Software defined networking and virtual machine placement in data centers

Renuga Kanagavelu

2015

Renuga Kanagavelu. (2015). Software defined networking and virtual machine placement in data centers. Doctoral thesis, Nanyang Technological University, Singapore.

<https://hdl.handle.net/10356/62165>

<https://doi.org/10.32657/10356/62165>

Software Defined Networking and Virtual Machine Placement in Data Centers

Renuga Kanagavelu

School of Computer Engineering

A thesis submitted to the Nanyang Technological University
in partial fulfilment of the requirement for the degree of

Doctor of Philosophy

2015

Acknowledgements

I would like to place on records, my sincerest gratitude to my supervisor, Assoc. Prof. Lee Bu Sung, Francis, who has guided and supported me throughout my Ph.D. candidature with his knowledge and patience. I have not only acquired in-depth knowledge in my research area but also learnt to be a considerable and respectful person from him as my role model. I am also thankful for countless time and effort he spent on helping me improve my research skills. Without his help, this work would not have been possible.

Special thanks go to Dr. Khin Mi Mi Aung; she motivated me to do my research and has been supportive all the way here. Besides, my gratefulness goes to my colleagues in Data Storage Institute, A*STAR, Singapore, they have helped me in many ways in the preparation and completion of this study.

I am very grateful to my mother whose support over the last few years has been invaluable. Her everlasting support and endless love encouraged me to work hard. Special and heartfelt thanks goes to my loving and understanding husband Mohan for his long time support both with his knowledge and love without whom this would not have been possible. Finally my special love for my son Sanjit and my twin daughters Saanvi and Shivani. They form the backbone and origin of my confidence and happiness.

Contents

List of Figures	ix
List of Tables	xiii
Glossary	xv
1 Introduction	3
1.1 Challenges in Data Center Networks	5
1.1.1 Resource Utilization with Layer-2 Networks	5
1.1.2 Multi-path Routing	5
1.1.3 Load Balancing	6
1.1.4 Failure Handling	6
1.1.5 VM Placement and Traffic Routing	6
1.1.6 Software Defined Networks and Limitations	7
1.2 Problem and Objectives	8
1.3 Contributions	9
1.3.1 Adaptive Layer-2 Multi-path Forwarding	9
1.3.2 SDN-based Edge-to-Edge Rerouting	10
1.3.3 SDN-based Data Replication	10
1.3.4 SDN Flow Cache Design and Maintenance	11
1.3.5 VM Placement with Two-path Routing	11
1.3.6 Power-efficient VM Placement and Migration	12
1.4 Organization of the Report	12

2	Background and Literature Review	15
2.1	Data Center Topologies	17
2.1.1	Switch Centric Topologies	17
2.1.1.1	Fat Tree	17
2.1.1.2	Valiant Load Balancing (VL2)	19
2.1.1.3	Portland	19
2.1.2	Server Centric Topologies	20
2.1.2.1	Dcell	20
2.1.2.2	BCube	20
2.1.2.3	SecondNet	21
2.1.3	Discussion	22
2.2	Software Defined Networking	23
2.2.1	OpenFlow Routing Architecture	23
2.2.2	SDN Limitations	25
2.2.3	Discussion	25
2.3	Traffic Engineering for Data Centers	26
2.3.1	Discussion	28
2.4	Data Center Virtualization and VM Placement	29
2.4.1	VM Placement with Server Resource Constraints	31
2.4.2	Network-aware VM Placement	31
2.4.3	Power-aware VM Placement and Migration	33
2.4.4	Discussion	34
2.5	Summary	34
3	Adaptive Layer-2 Multi-path Forwarding	37
3.1	System Architecture	37
3.1.1	Addressing Scheme	38
3.1.2	Chief Supervisor	40
3.2	Proposed Load Balancing Framework	40
3.2.1	Route Computation	41
3.2.2	Load Measurement	42
3.2.3	Load-Aware Adaptive Routing Algorithm	43
3.2.4	Fault Tolerance	45

3.3	Performance Study	47
3.4	Summary	53
4	SDN-based Edge-to-Edge Rerouting	55
4.1	Routing Control Framework	56
4.1.1	Framework Modules	57
4.1.1.1	Monitor	57
4.1.1.2	Host Tracker	58
4.1.1.3	Routing Engine	58
4.1.1.4	Route Tracker	61
4.2	Fault Tolerance	61
4.3	Performance Evaluation	63
4.4	Simulation Results	66
4.4.1	Results for Fat Tree Topology	66
4.4.1.1	Load Balancing	67
4.4.2	Mesh Topology	67
4.4.2.1	Load Balancing	69
4.4.2.2	Fault Tolerance	69
4.5	Summary	70
5	SDN-based Data Replication	71
5.1	Proposed SDN-based Data Replication Framework	72
5.1.1	Dynamic Routing Method	72
5.1.1.1	Functional Modules	72
5.1.2	Network Under Test	76
5.2	Performance Study	77
5.2.1	Throughput	77
5.2.2	Performance for Different I/O Queue Size	78
5.2.3	Load Distribution	79
5.2.4	Throughput at Primary Server	80
5.3	Summary	80

6	Flow Cache Design and Maintenance	83
6.1	Caching Architecture and Maintenance	84
6.1.1	Elephant Flow Detection	87
6.1.2	Flow Cache and Dynamic-index Hashing	88
6.1.3	Overflow Handling- Bucket Expansion	90
6.1.4	Cache Replacement Strategy	90
6.1.5	Bucket Shrinking	91
6.2	Performance Evaluation	92
6.2.1	Performance in Terms of Cache Hit Rate	93
6.2.2	Impact of Cache Bucket Size	93
6.2.3	Performance in Terms of Lookup Time	94
6.2.4	Bucket Expansion Overhead	95
6.3	Summary	97
7	Virtual Machine Placement with Two path Routing	99
7.1	VM Placement and Routing Framework	100
7.1.1	Hypervisor	101
7.1.2	Controller	102
7.1.3	Implementation Issues:	103
7.2	Problem Definition and Formulation	103
7.3	Proposed GVMTPR Algorithm	108
7.3.1	VM Placement	108
7.3.2	Two Path Routing	110
7.3.3	Illustration of Routing and Path Load	112
7.3.4	Traffic Splitting Policy	112
7.4	Performance Evaluation	115
7.4.1	Simulation Setup	115
7.4.2	Bandwidth Cost	115
7.4.3	Impact of VM Size Distribution	116
7.4.4	Impact of Flow Sizes	117
7.4.5	Load Balancing	118
7.4.6	Demand Acceptance Ratio	119
7.4.7	Performance with Impact of Workload	119

7.4.8	Protection Grade	121
7.5	Summary	122
8	Power Efficient VM placement and Migration	123
8.1	Power Efficient Virtual Machine Placement and Migration Scheme . . .	124
8.2	Problem Formulation	126
8.3	Heuristic algorithms for VM placement and Migration	127
8.3.1	Virtual Machine Placement	127
8.3.2	Virtual Machine Migration	128
8.4	Simulation Experiments	129
8.5	Summary	134
9	Conclusion	135
9.1	Summary	135
9.2	Future Work	138
	List of Publications	141
	References	143

