packet *into*$/$*out from* a queue, r/s means *receive*$/$*send* by a node. The event occurred time stamps and locations are recorded from the second to forth columns. The fifth to seventh columns shows the traffic type, packet size and flag, respectively. The data flow ID is recorded in the eighth column. The ninth and tenth columns indicate the information of source and destination nodes using a.b format, where a is node ID and b is port number. The last two columns shows the sequence number and ID of packet. Taking the first line in figure 5.21 as an

example, it shows that a packet with $ID = 0$, data flow $ID = 2$, sequence number is 0, $1000 - byte$ length in CBR traffic, is being send from source node $(1,30)$ to destination node $(0,60)$. The packet is forwarded into the queue of an intermediate node $(2,0,w)$ at $0.1second$ simulation time.

It is common for a trace file to be very large size and makes it inconvenient to read it line-by-line. Therefore, AWK script file is developed to analysis and gather results in NS2. In this simulation, the awk script is used to first calculate the total size of packet during the period of source sending $start\_time$ to destination receiving end_time. Then the script changes byte to bit (1 byte = 8 bits) and bps to $Mbps$ (1 Mbps = 1000000 bps)to output the final throughput result. Equation 5.1 shows the function of throughput calculation and the awk script is attached in the appendix D.

$$AggregatedThroughput = \frac{SizeofReceivedPacket \times 8}{(Endtime - StartTime) \times 1000000} \tag{5.1}$$

(Bianchi, 1998)

5.3.2 Number of Supported Servers

Network scalability is a major issue in the consideration of DCN architecture design, especially for cloud-oriented DCNs. We always desire to deploy as many as possible servers with limited number of network devices in a given network performance condition. A main contribution of this research is to provide a large-scale architecture. The number of server supported in the proposed STB architectures can be calculated by using Theorem 1 where we have provided in chapter 3. We also calculate the number of servers in DCell and tree-based architectures to compare the server proportions in different networks.

5.3.3 Average Shortest Path Length

The average shortest path length means the average length between each pair of nodes in the network. The length of two nodes is the minimal number of links from one node reaching to another. Because NS2 uses Dijkstra shortest path algorithm by default to calculate the optimal path between each pair of nodes, thus we use the Dijkstra's algorithm (Knuth, 1977) to calculate the average shortest path length in the simulation.

5.4 Data Analysis Tool

In order to verify the data generated by the experimentation and simulation, the T-Test analysis tool was used. The T-Test is a statistical data analysis and hypothesis test tool that used to test whether the null hypothesis is true or not. Three types of T-Test are Two-Sample Assuming Equal Variances, Two-Sample Assuming Unequal Variances, and Paired Two-Sample for Means. All of the different tests are used to take a two-sample student's t-test. The aim of using T-Test in this research is to know whether there is a significant difference between the performance of throughput and average shortest path length of DCN before and after using the proposed STB architecture. Because the main results of the Paired Two-Sample for Means t-test show whether the means of two-sample data before and after a process are equal or not. Therefore, we select the Paired Two-Sample for Means t-test to analyze the sample data generated from the experimentation and simulation.

5.5 Conclusion

This chapter presented the evaluation of proposed STB architecture and STBRouting scheme. In order to efficiently and effectively evaluate the proposed architecture and its routing scheme, we first tested our model in a small private cloud DCN in real experimentation from 12 to 24 servers. Then we evaluated STB in a simulation to test the throughput performance and fault-tolerant routing scheme in large scale network. The

approaches of data collection in the experiment and simulation are also introduced in this chapter. We selected aggregation throughput, number of supported servers, and average shortest path length as performance metrics to evaluate and compare among STB, DCell and tree-based architectures. The evaluation results and data analysis will be presented in next chapter.

CHAPTER 6

RESULTS ANALYSIS AND DISCUSSION

In this chapter, analysis is made on the results obtained from the experiment and simulation. We present the results of the aggregated throughput, number of supported servers, and the average shortest path length as performance metrics to evaluate the performance of the proposed STB, and compare with DCell and Tree-based architectures. We also present the statistical results of the T-Test to validate the data generated by the experimentation and simulation. The chapter is organized as follows. Section 1 presents throughput analysis from the experimentation to simulation results. The analysis of number of supported servers and average shortest path length are discussed in section 2 and 3. We then explain the result of data validated by using T-Test in section 5, and the chapter is concluded in section 6.

6.1 Throughput

Throughput is always a major parameter in network evaluation, especially for data center network. A higher throughput means a greater data transmission and faster information exchanging. In our experiment work, we evaluate aggregated throughput of the proposed STB architecture by calculating the proportion of sending and receiving packets with a period of time in all nodes.

6.1.1 Experimentation Result Analysis

Figure 6.1 plots the aggregate throughput of STB, DCell and that using the tree-based architecture with 12 servers. At the beginning stage of simulation, STB has a higher throughput than DCell and tree-based as it has more servers to participate in data forwarding. The transmission in tree-based completed at 146.7 seconds that spent more time than

that of STB and DCell, where are 82.5 and 83.809 seconds, respectively. STB and DCell is about 2 times faster than tree with the maximum aggregate throughput is 7.945Gbps and 7.657Gbps, respectively, but the tree-based architecture has 3.6Gbps only. The performance of STB does not have much significant difference with DCell is because both of the two are recursive architectures and have same number of 12 servers and 4 switches.

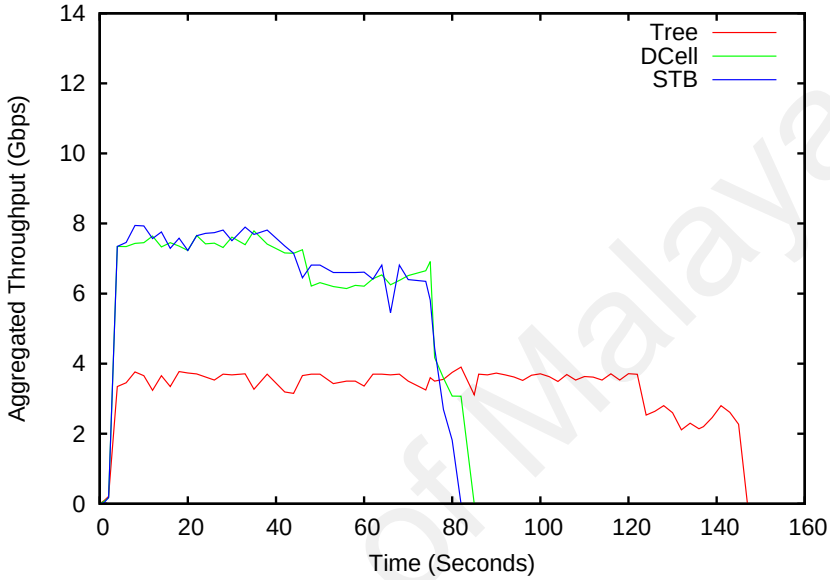

Figure 6.1: Aggregated Throughput with 12 Servers

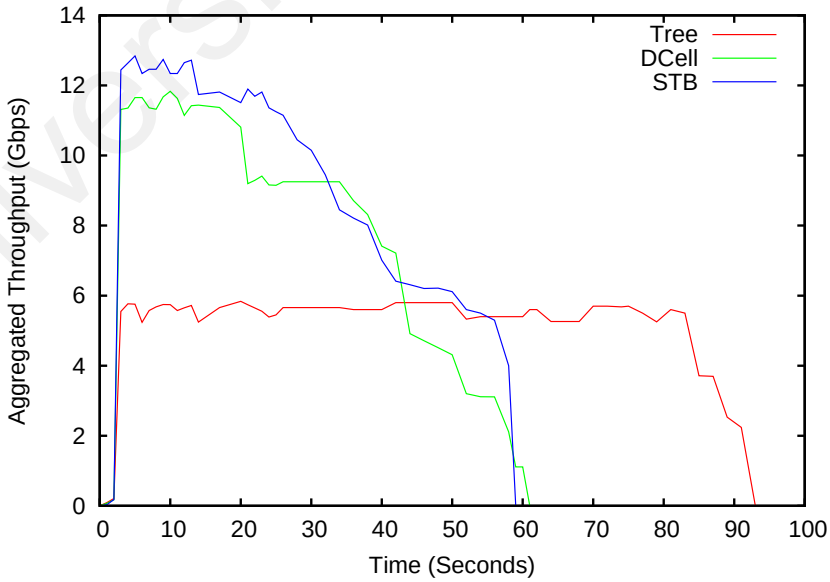

Figure 6.2: Aggregated Throughput with 24 Servers

Figure 6.2 shows the aggregate throughput of the three architectures with 24 servers. We can see that the values of aggregated throughput in the three architectures are all

increased with the number of server increasing. Throughput in all architectures decreases over the time, however STB is slower because the multiple parallel links in it. In this scenario, the word count task is completed by tree-based at 92.631 seconds. By contrast, STB and DCell use 59.326 and 60.89 seconds to complete the task. Our proposed STB architecture still has better throughput performance and execution speed than that of Tree-based and DCell even the differences are still not too much with DCell architecture in 24 servers network.