CONTENTS

List of Figures

1.1	Growth of Data Center traffic Source: Cisco Global Cloud Index based on projections for 2012-2017.(Fig. 1 of [1])	4
2.1	Conventional 3-tier Data Center topology (Fig. 1 of [2])	17
2.2	Illustration of Fat tree topology N=2	18
2.3	Illustration of DCell topology (Fig. 1 of [3]).	21
2.4	Illustration of BCube topology (Fig. 1 of [4]).	22
2.5	OpenFlow architecture	24
2.6	Illustration of VM placement	30
3.1	Three Tier Fat tree architecture.	38
3.2	HMAC addressing scheme	39
3.3	The load measurement protocol	44
3.4	Illustration of link failure	46
3.5	Simulation setup using Omnetpp simulator	48
3.6	Divider line dividing the network into two equal halves	49
3.7	Comparison of ECMP and our proposed LAAR algorithm in terms of bisection bandwidth [5]	49
3.8	Load distribution using LAAR [5]	50
3.9	Load distribution using ECMP [5]	50
3.10	Load distribution for stride 8 traffic pattern using LAAR and ECMP algorithms [5]	51
3.11	Link utilization using LAAR algorithm [5]	52
3.12	Load distribution without and with a link failure [5]	52
4.1	OpenFlow testbed	56

LIST OF FIGURES

4.2	OpenFlow-based re-routing control framework	57
4.3	Example of path cost comparison	60
4.4	Illustration of route selection when link F-E fails	62
4.5	Illustration of route selection when link B-E fails	63
4.6	Comparison of our proposed dynamic routing algorithm with ECMP and single path routing in terms of load distribution [6]	64
4.7	Comparison of our proposed dynamic routing algorithm with ECMP and single path routing in terms of link utilization [6]	65
4.8	Link utilization using the proposed approach before and after a link failure [6]	65
4.9	Simulation setup using Omnet++ simulator(fat tree topology)	67
4.10	Load distribution	68
4.11	Simulation setup using Omnet++ simulator(Mesh topology)	68
4.12	Mesh topology - load balancing across core switches	69
4.13	Mesh topology - traffic distribution upon a link failure	70
5.1	SDN based data replication architecture	73
5.2	SDN based data replication testbed	74
5.3	Shortest routes from replication host to primary server	75
5.4	Shortest routes from replication host to backup server	76
5.5	Performance of SDN vs. iSCSI(without SDN) based schemes [7]	79
5.6	Throughput for different I/O queue size [7]	79
5.7	Load Distribution across aggregate switches	80
5.8	Screenshot from the primary server during replication [7]	81
6.1	Cache architecture	84
6.2	Interactions between the controller, cache and flow table	86
6.3	Using VLAN PCB bits to classify flows	88
6.4	Illustration of placement of flow records	89
6.5	Illustration of bucket expansion	91
6.6	Illustration of bucket shrinking	92
6.7	Comparison of cache hit rate [8]	93
6.8	Impact of cache bucket size [8]	94
6.9	Lookup time for wild-card linear method [8]	95

6.10	Lookup time for mice flows [8]	96
6.11	Lookup time for elephant flows [8]	97
6.12	Bucket expansion overhead [8]	98
7.1	VM placement framework.	101
7.2	Intra-pod routing	113
7.3	Inter-pod routing.	113
7.4	Illustration of traffic splitting and protection grade. (a) Traffic Load across three paths. (b) 50:50 Traffic splitting. (c) 60:40 Traffic splitting. (d) 55:45 Traffic splitting	114
7.5	Performance evaluation based on bandwidth cost [9]	116
7.6	Performance comparison with varying VM sizes [9]	117
7.7	Performance under varying flow sizes [9]	118
7.8	Traffic load across core switches [9]	119
7.9	Demand acceptance ratio [9]	120
7.10	Maximum core link utilization for varying workloads [9]	120
7.11	Protection grade [9]	121
8.1	Power aware VM placement and migration illustration.	125
8.2	Active number of hypervisors for N=4 fat tree.	131
8.3	Active number of hypervisors for N=8 fat tree.	131
8.4	Inner pod flow percentage for N=4 fat tree [10]	133
8.5	Inner pod flow percentage for N=8 fat tree [10]	134

LIST OF FIGURES

List of Tables

2.1	DCN topology categories	18
3.1	Multiple paths between host E and host H	41
3.2	Multiple paths between host C and host P	41
3.3	Path load information table at switch E_2	43
3.4	Fraction of the traffic transmitted along the paths based on the load . .	44
3.5	Fraction of the traffic distributed along the paths after failure	47
4.1	Host tracker entries	58
4.2	Cost information	59
4.3	Route tracker records	61
5.1	Switch configuration	78
5.2	Replication host, Primary server, Backup server, and NOX configuration	78
6.1	Bucket expansion overhead	96
8.1	Parameter setting in Experiment I and Experiment II	130
8.2	Job Delay in Experiment I and Experiment II	133

Glossary

AA	Application-specific IP address, page 19
ARP	Address Resolution Protocol Protocol, page 20
AS	Aggregation switches, page 15
CN	Control Network , page 72
CS	Chief Supervisor, page 40
DAC	Data Center Address Configuration, page 19
DCN	Data Center Network , page 3
ECMP	Equal-Cost Multipath , page 1
FIFO	First In First Out , page 93
GVMTPR	Greedy VM placement with Two Path Routing , page 11
HMAC	Hierarchical MAC Addresses, page 39
HTTP	Hyper Text Transfer Protocol , page 25
IBM	International Business Machines , page 71
IP	Internet Protocol , page 5
iSCSI	Internet Small Computer System Interface , page 72
LA	Location-specific IP addresses, page 19
LAAR	Load-Aware Adaptive Routing, page 43

GLOSSARY

LDP	Location Discovery Protocol, page 19
LLRU	Localized Least Recently Used , page 11
MCRVMP	Min Cut Ratio-aware VM Placement, page 32
MDC	Modular Data Center, page 20
NIC	Network Interface Card, page 15
NOX	Network Operating System , page 74
NUT	Network Under Test , page 72
OSPF	Open Shortest Path First, page 5
PCB	Priority Code Bit , page 87
PM	Physical Machine, page 29
RSTP	Rapid Spanning Tree protocol , page 5
SDN	Software Defined Networking , page 1
SHA	Secure Hash algorithm, page 89
SPAIN	Smart Path Assignment in Networks, page 27
SPB	Shortest Path Brididging, page 37
STP	Spanning Tree Protocol, page 1
TCAM	Ternary Content Addressable Memory, page 2
TE	Traffic Engineering , page 4
ToR	Top of Rack, page 15
TRILL	Transparent Interconnect of Lots of Links, page 27
VDC	Virtual Data Center, page 22
VL2	Valiant Load Balancing, page 19

VLAN Virtual local area network , page 25

VM Virtual Machine, page 1

Abstract

Increasing popularity of cloud computing and server virtualization has led to large Data Centers with tens of thousands of servers interconnected by a network. In a Data Center, server resources are shared by a number of Virtual Machines (VMs) and network bandwidth is shared by the traffic generated by the VMs. Recently, there has been a prominent shift from the traditional north-south traffic pattern (between servers in the Data Center and external clients) to east-west traffic pattern (between servers within the Data Center). Such traffic patterns are characterized by a mix of large number of short(mice) flows and small number of large (elephant) flows. The servers in a Data Center are connected by a Data Center network which generally uses a 3-tier multi-rooted tree topology. A key limitation of current Data Center networks is the absence of effective traffic engineering mechanism for routing traffic between servers. This results in overloading of certain links leading to congestion affecting the performance of applications. The conventional Spanning Tree Protocol (STP) does not utilize multipath diversity leading to congestion and poor network utilization. The existing Equal-Cost Multipath routing (ECMP), which is a hash-based routing method can make use of the available multiple paths. But, it does not consider the dynamic state of the network link utilization which could lead to a number of large elephant flows mapped to the same link causing congestion. Further, the capability of recently-emerged software-defined networking (SDN), VM placement, and VM migration techniques are not effectively used for dealing with congestion and failures.