6.1.2 Simulation Result Analysis

Since the performance of aggregated throughput in STB and DCell are almost same in small-scale networks, we used simulation to evaluation the throughput performance in the three architectures in large-scale networks. The analyzing is divided into two parts, without and 2-20% link failure ratios.

6.1.2.1 Without Link Failure

Network without link failure provides a reliable environment to evaluate the theoretical results of aggregated throughput in our simulation. Figure 6.3 indicates the aggregated throughput with 39 servers and without link failure. As each server establishes 38 UDP connections to the rest of server nodes, the total transmitted data is 1.482TB, which is much more than that of running MapReduce application in our experimentation using physical devices. Therefore, the tree-based network uses 1,520 seconds to complete data forwarding. As we can see from the figure that, the different throughput performance between STB and DCell is becoming clear, where the maximum througput in STB is 14.83Gbps and the task ends at 925.2 seconds by comparing with 13.87Gbps and 1049.7 seconds in DCell.

Figure 6.4 and 6.5 give the throughput performance with 120 and 363 servers in the different architectures. With the increasing number of servers, the throughput perfor-

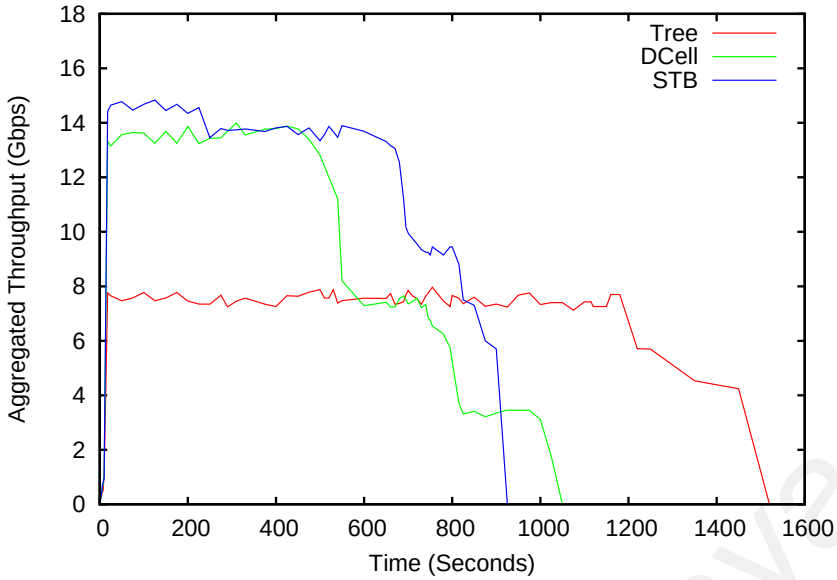

Figure 6.3: Aggregated Throughput with 39 Servers and without link failure

mance of STB is getting better and better than the other architectures. In the scenario of $STB_3(3)$ for example, the maximum throughput value of STB is 23.987Gbps, and packets exchanging completed in 538 seconds. By contrast,the maximum throughput for DCell and tree-based network are observed from 23.51Gbps to 13.437Gbps. In $STB_4(3)$ scenario, the maximum throughput value of STB is reached to 46.55Gbps, which is almost 2 times than that of $STB_3(3)$ because the number of servers is increased from 120 to 363.

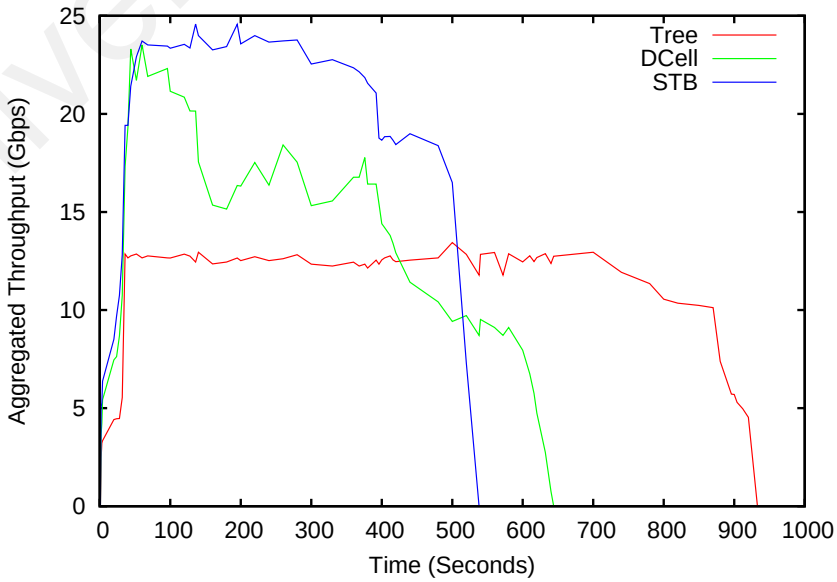

Figure 6.4: Aggregated Throughput with 120 Servers and without link failure

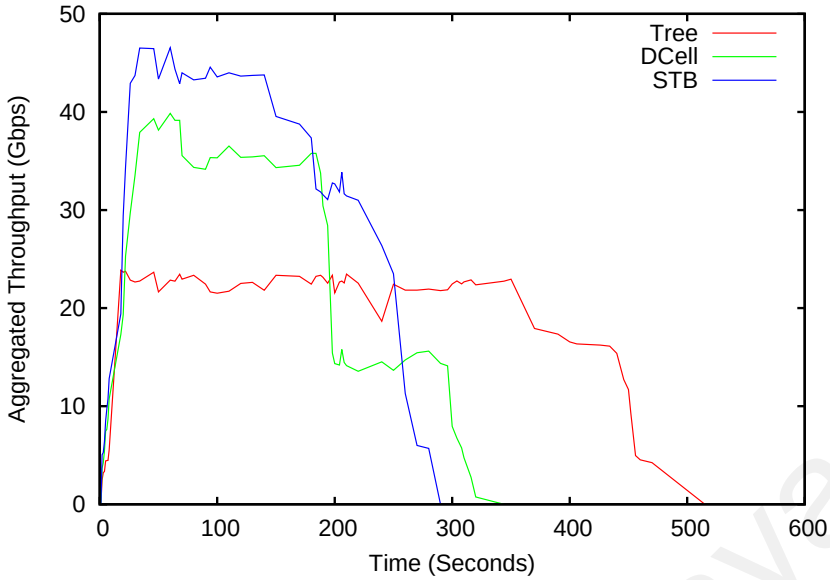

Figure 6.5: Aggregated Throughput with 363 Servers and without link failure

6.1.2.2 With Link Failure

We use a proportion of link failure approach to evaluate our proposed STBRouting scheme by observe the average throughput performance with different link failure ratio.