To address the above limitations, in this thesis, we investigated the problem of traffic engineering in Data Centers and studied several related issues focusing on congestion, fault tolerance, power consumption, and caching. Our

solution approaches leverage on the unique features of SDN, VM placement, and VM migration. First, we developed an adaptive layer-2 load balancing framework utilizing multi-path diversity. This mechanism adaptively routes traffic based on path load conditions that enhances forwarding in Layer-2 Ethernet switches. It distributes traffic adaptively on multiple paths to reduce congestion and also ensures effective traffic redistribution upon link failures. Leveraging on the the capability of SDN for monitoring the state of the links, we developed a flow-based adaptive rerouting scheme which considers the dynamic network state to provide high bisection bandwidth for throughput-sensitive flows. Our proposed scheme focuses on rerouting large flows to alternative paths in the event of link congestion or failure. We then extended the SDN-based framework for addressing the routing problem for the data replication application which is an effective approach for data reliability in Data Centers.

Although, SDN brings in several benefits, it is still a challenge for SDN to provide a fair treatment for elephant flows in a Data Center due to the use of Ternary Content Addressable Memory (TCAM) -based flow tables which are limited in size for the reasons of cost and energy efficiency. To address the problem of frequent eviction of elephant flows, we proposed a differential flow cache framework to provide a fair treatment to elephant flows in the presence of large number of mice flows.

We then addressed the problem of traffic engineering considering VM placement, a unique feature in Data Centers. We studied the problem of reducing congestion while providing a certain protection grade upon a link failure. We developed a greedy VM placement algorithm taking into consideration the impact on network load, and route traffic over two link-disjoint paths to solve the above problem. Leveraging on the feature of VM migration in Data Centers, we proposed a method for saving power using intelligent VM placement and migration in Data Centers.

We carried out performance study through simulation experiments and also prototype implementations. The results demonstrate that our methods are effective in achieving the objectives.

Chapter 1

Introduction

Data Centers are experiencing continuous and rapid growth in the network traffic due to the applications such as video streaming, big data processing and social networking. This leads to the creation of large data centers with thousands of switches interconnecting tens of thousands of servers that are hosting hundreds of thousands of Virtual Machines (VMs) [11, 12, 13, 14, 15, 16]. According to Cisco Global Index [1], it is forecast that the global traffic will reach about 7.7 Zettabytes per year in 2017, 76% of traffic is within the Data Center, and this traffic will continue to rise in the coming years.

Traditional Data Centers [2, 17] use 3-tier tree (with edge, aggregate and core layers) topology mainly designed for client-server type of communications. The servers inside a Data Center can communicate with the clients outside a Data Center. This generates *north-south* traffic which enters and exits the Data Center [18]. It has been observed that traffic patterns tend to be more of *east-west*, which is generated between servers within a Data Center [18]. The traditional Data Center network (DCN) is good for handling traffic in and out of the Data Center (north-south traffic) and is not effective to handle high volume of server-to-server traffic (east-west traffic) within the Data Center [18]. As a result, server-to-server traffic is likely to traverse multiple layers of switches, and each switch adds latency resulting in poor response time [2, 17]. With the increased use of virtualization [19], most cloud computing services and distributed platforms generate large flows of data traffic and distribute them across possibly hundreds or even thousands of servers inside a Data Center. Since the network link bandwidth is shared among different traffic flows, there is a possibility of congestion creating

1. INTRODUCTION

Figure 1.1: Growth of Data Center traffic Source: Cisco Global Cloud Index based on projections for 2012-2017.(Fig. 1 of [1])

bottleneck links [20]. Similar problem occurs in the event of component failures. This drives the need for effective traffic engineering (TE) mechanisms for improved use of network resources.

Traffic engineering is a method of mapping traffic onto network resources with the objectives of improved resource utilization while satisfying application's performance needs. The key objective of a TE method is to balance the traffic demands to reduce the possibility of congestion and accommodate more traffic demands in future without adding extra network resources in the DCN [21].

The *east-west* traffic is characterized by a mix of short (mice) and long (elephant) flows. Elephant flows which are generally throughput-sensitive are generated by applications like MapReduce, VM migrations that require large amounts of data transfer on the order of hundreds of kilo, mega, or gigabytes [22]. On the other hand, mice flows are generally latency-sensitive applications which transfer up to a few kilobytes of data. Recent studies on Data Center traffic [20] show that applications generate a mix of traffic flows- elephant flows and mice flows, with elephant flows accounting for less than 10% of the flows but carrying more than 90% of the traffic. On the other hand,

there are a large number of short mice flows. The challenge in handling this traffic mix requires a Data Center to assign relative priorities and route them appropriately in such a way to provide high bi-section bandwidth for elephant flows and short delay for mice flows.

1.1 Challenges in Data Center Networks

This section briefly reviews related literature and present the following key issues in current Data Center networks.

1.1.1 Resource Utilization with Layer-2 Networks

Traditional Layer-2 networks use Spanning Tree Protocol (STP) [23] for frame forwarding, which reduces the topology into a tree by blocking some ports and links. STP guarantees a single loop-free path between any two switches. However, it has some drawbacks. It does not use all the links resulting in poor bandwidth utilization. It does not support multipathing in Layer-2 (L2) networks even though there exist multiple paths between switches. It adds lots of load on core switches and converges very slowly [23]. Though Rapid Spanning Tree Protocol (RSTP) [24] which is the extension of STP, speeds up convergence time following a link failure, it still uses a single tree to deliver traffic between communication pairs. These limitations become critical in modern Data Centers which generate large amount of traffic.

1.1.2 Multi-path Routing

Multipathing in Layer-3 networks can be achieved by Equal-Cost Multi-Path (ECMP)[25] routing which routes traffic on to paths of equal cost based on hashing at every hop. The traditional Open Shortest Path First (OSPF) protocol [26] used in Internet Protocol (IP) networks allows ECMP routing. However, ECMP suffers from major drawbacks when used in Data Center networks [22, 27, 28]. ECMP algorithm which routes traffic on to multiple paths based on hashing at every hop, is effective to some extent for bursty short (mice) flows. But ECMP is not effective for large (elephant) flows because they are few in numbers and mapping of large flows (due to hash collision) onto the same link when other links are relatively free would result in poor usage of network link bandwidth and congestion [29]. This drives the need for effective traffic engineering

1. INTRODUCTION

with adaptive routing in Data Center networks to provide high bisection bandwidth. In other words, in the absence of intelligent traffic engineering for routing these flows, there is a potential danger of congesting a portion of the network creating hotspot links.