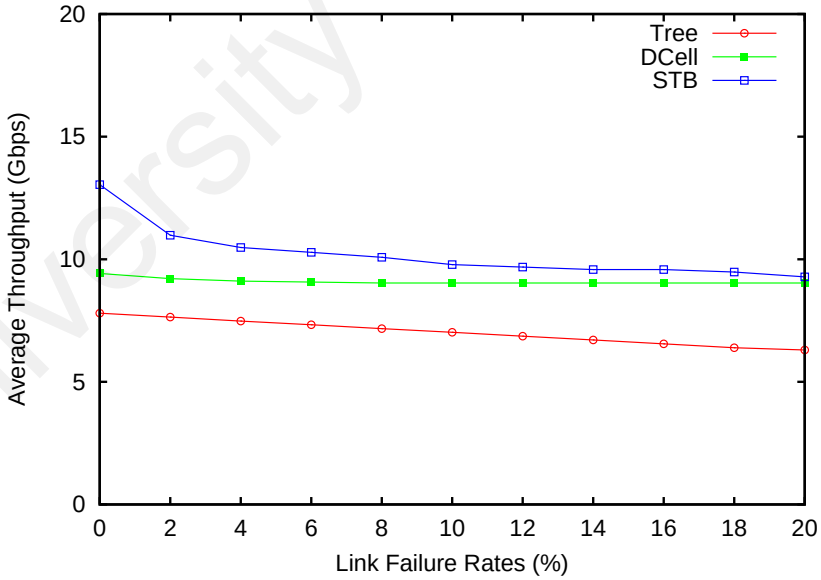

Figure 6.6: Effect of Link Failures with Different Ratio on Average Throughput with 39 Servers

Figure 6.6 - 6.8 plot the comparison of average throughput of STB, DCell and Tree-based architecture in 39, 120 and 363 servers with different link failure ratios. STB and DCell achieve very high throughput, but that of STB degrades more rapidly because of

more server-to-switch connections. DCell uses more server than STB to forward packet which impacts the transmission speed due to the forwarding capacity of server is lower than switch. The tree-based network has the lowest performance as no redundancy devices are deployed in our simulation. With the increasing number of servers, the influence of throughput performance in STB is getting smaller and smaller by different link failure ratio, but not too much in that of DCell. By contrast, the tree-based architecture is always impacted by the link failure.

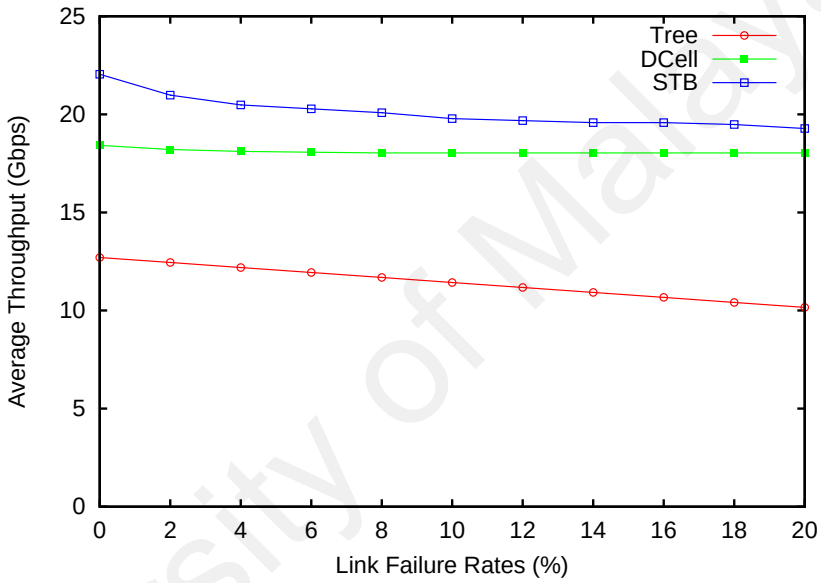

Figure 6.7: Effect of Link Failures with Different Ratio on Average Throughput with 120 Servers

6.1.2.3 Disscussion

STB achieves higher aggregated throughput performance than DCell and tree-based network architectures with the network scale increased. We observed that, at the beginning of our experimentation and simulation, STB has a better performance than tree-based network because server participates packet forwarding. Although more switches are deployed in the tree-based architecture to support given number of servers, the network throughput does not get improvement due to the bandwidth bottleneck in aggregation and core layer of the network. STB and DCell are all recursive structure that use multiple

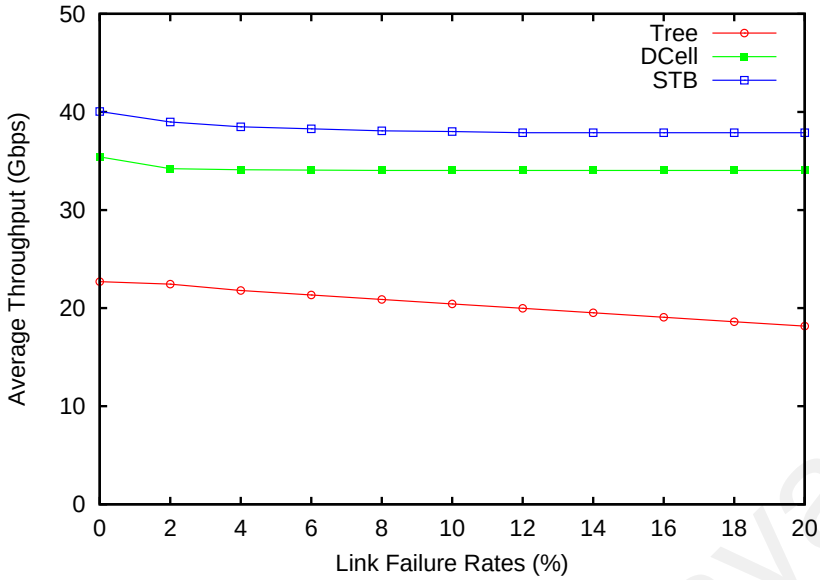

Figure 6.8: Effect of Link Failures with Different Ratio on Average Throughput with 363 Servers

small units to establish higher level structures. Therefore, the throughput performance between STB and DCell is no too many differences in small-scale networks. However, with the increasing number of servers, STB has higher aggregated and average throughput than DCell as STB has more connections. The average throughput of DCell is more stable than that of STB because the recursive unit in DCell connects to all other units in same level, which means packet can also be forwarded among different units without switch in same level. However, the approach using large number of servers to transfer packet impacts data forwarding rate. A case may occur that a major part of server process and forward data at the same time but some switches are idle.

6.2 Rates of Server Utilization

In this research, the rates of server utilization refers to a proportion of number of servers to entire nodes in DCN. In a same DC deploying budget condition, we always desire as little as possible network devices to support as many as servers. In other words, if two architectures have same network performance, the one which owns less switches but more servers, it has a better cost-performance ratio.

In this section, we use the proved theorem 1 in chapter 4 to calculate the number of supported servers and switches in STB. Assuming the initial recursive unit is combined by 1 switch connecting to k servers, we take the level value of n=6, and calculate the the number of servers in the $STB_6(3)$, $STB_6(4)$, $STB_6(5)$ and $STB_6(6)$ are 3,279, 21,844, 97,655 and 335,922, respectively (see Table 6.2). For supporting such number of servers, 1,093, 5,461, 19,531, and 55,987 network devices are deployed in STB. By contrast, the number of network devices in the tree-based architecture is 1,629, 7,285, 24,418, and 67,189, respectively. Similar with STB, it is worth to mention that, the DCell architecture is a hierarchical recursive structure also, it has same performance with STB in the proportion of server utilization, that is $\frac{k}{k+1}$).

Table 6.1: the proportion of servers in STB and Tree-based architectures

Architecture	No.of Servers	3,279	21,844	97,655	335,922
STB	No.of nodes	4,372	27,345	117,186	391,909
	Proportion of Servers(%)	75	80	83.3	85.7
$Tree-based$	No.of nodes	4,908	29,129	122,073	403,111
	Proportion of Servers(%)	66.8	75	79.9	83.3

At the stage of $STB_6(3)$ to $STB_6(4)$, 1 switch connects with servers from 3 to 4. We can calculate that the proportion is increased from $\frac{3}{4}$ to $\frac{4}{5}$. In this analogy, the value of proportion will tend to 100 when $k \to +\infty$. As compared to tree-based architecture, more network devices are needed in aggregate and core layers to supporting such number of servers, so the proportion is definitely lower than that of STB and DCell. Figure 6.9 indicates the proportion of server utilization between STB and tree-based architectures in same number of servers.

6.3 Average Shortest Path Length Analysis

The average shortest path length means the average length between each pair of nodes in the network. The length of two nodes is the minimal number of links from one node

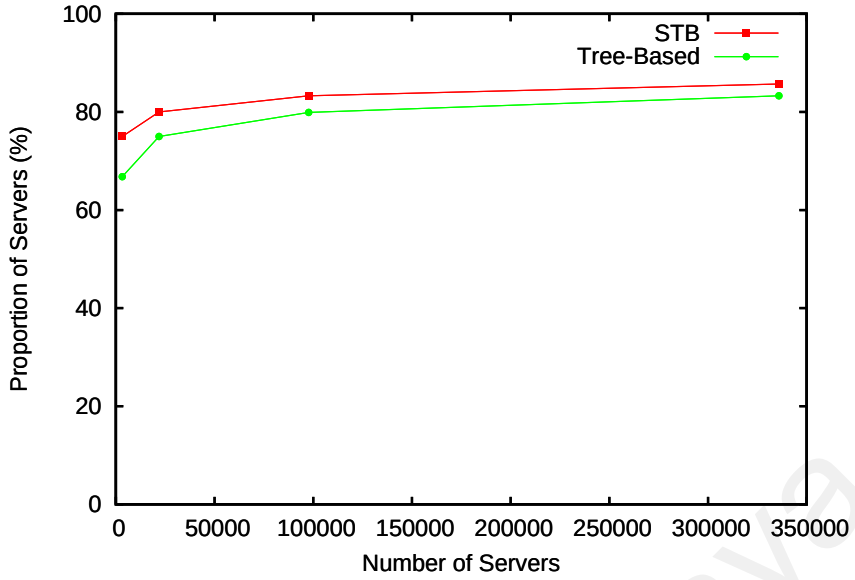

Figure 6.9: Proportion of Server Utilization in STB and Tree-based Architectures

reaching to another. We use the Dijkstra's algorithm (Knuth, 1977) to calculate the shortest path length in the simulation. Firstly, we assume there is no node failure happens in the network. Table 6.2 shows that the average path length from one S_0 server to the rest servers in different levels of STB. The $STB_n(k)$ (k=3,4 and 5)indicate the number of servers connecting one switch in the initial S_0 level. We can see that the longest average path lengths in the table is when $STB_4(3)$ in level 4, and the rest values are all no more than 5 hops, which proves the Theorem 2 in section 4.3.

Table 6.2: the average path length from S_0 server to the rest servers in different levels of STB without node failure

	Level	No. of SVs	No. of SWs	Sum. of SPL	Avg. of SPL
$STB_2(3)$	2	39	13	63	1.658
$STB_3(3)$	3	120	40	405	3.4
$STB_4(3)$	4	363	121	1812	5.005
$STB_2(4)$	2	48	21	160	3.404
$STB_3(4)$	3	340	85	1344	3.964
$STB_2(5)$	2	155	31	325	3.11

Figure 6.10 shows the average of shortest path length and levels of STB in contrast with DCell (when $k = 3$ in those of two architectures). When $n = 0$, the 2 architectures have only basic (lowest) level, thus we have value of 1 for average of shortest path length

as those servers are directly link to switch. With the level increase, the more servers are able to be supported in each of architecture and the length of shortest path for each server is increased as well. As we can see from the figure that, DCell has a faster growing with the increasing of n, meanwhile STB is growing slower than it. At the beginning stage, the average length of STB is same with DCell, however, when $n = 4$, DCell almost have 2.5 times value than that of STB in this parameter. The reason of this condition happens is due to the complicated recursive method in DCell.

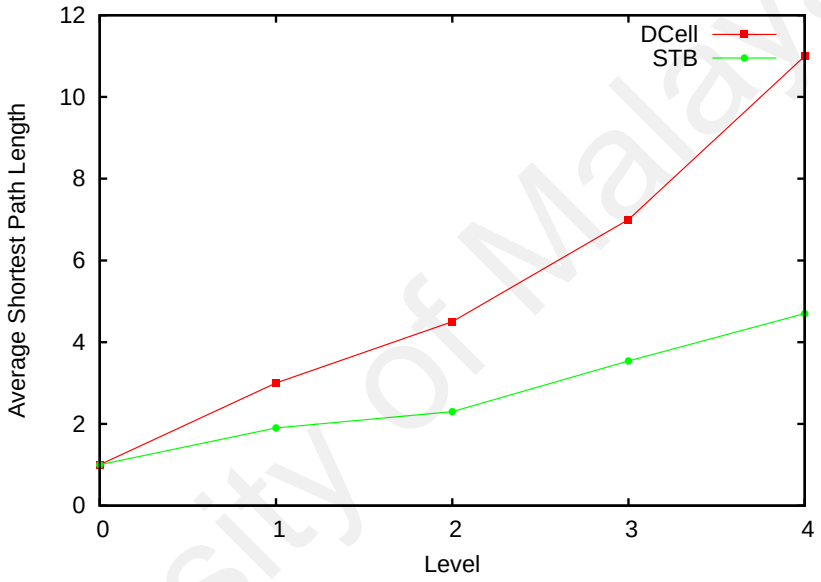

Figure 6.10: when $k = 3$, Average Shortest Path Length in $0-4$*levels* without server failure

Secondly, we evaluate the STB architecture in some servers failure happens situation. In this test, the proportion of server failure has been set in the beginning stage of simulation. Figure 6.11 shows the relations between the proportion and average of shortest path length in a S_4 network. As we can see from the figure that, when the proportion remains between 0.02-0.2, the average length has a limited wave range between 5.0-6.7.

STB has a better performance at average short path length comparing with DCell when the rate of server failure increasing. As only one level of switches in DCell, almost all of data flows are transferred by servers. STB has a shorter average length because all data packets are forwarded by switch. In section 4.3 we have proved that STB has

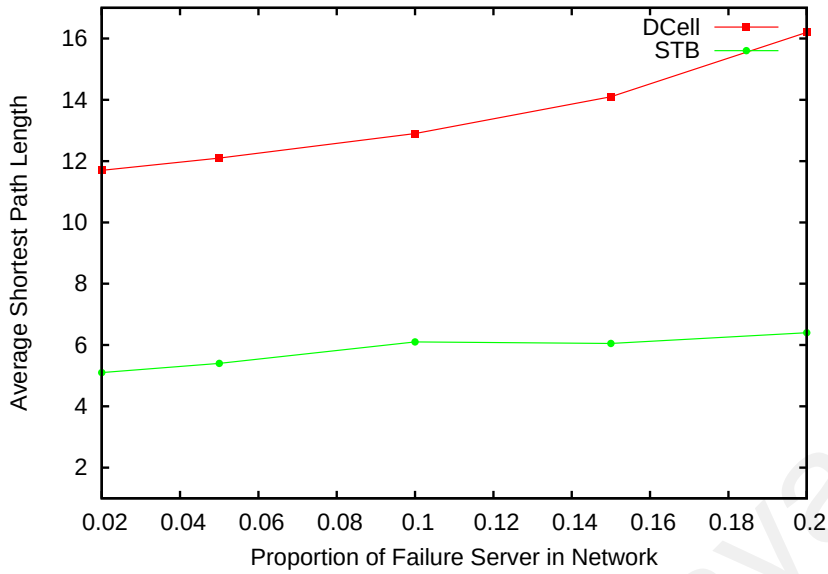

Figure 6.11: Server Failure and Average of Shortest Path Length in S_4 STB Network

a multiple parallel links between each two servers, hence data packets can be delivered through switch or server when a node failure happens in one link. Therefore the STB has the best performance contrast with DCell architectures.

6.4 Scalability and Fault-Tolerance

In this section, the performance of scalability and fault-tolerance are proved by using theoretical derivation.

The maximum possible number of servers that can be supported by using 128-port switches is 20,000 around in the tree-based DCN. The main reason for the situation is the number of supported hosts is limited by available port density and the requirement of fast failure recovery mechanism in the three layer architecture. Both the above statements are already proved in section 3.1.3.