1.1.3 Load Balancing

DCNs today suffer from congestion problem [30] as the traffic load changes frequently. Based on the traffic collected from 19 DCNs in [31], it is observed that the links in the core layer are more heavily utilized than those in the edge layer. An optimal DCN should be able to leverage on the high-capacity network interconnects by spreading traffic across the redundant links in the multi-rooted topologies for achieving high bisection bandwidth [32]. Designing effective routing strategies for DCNs depend on the traffic patterns and topology.

1.1.4 Failure Handling

Though there exist many paths between a pair of hosts in a DCN, a link failure leads to loss of traffic destined to a set of hosts temporarily until the failure is repaired or traffic is rerouted. For example, in a 3-level, 64-port fat tree [33], 32 hosts are affected by a link failure at the bottom level and 1024 hosts (15% of the topology's hosts) are affected by a failure at the top level [34]. From some recent studies, it is observed that DCN link failures could result in the loss of critical packets such as acknowledgement and keep-alive packets [35]. An efficient and fast network failure recovery mechanism for a DCN considering network traffic state to avoid possible congestion, is missing in the current DCN technologies [36, 37, 38].

1.1.5 VM Placement and Traffic Routing

In a Data Center, a virtual machine (VM) can be mapped to a server chosen from a list of candidate physical servers. VM placement in a Data Center is important as it has direct impact on network traffic. Virtualization is being deployed in Data Centers at a rapid pace to consolidate workloads for improved server utilization, for ease of provisioning, configuration management, and more generally, for efficient and flexible use of Data Center resources [39, 40, 41]. With the proliferation of east-west traffic across servers, cloud providers and large enterprises began to recognize the need for

virtualizing network bandwidth resources (in addition to the computing and storage resources). The performance of applications or jobs inside a VM is affected by other VMs placed in the same physical server and also by other communicating VMs that share Data Center network bandwidth [42, 43, 44, 45].

It is highly desirable that VMs be placed to maximize application performance, power efficiency, fault tolerance, and reduce network bandwidth usage. VM placement on servers has significant impact on network traffic [46]. An improper VM placement can result in the network traffic traversing more links and also cause congestion on links [47]. This leads to a research question as to how we can place VMs on to servers in a way to have minimal impact on network link loads. Further, conventional networks are not optimized for virtual machine placement and migrations in Data Centers [30, 48, 49].

1.1.6 Software Defined Networks and Limitations

Software-defined networking (SDN) enables network virtualization by de-coupling network control functionality from forwarding functionality. With SDN, traffic engineering can be done more effectively by using intelligent mechanisms for routing or rerouting traffic flows. OpenFlow [50] is an emerging technology for SDN implementation. OpenFlow separates the control path and data path, with all the networking logic and policies handled by a separate controller [51].

OpenFlow defines a framework to perform per-flow routing where switches maintain flow tables. Every OpenFlow switch maintains its own table of flow entries (flow table). An entry specifies a set of packet fields and the corresponding action to perform when there is a match. The action includes forward, drop, or modify header. Flow tables are built out of Ternary Content Addressable Memory (TCAM) that support wild-card entries and parallel lookups [50]. For example, the OpenFlow enabled HP 5406zl switch hardware [52] can support up to 1500 OpenFlow rule entries (OpenFlow rule is described by 10 header fields with 288 bits [50]), whereas 64000 forwarding entries can be supported for standard Ethernet switches. This is too small a number compared to the number of active flows arriving at the switch. It is possible to increase TCAM entries, but it consumes lots of Application-Specific Integrated Circuit (ASIC) space [53], and power (about 15 Watt/ Mbit)[54] and also has high cost(US\$350 for a 1M-bit chip). A major issue in a OpenFlow switch is the limited size of the flow table resulting in eviction of flows from the flow table. The limitation of flow table size gives unfair

1. INTRODUCTION

treatment to elephant flows, that are more likely to be evicted when compared to mice flows, causing additional traffic to the controller. To overcome this, there is a need for new cache architecture and set of algorithms designed to efficiently manage the cache in software-defined networks.

1.2 Problem and Objectives

Current Data Center networks face several challenges as described in section 1.1. This thesis mainly focuses on the design and development of dynamic traffic engineering techniques for Data Center networks with effective traffic routing based on SDN control that addresses the issues of load balancing and fault tolerance to avoid the occurrence of congestion. Considering the impact of VM placement on network traffic load, this thesis also studies the problem of VM placement and traffic routing from the perspective of traffic protection and energy efficiency in addition to the congestion problem. Though SDN provides network programmability, it has scalability limitation due to the small switch size and large header fields. SDN switches can hold up to thousands of entries which is too small for the incoming flows leading to the flow table explosion and unfair eviction of elephant flows. This fairness problem can be resolved by efficient cache management proposed in the thesis.

This thesis addresses the traffic engineering problem considering the features and capabilities in Data Centers such as east-west traffic pattern, SDN, VM placement, VM migration and switch constraints. It aims to answer the following research questions.

- How can we achieve performance benefits using layer-2 multipathing considering the dynamic network state using computationally simple methods?
- How can we make use of SDN capabilities to realize adaptive load balancing?
- How can we dynamically reallocate flows in the event of failures to reduce traffic congestion?
- How can we resolve the fairness problem arising from unfair eviction of elephant flows from flow tables in SDN switches?
- What are the performance benefits of adding network knowledge while doing VM placement?

- How can we achieve performance benefits in terms of congestion and also traffic protection considering network knowledge with VM placement and traffic distribution?
- How can we achieve performance benefits in terms of energy efficiency considering network knowledge with VM placement?

1.3 Contributions

In this section, we first enumerate the goals and explain the proposed approaches.

1. To develop an efficient algorithm for adaptive layer-2 multi-path forwarding by adaptively distributing traffic over different available paths in a balanced way to reduce congestion by minimizing the load on hotspot links.
2. To develop efficient SDN-based routing/rerouting algorithms for routing/rerouting of flows adapting to the load conditions and in the event of network failures.
3. To develop a SDN-based flow routing mechanism and prototype for data replication in Data Centers and investigate its performance benefits.
4. To design a flow cache which would be fair with elephant flows in terms of evictions when compared to mice flows in SDN switches.
5. To develop an efficient VMs placement scheme to reduce congestion with the consideration of resource requirements of VMs including communication bandwidth and traffic protection grade.
6. To develop an efficient mechanism with VM placement and migration for improving energy efficiency.

The contributions are explained in more detail in the following sections.

1.3.1 Adaptive Layer-2 Multi-path Forwarding

A load-aware adaptive routing scheme has been proposed with the objective of increasing bandwidth utilization. This method distributes the traffic over different available

1. INTRODUCTION

multiple paths based on the dynamic network state in a balanced way to reduce congestion by minimizing the load on hotspot links. Although some of the previous solutions such as TRILL [55], SEATTLE [56], and SPAIN [57] can leverage multipath diversity on an arbitrary topology, they only use the ECMP or even single path to deliver a traffic flow [57] and did not address the load balancing in a real time manner [57]. The effectiveness of our approach and algorithm was demonstrated by simulation results. In our simulation, it was shown that our LAAR algorithm reduces the load of core switches up to 25% when compared to the well known ECMP.