By contrast, the scale of STB is unlimited because its recursively defined structure. As we mentioned in section 4.2, the construction of higher level STB is typically by adding a new initial recursive unit (1 mini switches with 3 servers in the case as an example) between one pair of servers in current level. Hence, the number of supported devices can be exponential increased with the level growing in STB. In theory, the num-

ber of devices supported in STB is unlimited. It is worth to mention that, according to the equation 4.1-4.6 from theorem 1 to 4, the number of supported devices can be calculated easily. Hence, the scalability of STB network can be easily controlled by increasing or decreasing the number of connected server in the initial recursive unit.

The issue of fault-tolerance in STB actually, has already been proved in section 4.4.3. The equation 4.7 in theorem 4 indicates that each pair of two servers have 2 parallel paths at least, and 2^{2n} at most. The parallel paths are the links between pairs of servers, which is independent to others. Those pair of two servers can be selected from any position, any levels in STB, and no matter if they connected to the same switch or not. Suppose a link failure happens, there are enough number of parallel paths can be selected to forward data in STB. Hence, from figure 6.6 - 6.8 we can see that the performance of average throughput with different link failure rates in STB is much better than that of the tree-based architecture. In addition, with the increasing number of servers, more parallel paths are supported, and the influence of throughput performance in STB is getting smaller and smaller by different link failure ratio. In this research, a major feature about the fault-tolerance is, different than the tree-based architecture, the parallel paths existing in STB without any redundancy devices support.

6.5 Data Validation

We select the Paired Two-Sample for Means t-test to analysis the sample data generated from the simulation and verify whether there is a significant difference between the performance of throughput of the proposed STB architecture and the DCell and tree-based, respectively. Based on the 30 times simulations, the values of $P(T \leqslant t)two-tail$ are 0.003497545 and 0.006544028, which are less than 0.05, mean that there are significant differences in STB as compared to DCell and tree-based, in the performance of throughput when proportion of server failure happens.

Results indicate that STB has a better performance than DCell and tree-based. As we known that the bandwidth between switches is normally higher than that of switch and server, and it is also restricted by the lowest value between two servers in data forwarding, it is obviously that data forwards between switches is much faster than servers with same path length. A major feature of STB is using fewer switches connecting more servers, which ensure that the nodes located between two servers are switches, to provide a higher transfer efficiency.

6.6 Conclusion

In this chapter, we have presented the data analysis and discussion of our proposed STB architecture by comparing with DCell and tree-based. The best application scenario for STB is large data centers. Due to the restriction by the number of devices in current experiment, we cannot deploy a real large-scale for testing the STB architecture. However, we use limited devices, virtual machines and simulator to test the proposed model. After the evaluation in different scenarios, we found a trend that the throughput performance of STB is better and better with the number of server increased. Therefore, we believe that, the STB architecture has a potential capacity to have better throughput performance in a real large-scale cloud-oriented data center network.

CHAPTER 7

CONCLUSION

This chapter provides the conclusion on the achievements from the research – the significance and contribution of the research outcomes. It also contains recommendation on areas for future research.

7.1 Evaluation on Achievement of Objectives

The research reported in this thesis involves the development of the Sierpinski Triangle Based (STB) data center network architecture for large-scale cloud-oriented data centers. The purpose of STB is to provide a higher throughput and server utilization, a shorter shortest path length, as well as a fault-tolerant routing mechanism in DCNs. STB uses Sierpinski triangle fractal to deploy a hierarchical and recursive network structure in which higher level STB network is combined by multi lower level STBs. This means that the bottleneck of aggregation throughput in tree-based can be solved in STB. In the process of routing selection, STB routing mechanism provides at least two parallel paths for each pair of two servers in DCN without redundancy devices. The performances of the STB, tree-based, and DCell architectures have been evaluated using the real experiment and NS2 simulator from the perspectives of network throughput, average shortest path length, and proportion of servers to switches.

The objective of literature review is achieved in chapter 2, where the existed architectures such as tree-based, Fat-tree, DCell, BCube, CamCube, VL2, and so on, for DCNs were reviewed to provide an overview and relevant knowledge on the subject matter of the research. A thematic taxonomy of the state-of-the-art architectures is also produced in this chapter.

Chapter 3 presents the performance analysis of current widely used tree-based architecture in DCN from the perspectives of throughput restriction, network scalability, cost of construction, and resource fragmentation. We point out the potential problems using the tree-based architecture in cloud-oriented DCN and give a benchmark experiment of the problem issues for verification.

Design of the proposed STB architecture is achieved in chapter 4. STB uses server with multiple network adapters and mini-switch to construct the recursive network by adding a number of $0-level$ triangle structures on the $(n-1)-level$ to construct $n-level$ Sierpinski triangle structure. This construction method can effectively increase the throughput by removing the aggregate layer contrast to the tree-based architecture in DCN. Moreover, STB uses an angle based node identification scheme to describe the position of inter node and provide a fault-tolerance routing in DCN.

The objective of evaluation is achieved in chapter 5 where presents the use of real experiment and NS2 simulator to test and simulate the environment to apply the STB and selected architectures. Knowledge was gained on how to deploy the proposed architectures, establish the DCN and create network components.

In chapter 6, the behaviors of the STB, tree-based, and DCell are evaluated. The performances of throughput, average shortest path length, rates of server utilization, scalability and fault-tolerance have been compared among the selected architectures. The results of the experiment show that the STB architecture has higher throughput, shorter average shortest path length, and higher proportion of servers to switches than traditional tree-based, and DCell architectures. In addition, the better performance of scalability and fault-tolerance in STB to the tree-based is also proved by using theoretical derivation. Furthermore, the results are validated by using T-Test to verify the authenticity in the last section of this chapter, which achieved the objective of data validation.

7.2 Contributions

The main contribution of this work is our proposed STB network architecture which provides the goals of large-scale data center network design, especially for cloud-oriented data centers. By comparing with Tree-based, and DCell architectures in simulation and real experiment, the STB has a better performance in proportion of switches to servers, average shortest path length, and network throughput as well, and the results have been verified by the theoretical analysis and simulations. The major contributions from this research can be summarized as follows:

1. Establishment of taxonomy to analyze the implications and critical aspects of existing DCN architectures and making a comparison among the architectures on the basis of significant parameter metrics.

2. Design of the STB architecture in order to improve performance of DCN in large-scale cloud-oriented data centers

3. Writing of the codes of the proposed STB routing mechanism.

4. Establishment of the measurement parameters to evaluate the performance of STB.

5. Recommendation of areas for future research endeavors.

This research results were accepted in the following papers.

1. **Han Qi**,Muhammad Shiraz,Abdullah Gani, Md Whaiduzzaman,Suleman Khan. Sierpinski Triangle Based Data Center Architecture in Cloud Computing. The Journal of Supercomputing. 69(2), 887-907, DOI: 10.1007/s11227-014-1187-9(ISI Indexed Q2)(Qi, Shiraz, Gani, et al., 2014)

2. **Qi Han**, Muhammad Shiraz, Liu Jie Yao, Abdullah Gani, Zulkanain Bin Abdul Rahman, Data Center Network Architecture in Cloud Computing: Review, Taxonomy, and Open Research Issues. Journal of Zhejiang University SCIENCE C, 15(9), 776-793.

DOI: 10.1631/jzus.C1400013. (ISI-Indexed Q3)(Qi, Shiraz, Liu, et al., 2014)

3. **Han Qi**, Abdullah Gani, Research on mobile cloud computing: review, trend and perspectives. 2012 Second International Conference on Digital Information and Communication Technology and it's Applications (DICTAP),Bangkok, Thailand, 16-18 May 2012, pp 195-202 (ISI IEEE, Indexed).(Qi & Gani, 2012)

Papers in collaboration with others:

1. Suleman Khan, Muhammad Shiraz, Ainuddin Wahid Abdul Wahab, Abdullah Gani, **Qi Han**, Zulkanain Bin Abdul Rahman, A Comprehensive Review on Adaptability of Network Forensics Frameworks for Mobile Cloud Computing The Scientific World Journal, vol. 2014, Article ID 547062, 27 pages, 2014. doi:10.1155/2014/547062.(Khan et al., 2014)

2. Muhammad Shiraz, Ejaz Ahmed, Abdullah Gani, **Qi Han** Investigation on Runtime Partitioning of Elastic Mobile Applications for Mobile Cloud Computing Journal of Supercomputing, , Volume 67, Issue No. 1, pages 84- 103 January 2014, DOI:10.1007/s 11227-013-0988-6(ISI Indexed Q2)(Shiraz et al., 2014)

7.3 Strength and Weakness

There are several strengths and weaknesses in undertaking this research, and they are summarized as follows:

7.3.1 Strength

1. STB architecture improves the throughput performance of DCN. High-level STBs are built recursively from several low-level STBs. STB uses only mini-switches to replace high-advanced network devices to scale out. Therefore, STB is able to support large-scale DCN without using core switches/routers.

2. STB deploys the Angle based node identify mechanism to replace the IP based address allocation for supporting the scale of DCN with exponential growth and fault-

tolerant routing mechanism.

3. Owing to the difficulty to deploy a real large-scale DCN in our research, we used five PCs and one switch to establish a small-scale DCN, and NS2 to simulate our proposed scheme in large-scale network, which can reduce the costs of our research, effectively.

4. In this research, NS2 is used to deploy large-scale DCN. As the NS2 is an open source product, we can directly modify the source files to achieve our scheme.

7.3.2 Weakness

One of the limitations of this fault-tolerant routing scheme is the rerouting path may not be the optimal.

1. In this research, the experiment in real devices was deployed in a small-scale DCN, which may not full indicate the significant performance of STB in a large-scale.

2. The simulation environment of NS2 is a little bit simple and cannot measure all performance of STB architecture in DCN.

3. Parent node failure may cause part of children nodes failed.

This research work is limited in the performance evaluation among the selected DCN architectures. We only selected three metrics to compare that is not enough to evaluate the performance of proposed architecture objectively. It is important to note the other supplementary issues such as deployment cost, energy consumption, applicability of current routing protocol, wireless/optical network communication, security and others are not covered in this research.

7.4 Future Research Work

In this research, we present a novel architecture to improve performance of the DCN for cloud-oriented data centers. This method should not be used in large-scale DCN only, but is also suitable for use in some relative small-scale networks.

In the simulation done in this research, the STB routing mechanism is simulated

without experimented in real DCN environment. Thus the issue of implementing the proposed STB routing and Angle based node identify mechanisms in real device experiment are aimed to be addressed in our future research. In addition, the deployment of STB architecture in real experiment is taken in a small-scale DCN in this research due to the restriction of devices, which may not fully indicate the performance of the proposed STB architecture in large-scale DCN. Thereby, implementing the STB architecture in real large-scale DCN environment is also aimed in our future research.

In the near future, the hybrid-based structures could still be a major structure widely used in DCN. To be competitive, servers would need to obtain multi-core processor, large memory and multiple network adapters to improve the network performance capacity and participate data forwarding. However, there are several other technical and non-technical factors that need to be considered for further development. For example, servers often have lower performance in data transmission compared to switches. Also, the existing routing algorithms would meet a radical shift when they directly transplant from traditional network to large DCN. To overcome these concerns, an efficient dynamic schedule algorithm, DCN-oriented routing protocol with a significant advantage is necessary for providing high quality service to end users.

7.5 Conclusion

This thesis presents the report of the research conducted on the data center network (DCN) architectures in cloud-oriented data centers. By using the traditional tree-based architecture in DCN, the performance is affected because of the limitations in aggregation throughput, server utilization, bandwidth capacity and network scale. The performance of DCN can be enhanced by using the proposed Sierpinski Triangle Based (STB) architecture, which uses the well-known Sierpinski triangle fractal to deploy the structure together with the STB routing mechanism, to mitigate the throughput bottleneck in ag-

gregate layers as accumulated in tree-based architecture.

We evaluated the behavior of STB architecture developed in this research, analyzed its performance and compared it to the traditional tree-based, and DCell architectures in proposed DCN environments. The performance has been evaluated based on throughput, average shortest path length, and proportion of servers utilization using a real experimental test-bed and NS2 simulator. The results show that the STB architecture achieves higher network performance than the tree-based, and DCell architectures.

REFERENCES

Abu-Libdeh, H., Costa, P., Rowstron, A., O'Shea, G., & Donnelly, A. (2010). Symbiotic routing in future data centers. *ACM SIGCOMM Computer Communication Review*, *40*(4), 51–62.

Al-Fares, M., Loukissas, A., & Vahdat, A. (2008). A scalable, commodity data center network architecture. In *Acm sigcomm computer communication review* (Vol. 38, pp. 63–74).

Alon, N., & Roichman, Y. (1994). Random cayley graphs and expanders. *Random Structures & Algorithms*, *5*(2), 271–284.

Armbrust, M., Fox, A., Griffith, R., Joseph, A., Katz, R., Konwinski, A., et al. (2010). A view of cloud computing. *Communications of the ACM*, *53*(4), 50–58.

Baker, S. (2007). Google and the wisdom of clouds. *Business Week*, *14*.

Beimborn, D., Miletzki, T., & Wenzel, S. (2011). Platform as a service (paas). *Business & Information Systems Engineering*, *3*(6), 381–384.

Beloglazov, A., & Buyya, R. (2010). Energy efficient resource management in virtualized cloud data centers. In *Proceedings of the 2010 10th ieee/acm international conference on cluster, cloud and grid computing* (pp. 826–831).

Beloglazov, A., Buyya, R., Lee, Y., Zomaya, A., et al. (2011). A taxonomy and survey of energy-efficient data centers and cloud computing systems. *Advances in Computers*, *82*(2), 47–111.

Bhardwaj, S., Jain, L., & Jain, S. (2010). Cloud computing: A study of infrastructure as a service (iaas). *International Journal of engineering and information Technology*, *2*(1), 60–63.

Bianchi, G. (1998). Ieee 802.11-saturation throughput analysis. *Communications Letters, IEEE*, *2*(12), 318–320.

Borthakur, D. (2007). The hadoop distributed file system: Architecture and design. *Hadoop Project Website*, *11*, 21.

Boru, D., Kliazovich, D., Granelli, F., Bouvry, P., & Zomaya, A. Y. (2013). Energy-efficient data replication in cloud computing datacenters. In *Ieee globecom 2013 international workshop on cloud computing systems, networks, and applications (gc13 ws-ccsna), atlanta, ga, usa.*

Buxmann, P., Hess, T., & Lehmann, S. (2008). Software as a service. *Wirtschaftsinformatik*, *50*(6), 500–503.

Buyya, R., Yeo, C. S., & Venugopal, S. (2008). Market-oriented cloud computing: Vision, hype, and reality for delivering it services as computing utilities. In *High*

performance computing and communications, 2008. hpcc'08. 10th ieee international conference on (pp. 5–13).

Chang, F., Dean, J., Ghemawat, S., Hsieh, W., Wallach, D., Burrows, M., et al. (2008). Bigtable: A distributed storage system for structured data. *ACM Transactions on Computer Systems (TOCS)*, *26*(2), 4.

Chen, K., Singla, A., Singh, A., Ramachandran, K., Xu, L., Zhang, Y., et al. (2012). Osa: An optical switching architecture for data center networks with unprecedented flexibility.

Chen, Y., Alspaugh, S., Borthakur, D., & Katz, R. (2012). Energy efficiency for large-scale mapreduce workloads with significant interactive analysis. In *Proceedings of the 7th acm european conference on computer systems* (pp. 43–56).

Chen, Y., Griffith, R., Liu, J., Katz, R. H., & Joseph, A. D. (2009). Understanding tcp incast throughput collapse in datacenter networks. In *Proceedings of the 1st acm workshop on research on enterprise networking* (pp. 73–82).

Clausen, T., Jacquet, P., Adjih, C., Laouiti, A., Minet, P., Muhlethaler, P., et al. (2003). Optimized link state routing protocol (olsr).

Clos, C. (1953). A study of non-blocking switching networks. *Bell System Technical Journal*, *32*(2), 406–424.

Condie, T., Conway, N., Alvaro, P., Hellerstein, J. M., Elmeleegy, K., & Sears, R. (2010). Mapreduce online. In *Nsdi* (Vol. 10, p. 20).

Cui, Y., Wang, H., Cheng, X., & Chen, B. (2011). Wireless data center networking. *Wireless Communications, IEEE*, *18*(6), 46–53.