1.3.2 SDN-based Edge-to-Edge Rerouting

To ease the congestion, a different approach, SDN-based adaptive routing scheme has been proposed which dynamically reallocates different flows in Data Center networks in the event of congestion and faults. A prototype using OpenFlow-enabled switches has been implemented to demonstrate the effectiveness of the proposed scheme. The proposed SDN-based TE work differs from other previous proposals Hedera [27], MicroTE [58] by considering the current network state for path selection and flow forwarding so that it can dynamically adapt to varying traffic conditions which is suitable for Data Centers. Previous proposals Hedera [27], MicroTE [58] performed flow level TE based on predicted traffic demands without considering the current network state. Estimation of traffic demands of flows between all pairs of hosts and routing/rerouting flows incur large overhead in terms of time complexity and number of messages. Also, inaccurate traffic prediction leads to poor performance. With our prototype, it was shown that our SDN-based edge-to-edge rerouting technique reduces the stress of core switches up to 40% when compared to the well known ECMP.

1.3.3 SDN-based Data Replication

The SDN-based adaptive routing scheme has been extended to support the data replication application in Data Centers. Previous proposals [59, 60, 61, 62, 63] performs remote mirroring operation at a geographically distributed location. The proposed scheme addresses this problem within the Data Center and differs from the existing techniques, by providing more than one route each to a different replicated copy with the objective of load balancing. We carried out the design and implementation of OpenFlow controller service that automatically determines the least loaded path for a flow to

the location of the primary destination and the replica server. In our prototype, it was shown that our SDN-based replication technique improves the performance up to 20% and reduces the stress of core switches up to 25% when compared to the traditional iSCSI method.

1.3.4 SDN Flow Cache Design and Maintenance

Though SDN provides network programmability, it has some limitations. One of the major issues in OpenFlow switches is the limited size of the flow table resulting in evictions of flows from the flow table. Large elephant flows are more likely to be evicted due to the flooding of short mice flows. This problem has been addressed by the proposed differential flow cache SDN framework with hash-based placement and Localized Least Recently Used (LLRU)-based replacement mechanisms. The framework achieves fairness and efficient cache maintenance with fast lookup and reduced cache miss ratio. Previous works on mice flooding problem were done in the context of Internet [64, 65]. Another solution DIFANE [66] reduces the load of the controller by exploiting a special type of switch, called authority switch that is responsible for caching some portion of the flow table. But, it requires switches to have sufficient CPU resources to realize the extra control plane functionalities. The effectiveness of the proposal was demonstrated through simulation results. It was shown that our proposal reduces the lookup time up to 50% when compared to the wild-card linear method.

1.3.5 VM Placement with Two-path Routing

Considering the impact of VM placement on network traffic load, this thesis considered the problem of VM placement and traffic routing from the perspective of traffic protection in addition to the congestion problem. The proposed intelligent VM placement algorithm, called, Greedy VM placement with Two Path Routing (GVMTPR), which takes the communication requirements between VMs as well as the network traffic state into account during the deployment of the VMs. Previous works have investigated how to optimally place VMs [46, 67], or select routes by exploiting multipath diversity in the network [27, 58], or jointly leveraging both VM placement and routing [42]. In contrast to previous works, the emphasis of our investigation is to fill the gap for a VM placement and traffic routing scheme that is able to do traffic routing traffic so that traffic can be protected against failures in addition to reducing congestion. The

1. INTRODUCTION

effectiveness of the proposed approach and algorithm has been demonstrated through simulation results. It was shown that our GVMTPR improves the bandwidth cost up to 50% and reduces the stress of core switches up to 40% when compared to baseline methods.

1.3.6 Power-efficient VM Placement and Migration

The VM placement scheme has been extended to power-aware VM placement and migration to ensure power efficiency in Data Centers. Previous works have investigated how to optimally place VMs [44, 68] to save energy, or powering-off network components that are not required for the traffic demands at a given time [69], or jointly leveraging on both VM placement and routing [70] to save energy. The proposed power-aware migration attempts to save energy by considering server and network components and VM migration. It aggregates virtual machines to a few physical machines to reduce server power consumption and also communicating virtual machines would be migrated into close locations to save power consumption. It achieves up to 50% reduction in the number of active physical machines, up to 26% reduction in job delay as well as up to 50% flows migrating to the same pods, when compared to baseline methods.

1.4 Organization of the Report

This thesis is organised into nine chapters. Overall, the problem of traffic engineering in Data Centers to address the issues of load balancing, fault tolerance, VM placement, fairness and power consumption was investigated.

The organization of the rest of the thesis is as follows:

Chapter 2 provides a comprehensive literature survey of various Data Center network topologies and explaining their features, pros and cons. It describes software-defined networking concept and OpenFlow protocol. It reviews current traffic engineering techniques which could be used in Data Center networks and present their shortcomings. Finally, it reviews VM placement techniques from the perspective of network-awareness and power-awareness.

Chapter 3 describes layer-2 load balancing framework with a multi-path forwarding mechanism which balances the load across the network adaptively, thereby reducing the

occurrence of congestion that leads to efficient utilization of the bi-section bandwidth in Data Centers.

Chapter 4 describes flow-based dynamic edge-to-edge re-routing with SDN prototype implementation which is designed for easing congestions by delivering high bisection bandwidth for throughput-sensitive flows. It explains the OpenFlow-based software-defined network technique used for prototyping. It presents and discusses the results obtained from prototype experimentation.

Chapter 5 describes a load balancing mechanism for SDN-based Data replication in Data Center networks. It presents the method and discusses the results obtained from the prototype implementation.

Chapter 6 presents a flow cache mechanism developed to address a major issue in OpenFlow switch, which is the limited size of the flow table resulting in eviction of large flows from the flow table. It describes a differential flow cache framework that achieves fairness and efficient cache maintenance with fast lookup and reduced cache miss ratio.

Chapter 7, reports on the problem of VM placement with traffic routing on two paths to reduce occurrence of congestion while satisfying a certain protection grade which is defined as the fraction of rate (or bandwidth) guaranteed to be available in the event of single link failures. It presents the problem formulation. It then describes the proposed algorithm followed by performance study.

Chapter 8, presents a power efficient solution for virtual machine placement and migration in Data Center networks. It studies the benefits of considering the network knowledge in VM placement and migration in terms of energy efficiency.

Finally, Chapter 9 concludes the thesis summarizing our contributions and observations. It then presents possible directions for future work.

1. INTRODUCTION

Chapter 2

Background and Literature Review

This Chapter describes the conventional three-tier Data Center network (DCN) architecture and several state-of-the-art DCN topologies. We survey the various milestone research work related to Data Center traffic engineering techniques, VM placement strategies, classify them into different categories to facilitate better understanding and discuss their merits and limitations. In addition, we introduce Software Defined Networking (SDN) and OpenFlow protocol, a *de facto* standard for realizing core concept of SDN.

A Data Center houses a number of servers (could be on the order of tens of thousands), storage, network devices and cooling systems to host applications and store data. Computing systems are packed into racks and are connected by a high-bandwidth network. A DCN is a high speed communication network which has a topology and protocols (e.g., Ethernet, IP) associated with it. The conventional Data Center network has 3-tier hierarchical topology, which aggregates bandwidth from bottom up. Figure 2.1 shows a conventional 3-tier hierarchical Data Center Network architecture[71]. In a rack, a few tens of servers are connected to a Top of Rack (ToR) switch through network interface cards (NICs). ToR switches are connected to the aggregation switches (AS) which carry the server-to-server traffic between racks. The core switches provide connectivity between aggregation switches. The core switches also provide connectivity to the Internet outside the Data Center. As the data traffic grows, Data Centers experiencing exponential growth in servers demand for high end-to-end aggregate bandwidth.