Dally, W., & Towles, B. (2004). *Principles and practices of interconnection networks*. Morgan Kaufmann.

Dean, J., & Ghemawat, S. (2008). Mapreduce: Simplified data processing on large clusters. *Communications of the ACM*, *51*(1), 107–113.

Deering, S. E. (1998). Internet protocol, version 6 (ipv6) specification.

Ding, Z., Guo, D., Liu, X., Luo, X., & Chen, G. (2012). A mapreduce-supported network structure for data centers. *Concurrency and Computation: Practice and Experience*, *24*(12), 1271–1295.

Droms, R. (1997). Dynamic host configuration protocol.

Farrington, N., Porter, G., Radhakrishnan, S., Bazzaz, H. H., Subramanya, V., Fainman, Y., et al. (2011). Helios: a hybrid electrical/optical switch architecture for modular data centers. *ACM SIGCOMM Computer Communication Review*, *41*(4), 339–350.

Formu, J. (2009). Cloud cube model: Selecting cloud formations for secure collaboration.

Foster, I., Kesselman, C., Nick, J. M., & Tuecke, S. (2002). Grid services for distributed system integration. *Computer*, *35*(6), 37–46.

Frécon, E., & Stenius, M. (1998). Dive: A scaleable network architecture for distributed virtual environments. *Distributed Systems Engineering*, *5*(3), 91.

Gantz, J., & Reinsel, D. (2012). The digital universe in 2020: Big data, bigger digital shadows, and biggest growth in the far east. *IDC iView: IDC Analyze the Future*.

Ghemawat, S., Gobioff, H., & Leung, S.-T. (2003). The google file system. In *Acm sigops operating systems review* (Vol. 37, pp. 29–43).

Greenberg, A., Hamilton, J., Jain, N., Kandula, S., Kim, C., Lahiri, P., et al. (2009). Vl2: a scalable and flexible data center network. *ACM SIGCOMM Computer Communication Review*, *39*(4), 51–62.

Greenberg, A., Hamilton, J., Maltz, D. A., & Patel, P. (2008). The cost of a cloud: research problems in data center networks. *ACM SIGCOMM Computer Communication Review*, *39*(1), 68–73.

Greenberg, A., Lahiri, P., Maltz, D. A., Patel, P., & Sengupta, S. (2008). Towards a next generation data center architecture: scalability and commoditization. In *Proceedings of the acm workshop on programmable routers for extensible services of tomorrow* (pp. 57–62).

Guo, C., Lu, G., Li, D., Wu, H., Zhang, X., Shi, Y., et al. (2009). Bcube: a high performance, server-centric network architecture for modular data centers. In *Acm sigcomm computer communication review* (Vol. 39, pp. 63–74).

Guo, C., Wu, H., Tan, K., Shi, L., Zhang, Y., & Lu, S. (2008). Dcell: a scalable and fault-tolerant network structure for data centers. In *Acm sigcomm computer communication review* (Vol. 38, pp. 75–86).

Heller, B., Seetharaman, S., Mahadevan, P., Yiakoumis, Y., Sharma, P., Banerjee, S., et al. (2010). Elastictree: Saving energy in data center networks. In *Nsdi* (Vol. 3, pp. 19–21).

Ikeda, T., & Tsutsumi, O. (1995). Optical switching and image storage by means of azobenzene liquid-crystal films. *SCIENCE-NEW YORK THEN WASHINGTON-*, 1873–1873.

Infrastructure, C. D. C. (2007). 2.5 design guide, cisco systems. *Inc, San Jose, CA*.

Isard, M., Budiu, M., Yu, Y., Birrell, A., & Fetterly, D. (2007). Dryad: distributed data-parallel programs from sequential building blocks. *ACM SIGOPS Operating Systems Review*, *41*(3), 59–72.

Kandula, S., Padhye, J., & Bahl, P. (2009). Flyways to de-congest data center networks.

Katayama, Y., Takano, K., Kohda, Y., Ohba, N., & Nakano, D. (2011). Wireless data center networking with steered-beam mmwave links. In *Wireless communications*

and networking conference (wcnc), 2011 ieee (pp. 2179–2184).

Khan, S., Shiraz, M., Abdul Wahab, A. W., Gani, A., Han, Q., & Bin Abdul Rahman, Z. (2014). A comprehensive review on adaptability of network forensics frameworks for mobile cloud computing. *The Scientific World Journal*, *2014*.

Knuth, D. (1977). A generalization of dijkstra's algorithm. *Information Processing Letters*, *6*(1), 1–5.

Kumar, A. A., Rao, S., Goswami, D., & Sahukari, G. (2012). Dcell-ip: Dcell emboldened with ip address hierarchy for efficient routing. In *Proceedings of international conference on advances in computing* (pp. 739–746).

Lee, Y. C., & Zomaya, A. Y. (2012). Energy efficient utilization of resources in cloud computing systems. *The Journal of Supercomputing*, *60*(2), 268–280.

Leighton, F. T. (1992). *Introduction to parallel algorithms and architectures*. Morgan Kaufmann San Francisco.

Li, D., Guo, C., Wu, H., Tan, K., Zhang, Y., & Lu, S. (2009). Ficonn: Using backup port for server interconnection in data centers. In *Infocom 2009, ieee* (pp. 2276–2285).

Li, W., & Svard, P. (2010). Rest-based soa application in the cloud: A text correction service case study. In *Services (services-1), 2010 6th world congress on* (pp. 84–90).

Lian, F.-L., Moyne, J., & Tilbury, D. (2002). Network design consideration for distributed control systems. *Control Systems Technology, IEEE Transactions on*, *10*(2), 297–307.

McCanne, S., Floyd, S., Fall, K., Varadhan, K., et al. (1997). *Network simulator ns-2*.

Model, C. C., Secure Collaboration, S. C. F. for, et al. (2009). Jericho forum.

Niranjan Mysore, R., Pamboris, A., Farrington, N., Huang, N., Miri, P., Radhakrishnan, S., et al. (2009). Portland: a scalable fault-tolerant layer 2 data center network fabric. In *Acm sigcomm computer communication review* (Vol. 39, pp. 39–50).

Pioro, M., Szentesi, A., Harmatos, J., Jüttner, A., Gajowniczek, P., & Kozdrowski, S. (2002). On open shortest path first related network optimisation problems. *Performance evaluation*, *48*(1), 201–223.

Popa, L., Ratnasamy, S., Iannaccone, G., Krishnamurthy, A., & Stoica, I. (2010). A cost comparison of datacenter network architectures. In *Proceedings of the 6th international conference* (p. 16).

Qi, H., & Gani, A. (2012). Research on mobile cloud computing: Review, trend and perspectives. In *Digital information and communication technology and it's applications (dictap), 2012 second international conference on* (pp. 195–202).

Qi, H., Shiraz, M., Gani, A., Whaiduzzaman, M., & Khan, S. (2014). Sierpinski triangle based data center architecture in cloud computing. *The Journal of Supercomputing*,

69(2), 887–907.

Qi, H., Shiraz, M., Liu, J.-y., Gani, A., Rahman, Z. A., & Altameem, T. A. (2014). Data center network architecture in cloud computing: review, taxonomy, and open research issues. *Journal of Zhejiang University SCIENCE C*, *15*(9), 776–793.

Ranachandran, K., et al. (2008). 60ghz data-center networking: wireless=> worryless. *NEC Laboratories America, Inc., Tech. Rep., July*.

Redkar, T., & Guidici, T. (2011). *Windows azure platform*. Apress.

Rimal, B. P., Choi, E., & Lumb, I. (2009). A taxonomy and survey of cloud computing systems. In *Inc, ims and idc, 2009. ncm'09. fifth international joint conference on* (pp. 44–51).

Sanaei, Z., Abolfazli, S., Gani, A., & Buyya, R. (2014). Heterogeneity in mobile cloud computing: taxonomy and open challenges. *Communications Surveys & Tutorials, IEEE*, *16*(1), 369–392.

Shankar, S. (2009). *Amazon elastic compute cloud.* CS.

Shin, J.-Y., Sirer, E. G., Weatherspoon, H., & Kirovski, D. (2012). On the feasibility of completely wireless datacenters. In *Proceedings of the eighth acm/ieee symposium on architectures for networking and communications systems* (pp. 3–14).

Shiraz, M., Ahmed, E., Gani, A., & Han, Q. (2014). Investigation on runtime partitioning of elastic mobile applications for mobile cloud computing. *The Journal of Supercomputing*, *67*(1), 84–103.

Sierpinski, W. (1916). Sur une courbe cantorienne qui contient une image biunivoque et continue de toute courbe donnée. *Comptes Rendus*, *629*.

Singh, A., Korupolu, M., & Mohapatra, D. (2008). Server-storage virtualization: integration and load balancing in data centers. In *Proceedings of the 2008 acm/ieee conference on supercomputing* (p. 53).

Singla, A., Hong, C.-Y., Popa, L., & Godfrey, P. B. (2012). Jellyfish: Networking data centers randomly. In *Proceedings of the 9th usenix conference on networked systems design and implementation* (pp. 17–17).

Tarantino, A. (2012). Point-of-view paper: High tech's innovative approach to sustainability. *International Journal of Innovation Science*, *4*(1), 37–40.

Tennenhouse, D. L., & Wetherall, D. J. (2002). Towards an active network architecture. In *Darpa active networks conference and exposition, 2002. proceedings* (pp. 2–15).

USEPA. (2012). *2012 annual report - us environmental protection agency* (Tech. Rep.). Available from `http://www2.epa.gov/sites/production/files/2013-11/documents/rad_12_annual_rep`

Vahdat, A., Al-Fares, M., Farrington, N., Mysore, R. N., Porter, G., & Radhakrishnan, S. (2010). Scale-out networking in the data center. *IEEE micro*, *30*(4), 29–41.

Valiant, L. G. (1990). A bridging model for parallel computation. *Communications of the ACM*, *33*(8), 103–111.

Wang, G., Andersen, D. G., Kaminsky, M., Papagiannaki, K., Ng, T., Kozuch, M., et al. (2010). c-through: Part-time optics in data centers. In *Acm sigcomm computer communication review* (Vol. 40, pp. 327–338).

Wang, T., Su, Z., Xia, Y., & Hamdi, M. (2014). Rethinking the data center networking: Architecture, network protocols, and resource sharing.

Wang, X., Fan, J., & Cheng, B. (2014). One-to-many disjoint path covers in dcell networks. *Computer Science and Systems Engineering*, *68*, 465.

Wang, X., Fan, J. X., Cheng, B. L., Liu, W. J., & Li, F. F. (2013). A hamilton path embedding algorithm on dcell. *Applied Mechanics and Materials*, *336*, 2468–2471.

Ward, B. (2002). *The book of vmware: the complete guide to vmware workstation*. No Starch Press.

Wu, H., Lu, G., Li, D., Guo, C., & Zhang, Y. (2009). Mdcube: a high performance network structure for modular data center interconnection. In *Proceedings of the 5th international conference on emerging networking experiments and technologies* (pp. 25–36).

Wu, K., Xiao, J., & Ni, L. M. (2012). Rethinking the architecture design of data center networks. *Frontiers of Computer Science*, *6*(5), 596–603.

Zahariev, A. (2009). Google app engine. *Helsinki University of Technology*.

LIST OF PUBLICATIONS AND PAPERS PRESENTED

Khan, S., Shiraz, M., Abdul Wahab, A. W., Gani, A., Han, Q., & Bin Abdul Rahman, Z. (2014). A comprehensive review on adaptability of network forensics frameworks for mobile cloud computing. *The Scientific World Journal*, *2014*.

Qi, H., & Gani, A. (2012). Research on mobile cloud computing: Review, trend and perspectives. In *Digital information and communication technology and it's applications (dictap), 2012 second international conference on* (pp. 195–202).

Qi, H., Shiraz, M., Gani, A., Whaiduzzaman, M., & Khan, S. (2014). Sierpinski triangle based data center architecture in cloud computing. *The Journal of Supercomputing*, *69*(2), 887–907.

Qi, H., Shiraz, M., Liu, J.-y., Gani, A., Rahman, Z. A., & Altameem, T. A. (2014). Data center network architecture in cloud computing: review, taxonomy, and open research issues. *Journal of Zhejiang University SCIENCE C*, *15*(9), 776–793.

Shiraz, M., Ahmed, E., Gani, A., & Han, Q. (2014). Investigation on runtime partitioning of elastic mobile applications for mobile cloud computing. *The Journal of Supercomputing*, *67*(1), 84–103.

APPENDIX A

PYTHON SOURCE CODE OF SAMPLE FILE GENERATION IN CHAPTER 5

```
___author___ = "QH"
___date___ = "$Date: 2013/04/09 $"


def generateRandom(rangeFrom, rangeTo):
    import random
    return random.randint(rangeFrom,rangeTo)
def generageMassiveIPAddr(fileLocation,numberOfLines):
    IP = []
    file_handler = open(fileLocation, 'a+')
    for i in range(numberOfLines):
        IP.append('192.168.' + str(generateRandom(0,255))
        +'.'+ str(generateRandom(0,255)) + '\n')
    file_handler.writelines(IP)
    file_handler.close()
if ___name___ == '___main___':
    from time import ctime
    print ctime()
    for i in range(10):
        print '  ' + str(i) + ": " + ctime()
        generageMassiveIPAddr('d:\\SampeIP.txt', 100000000)
    print ctime()
```

APPENDIX B

PYTHON SCRIPT FOR CALCULTING THE WORD COUNT

```
___author___ = "QH"
___date___ = "$Date: 2013/04/18 $"
import os
from time import ctime
def findIPAtOnce(targetFile):
    print "Started At: " + ctime()
    Result = {}
    file_handler = open(targetFile, 'r')
    for line in file_handler:
        if line in Result:
            Result[line] = Result[line] + 1
        else:
            Result[line] = 1
    print "Write to Dic Finished At: " + ctime()
    file_handler.close()
    Result = sorted(Result.items(), key=lambda d: d[1])
    print "Sorting Finished At: " + ctime()
    print 'Result:'
    for i in range(10):
        print '  ' + str(Result.pop())
if ___name___ == '___main___':
    findIPAtOnce("d:\\massiveIP.txt")
```

APPENDIX C

SOURCE CODE FOR DEPLOYING MAPREDUCE ON SERVERS

#1.core-site.xml

```
<configuration>
   <property>
        <name>fs.default.name</name>
        <value>hdfs://localhost:9000</value>
    </property>
    <property>
        <name>hadoop.tmp.dir</name>
        <value>/opt/hadoop-1.2.1/tmp</value>
    </property>
</configuration>
```

#2.hdfs-site.xml

```
<configuration>
  <property>
        <name>dfs.replication</name>
        <value>1</value>
    </property>
    <property>
        <name>dfs.name.dir</name>
```

```
        <value>/opt/hadoop-1.2.1/hdfs/name</value>
    </property>
    <property>
        <name>dfs.data.dir</name>
        <value>/opt/hadoop-1.2.1/hdfs/data</value>
    </property>
</configuration>

#3.mapred-site.xml:
<configuration>
  <property>
        <name>mapred.job.tracker</name>
        <value>localhost:9001</value>
    </property>
</configuration>
```

APPENDIX D

SOURCE CODE OF AWK SCRIPT FOR THROUGHPUT

```
BEGIN {
	recvdSize=0;
	startTime=0;
	stopTime=1000;
}
{
	event=$1;
	time=$2;
	node_id=$3;
	level=$4;
	flags=$5;
	seqno=$6;
	type=$7;
	pkt_size=$8;

	if(event=="s" && type == "cbr" && node_id == "_0_") {
		if( time < startTime ){
		startTime=time;
		}
	}
	if(event=="r" && type == "cbr" && node_id == "_2_") {
		if( time > stopTime ){
```

```
            stopTime=time;
            }
        }
        recvdSize+=pkt_size;

}
END{
        printf("Aggregated Throughput[Mbps] Reserved by QH=%.3f\t\t
        StartTime=%.3f\t StopTime=%.3f\n",(recvdSize/
        (stopTime-startTime))*(8/1000000),startTime,stopTime);

}
```