2. BACKGROUND AND LITERATURE REVIEW

It is to be noted that even the use of high-end network element at higher layers could only provide 50% of end-to-end total bandwidth [49]. Thus, researchers have proposed DCN architectures, that provide up to full bisection bandwidth and can scale to hundreds of thousands of servers [27, 48, 49, 72]. Some of the limitations of DCNs are as follows.

- *Oversubscription* - In the conventional Data Center network, the servers in different layer-2 domains must go through the layer-3 switches. Aggregate switches are typically oversubscribed by the factor of 20:1 and core layers are oversubscribed by 240:1 [48]. As the oversubscription ratio is higher, more resources are shared leading to the limited bandwidth availability between servers in different parts of the Data Center [48, 49]. Care must be taken while choosing the servers for the applications to ensure the traffic does not saturate network links.
- *Limited Multi-path Diversity* - Equal-cost Multi-path (ECMP) is a popular technique considered in Data Center networks for effectively using multiple paths. This technique directs flows based on the hash value of certain fields of the flow, as all packets pertaining to a flow will follow the same path making certain parts of the network heavily loaded while leaving the other parts lightly loaded. This technique neither considers the flow size nor the network utilization [25]. To better balance the load in the network the packets/flows must be routed based on the current network utilization and the flow size. Thus, ECMP does not consider dynamic nature of Data Center traffic characteristics leading to congestion on some links.
- *scalability* - Over the years, the scale-up architecture becomes the de facto standard in Data centers. It works simply by replacing an existing server/router with more powerful one. When the workload becomes too large in a Data Center, the existing servers and switches are replaced by a new ones having more capacity. This is expensive and not scalable [69]. The problem associated with the scale-up architecture is the difficulty of maintenance. It requires periodic movement of applications and data from one machine to another and also cannot leverage on the economy of scale.

Figure 2.1: Conventional 3-tier Data Center topology (Fig. 1 of [2])

2.1 Data Center Topologies

DCN architectures can use either of the following two basic topologies [4][32]: (a) switch-centric topology [49], (b) server-centric topology [4]. While the switches participate in routing and forwarding in the case of switch-centric topology, in the case of server-centric topology servers with network interfaces also participate in the routing and forwarding of packets. DCN architectures can be classified as shown in Table 2.1. Fat-Tree [49], VL2 [48], and PortLand [72] follow the switch-centric approach and DCell [3] and BCube [4] follow the server-centric approach.

2.1.1 Switch Centric Topologies

2.1.1.1 Fat Tree

Clos network topology [73] has been used half a century ago for telephone switches, aiming to deliver high network throughput with commodity switches. Recently, in [49], Al-Fares *et al.* adopt a standard Clos topology called fat tree [33], which provides non-

2. BACKGROUND AND LITERATURE REVIEW

Table 2.1: DCN topology categories

DCN Topologies	Example
Switch Centric Topologies	Fat Tree
	VL2
	Portland
Server Centric Topologies	Dcell
	Bcube
	SecondNet

Figure 2.2: Illustration of Fat tree topology $N=2$

blocking network connection with 1:1 oversubscription ratio. This is a switch-centric network topology. More details about the fat tree as a Data Center architecture can be found in [49]. It provides extensive path diversity using smaller commodity switches.

We summarize the important terms here. An N -ary fat tree as shown in Figure 2.2, is built from $2N$ -port switches, where N is a positive integer. A fat tree topology contains three layers of switches, namely edge layer, aggregation layer and core layer from bottom to top as shown in Figure 2.2. The lower two layers are separated into

2N pods, each containing two layers of N switches, with lower layer edge switches and upper layer aggregation switches. In the uppermost layer, there are N^2 core switches; each core switch has a connection towards each of the 2N pods. Since each edge switch connects to N end stations and each pod has N edge switches, there are N^2 end stations in a pod and $2N^3$ end stations in a fat tree topology with $N = 2$.

2.1.1.2 Valiant Load Balancing (VL2)

VL2 has been developed to achieve uniform traffic distribution and avoid traffic hotspots by scaling out the network. VL2 is a three-tier architecture with ToR (Top of the Rack) edge, aggregation, and core layers. VL2 [48] uses commodity switches to build a layer 2 domain and develops a randomization-based load balancing mechanism. The switches are connected as a Clos [73] topology which provides multiple paths between switches with no oversubscription. VL2 scales horizontally by providing more interconnects between aggregate switches. For address resolution, VL2 uses location-specific IP addresses (LA) which are hierarchical within the network which help to reduce overhead due to mobility. It also uses application-specific IP address (AA) for applications which remain the same in the event of application migration.

The valiant load balancing scheme [74] used in VL2 is similar to ECMP. It distributes the flows uniformly across the paths by randomly choosing the path without centralized coordination [48]. However, for Data Center unpredictable traffic, the random path selection still lead to link congestion. Apart from that, it does not guarantee the absolute bandwidth between servers [75].

2.1.1.3 Portland

PortLand [72] uses multi-rooted fat-tree topology with scalable, fault-tolerant layer-2 routing and forwarding. It addresses the the problems like efficient VM migration, VM population scalability. It performs static routing using location-specific addresses. The PMAC (physical MAC) address assignment is done by distributed location discovery protocol (LDP). It uses two-level routing tables to route the traffic. The main difference between existing DAC (Data Center Address configuration) and Portland's LDP is, the Portland's LDP is designed for multi-rooted trees and cannot be applied to arbitrary topologies. PortLand uses OpenFlow (an enabler of software defined networks to be

2. BACKGROUND AND LITERATURE REVIEW

explained later in this chapter), and forwards packets at the TCP flow granularity. Portland can automatically detect a failure and recover from it.

There are several drawbacks associated with this architecture. Since, it is designed for multi-rooted fat-tree topology [72], it is not suitable for other DCN topologies. The Address Resolution Protocol (ARP) resolution by the fabric manager [72], make it easy to face failures by malicious attacks [75]. This architecture requires large connectivity requirements between the edge switches and servers [72].

2.1.2 Server Centric Topologies

2.1.2.1 Dcell

Modular Data Center design [76, 77], was proposed to construct Data Centers with modules instead of server racks. Each module can hold up to a few thousands of servers on multiple racks. DCell and BCube are the examples of modular Data Centers.

A recursively defined DCN architecture, referred to as DCell model as shown in Figure 2.3, has been developed by Guo *et al.* [3]. DCell which mainly addresses the network scalability issue, in which a higher level DCell is constructed from lower level DCells. The basic building block $DCell_0$ consists of n equally-acted servers and a mini-switch. This proposal exhibits desired characteristics such as double exponential scalability, fault-tolerance and higher network capacity. But, it needs rich physical connectivity between servers. DCell is fault tolerant since it does not have single point of failure and its distributed fault-tolerant routing protocol performs near shortest-path routing in the presence of link or node failures. Though this architecture is highly scalable and provides one-to-one and all-to-all communications, it suffers from high oversubscription and low cross section bandwidth [43].

2.1.2.2 BCube

As stated earlier, modular Data Center (MDC) has become a new approach to building Data Center [76, 78]. The BCube architecture serves as a suitable fundamental building block to realize a MDC design, which requires special features such as higher network capacity and less performance degradation in the event of hardware failures. The BCube architecture is a recursive structure in nature[4]. For example, a $BCube_1$ is constructed from n $BCube_0$ and n n -port switches. More generally, a $BCube_k$ ($k \geq 1$)

2. BACKGROUND AND LITERATURE REVIEW

Figure 2.4: Illustration of BCube topology (Fig. 1 of [4]).

network. A tenant can request a virtual data center (VDC) with the specified server resources and network bandwidth resources. It uses port-switching based source routing (PSSR) to achieve scalability in arbitrary topologies. Since it mainly depends on the physical topology of the network, it limits its performance.

2.1.3 Discussion

A number of different topologies have been proposed in the literature as aforementioned [49, 56, 71, 76] to meet the requirements for high performance Data Center networks with the goal of high throughput, low latency, reliability and scalability. Each of which has its own advantages and limitations.

Among all these topologies, FAT tree has the low latency [80], better fault tolerance [80] and nonblocking bandwidth characteristic [31]. It can be used to build large scale high performance Data Centers [81]. The Fat-Tree is used as a baseline DCN topology for prototype implementation and performance evaluation. We also note that Fat tree has been used as a base line topology in other works such as Portland [72], Hedera [27]. The advantages of fat tree architecture are listed below:

1. It is highly scalable for large scale Data Centers [49, 80].
2. It is less expensive in terms of operational and capital expenditure as it uses off-the-self commodity switches [49].
3. It provides high bisection bandwidth [49, 82].
4. It has multiple paths between two end hosts [49].

2.2 Software Defined Networking

To handle large amount of traffic [1] with different characteristics [20], the underlying network architecture should react dynamically in real time to provide a suitable service for each traffic type in a very short period. Apart from this, there is a need for an efficient network management to improve resource utilization and better performance [12]. However, current conventional networking technologies cannot meet these demands because of their static nature and dependency on network device vendors [83]. To fill this gap, SDN concept has been introduced by academia [50] and now become the industry standard [84]. SDN based traffic engineering is a recent research area [27, 58] which provides flexible routing management and fine grained traffic control over the MPLS- or IP-based TE [29].

Different from traditional networks in which data plane and control plane are implemented on hardware, SDN [83, 85] decouples the network control plane from the data forwarding plane. Employing a central server, which uses knowledge of switches routing information and network status, SDN manages routing algorithms separately while keeping data flows running on original network paths. Hence, SDN provides much easier modification to switch algorithms, as well as faster control over network status changes. Thus, it provides agile traffic control and management. OpenFlow, was proposed by McKeown et al. [50], is a communication interface between SDN controllers and network devices considered as the most popular enabler for SDN. It is an open standard, which enable users to easily implement experimental routing protocols via software. There are several versions of OpenFlow with the latest version 1.4 [85] being adopted by switch vendors [17] and SDN controller developers [51].

2.2.1 OpenFlow Routing Architecture

OpenFlow is an open standard, which enables users to easily implement experimental routing protocols via software. The key difference between the conventional router and OpenFlow is that the forwarding (data plane) and routing (control plane) layers are decoupled. The network is centrally managed by a dedicated controller such as NOX [51] classic controller as shown in Figure 2.5, which communicates with OpenFlow enabled switches using the OpenFlow protocol. Every OpenFlow switch maintains a flow table with entries for the flows. An entry in the table specifies a few packet fields

2. BACKGROUND AND LITERATURE REVIEW

Figure 2.5: OpenFlow architecture

and associated action to be carried out in the event of a matching with the incoming packet. In the event when a switch is unable to find a match in the flow table, the packet is forwarded to the controller to make the routing decisions. After deciding how to route the new flow, the controller then installs a new flow entry at the required switches, so that the desired actions can be performed on the new flow. OpenFlow switches use per-flow timeouts to manage the lifetime of each flow rule. There are two timeout mechanisms: `idle_timeout` and `hard_timeout` used in OpenFlow [85]. In an `idle_timeout` mechanism, the flow entry will be removed from the flow table if no packet matches the rule within a certain inactivity period. In a `hard_timeout` mechanism, the flow entry will be removed after a given number of seconds, regardless of how many packets it has matched. So, the flow entries will not stay in a flow table forever and will be evicted when the controller needs to reinsert the new flow information into the flow table. There can be two kinds of flow tables: exact match table and wildcard-aware linear table [85] based on the lookup mechanism. The exact match table uses hash calculation for fast lookup, but requires maintenance of a large number of flow entries. On the other hand, the wildcard-aware linear table uses linear search and requires maintenance of a small number of flow entries, but resulting in longer lookup time.

2.2.2 SDN Limitations

Though SDN has attractive features like switch programmability, it has the drawback of high switch cost. SDN performs per-flow routing in which switches maintain flow tables. The flow tables can be built with Ternary Content-Addressable Memory that support wildcard entries and parallel lookups. Current switches usually support very limited number of entries, eg.1500 [17]. OpenFlow defines a framework to perform per-flow routing where switches maintain flow tables. The OpenFlow enabled switches for example HP 5406zl switch hardware [17] can support up to only 1500 OpenFlow rule entries. This is due to the large size of a OpenFlow rule described by 15 header fields with 356 bits [50] which is 6 times more than the conventional L2 switch(48-bit MAC +12 bit virtual local area network (VLAN) id). It has severe impact on SDN switch's performance. First,this flow table size is very small when compared to the number of active flows arriving at the switch [20, 86]. Increasing the flow table size by having more TCAMs which are power hungry (about 15 Watt/1 Mbit)[54] and expensive (US\$350 for a 1M-bit chip) leads to high switch cost. Second, this table can be quickly filled up by small flows (mice flows) such as MSSQL and HTTP which are dominating in the Data Center traffic [20, 48]. Mice flooding in the Data Center network traffic will cause the large flows (elephant flows), which are very large in size (data volume) but few in numbers when compared to mice flows [20], to be evicted prematurely from the flow table, particularly when a burst of many mice flow packets arrive at a switch between elephant flow packet arrivals [65, 87]. Flow table misses cause additional delay due to round-trip time to the controller and in addition, an extra *packet-in* event is generated and sent to the controller, incurring additional load on the controller. Apart from this, more misses in TCAM table resulted in more insert operations and long delay.

2.2.3 Discussion

Leveraging on the the network programmability feature of SDN, we developed Layer-2 multi-path routing to reduce congestion. Though, SDN offers benefits like dynamic programming at switching devices for flexible routing management and fine traffic control, it has limitations due to the large size header fields as explained in previous section. To overcome this, there is a need for a new SDN framework which will save the cost and the power of the SDN switching device. Our proposed differentiated SDN cache

2. BACKGROUND AND LITERATURE REVIEW

architecture is simple which can be adopted in Data Centers. It also ensures fairness for elephant flows reducing the number of their evictions in the presence of mice flooding.

2.3 Traffic Engineering for Data Centers

Traffic Engineering (TE) is defined as “routing optimization for enhancing network service capability without causing network congestion” [88]. The main objective of TE is to balance the network load and minimize the possibility of congestion.

From several Data Center traffic research studies that are available in the literature [20, 31, 48, 86], it is understood that the traffic is bursty and unpredictable in nature [20, 48] and most of them are short flows lasting only a few seconds with flow inter arrival times of less than 1 ms [20]. Though 75% of traffic is kept within racks, links in the core layer are heavily utilized in Data Centres [48]. So, there is a need to control the Data Center traffic to avoid congestion. Traffic engineering and routing techniques have been widely used for Internet topologies [89] that are designed for highly predictable traffic, are unsuitable for DC networks.

As the trend of Data Center traffic moves from the traditional north-south traffic to east-west traffic, there is a need to efficiently use the existence of multiple paths between a pair of hosts [33, 73]. *Equal-Cost Multi-Path (ECMP)* routing method [25] was developed to make use of this redundancy. This technique directs flows based on the hash value of certain fields of the flow. As all packets pertaining to a flow will follow the same path, it is possible that a certain parts of the network are heavily loaded while leaving the other parts lightly loaded. This technique neither considers the flow size nor the network utilization [25]. Therefore, hashing of two or more large flows could end up in sharing the same path or link causing congestion [29].

Conventional STP [23] does not support path diversity which results in limited bi-section bandwidth and poor reliability [75]. Smart Path Assignment In Networks (SPAIN) [57] provides multi-path forwarding over a network built from commodity Ethernet switches to provide multipathing support over arbitrary topologies. It computes disjoint paths between the pairs of edge switches and multiple spanning trees with VLANs are built over the network. Flows are routed hop-by-hop and are restricted to the spanning trees. Agents spread the traffic across the paths evenly and re-route the flows to

alternate paths in the event of failures. Though it supports multipathing, load balancing and fault tolerance, it lacks scalability [75]. The scalability of SPAIN network is limited by the requirement of downloading the switch level topology and the VLAN mappings by each host. In addition, SPAIN requires more entries on switch's forwarding table as the switches store multiple entries for every destination and VLAN [23]. Betoule et.al, proposed distributed source routing algorithm for topology discovery suitable for ISP transport networks [90]. However, our approach differs from this by leveraging centralised controller approach like Software defined networking.

SEATTLE [56] uses a one-hop Distributed Hash Table (DHT) and based on host's location with its MAC address, it forwards a frame along the shortest path. For a broadcast or multicast group, it uses a single spanning tree protocol. While it avoids loops, traffic routing is done over a single path leading to inefficient resource utilization [57].

Transparent Interconnect of Lots of Links (TRILL) is a Internet Engineering Task Force (IETF) Standard [55]. TRILL uses a time to live field (TTL) in the frame header to mitigates the loop problem and deploys multipathing to support scalability. It uses ECMP to deliver frames. This will degrade the performance when TRILL is used on arbitrary topologies because it does not support multiple paths of different lengths [57].

While ECMP makes decisions about routing flows in the Data Centre, trend has been moved towards dynamic routing that adapt to the actual traffic characteristics. ECMP suffers from major drawbacks, as we discussed in Chapter 1. Motivated by the ECMP drawbacks, many multipath routing solutions designed for DCNs have been proposed. We review some of the most relevant works .

Hedera [27] is an adaptive flow scheduling solution based on Fat tree networks. It is built on the Portland Testbed [72]. It monitors the current levels of traffic loads between source-destination pairs and identify large flows. After identifying large flows, it uses one of the two algorithms namely first-fit and simulated annealing to reschedule(reroute) minimal number of active flows to maximize the bisectional bandwidth. Hedera deploys a central network fabric manager to monitor the behavior of large flows and leverage decisions on scheduling among them. While this mechanism performs better than ECMP, it incurs hash overhead in terms of time complexity and messages due to traffic load measurements between source-destination pairs periodically, and possibility of routing and rerouting of large number of flows often.

2. BACKGROUND AND LITERATURE REVIEW

MicroTE [58], a centralized system which leverages the short-term predictability to schedule flows for Data Centres. It was implemented using OpenFlow protocol [50]. Every two seconds, the servers send their average traffic matrix to the OpenFlow controller. Based on that, the controller assigns the predicted traffic flows to the optimal path to minimize maximum link utilization. After that, the remaining unpredictable flows are placed on available paths based on the remaining bandwidth on available paths. This method [58] has lots of overhead and performs slightly better than ECMP.

Tso et al. [91] proposed DCN TE scheme which is the modified version of Penalizing Exponential Flow-splitting (PEFT) to handle the dynamic DCN traffic. Each switch has to monitor the incoming and outgoing traffic volume across all its ports and based on that the optimal routing paths will be calculated. It splits and sends the flows through unequal cost paths which results in packet reordering problem. The run time local traffic matrix (TM) which are collected at switches are useful in making packet forwarding decisions on hop-by-hop basis. It introduces heavy load on switches. Apart from that, routing decisions are not globally optimal.

Li et.al.[92] proposed a OpenFlow based dynamic load balancing scheme for Fat tree topologies. The centralised controller distributes the upcoming flows based on the real-time traffic characteristics, through the alternative paths in such a way that each alternative path receives equal amounts of traffic. However, this method lacks scalability and not suitable for other topologies. It does not have fault tolerance features. The routing decision is made based on the local conditions. So, it is not globally optimal.

2.3.1 Discussion

We consider the following major TE approaches Hedera [27], MicroTE [58], Penalizing Exponential Flow-splitting (PEFT)[91], and Li's load balancing [92] for comparison with our TE approach. Among the aforementioned proposals, Hedera [27], MicroTE [58] performed flow level TE based on predicted traffic demands. They estimate the traffic demands based on the switch's flow counter measurements. However, it is difficult to predict the traffic demands accurately because they cannot predict the traffic before the traffic enters the network. This inaccurate traffic demand leads to undesired performance in network provisioning [93]. The modified PEFT scheme [91] uses run time TM that are collected at the switches to make the forwarding decision hop-by-hop basis and the frequent update of TM's exchange imposes heavy load on switches.

Our SDN-based TE work differs from others by considering the current network state for path selection and flow forwarding so that it can dynamically adapt to the varying traffic conditions which is suitable for Data Center unpredictable traffic nature. We basically map traffic to various paths as and when traffic flows arrive dynamically. We also consider rerouting of a few flows on a link when the link fails or its load exceeds a certain threshold to reduce possibility of congestion. Our mechanisms are more effective and computationally simple making them suitable for practical implementation.

The works above have discussed traffic engineering scheme in data centre networks that are focused on flow scheduling mechanisms. Apart from this, TE issues can be met by adaptively migrating VMs, which contributes the significant amount of traffic [1] in the Data Center. Data Center virtualization requires optimal VM placement to ease network congestion [42, 43, 44, 45].

2.4 Data Center Virtualization and VM Placement

A Virtual Data Center (VDC) is a Data Center where the resources like servers, networks, storage are virtualized. A hypervisor is running on a physical machines (PMs) and creates virtual machines (VMs) on it. In Data Centers with oversubscription, the quantity of network bandwidth available between VMs is determined by the physical location of the VMs affecting its performance. Figure 2.6 shows an example of a Data Center with four tenants (VDCs) each with 2 VMs sending traffic between them. The VMs of the tenants are highlighted with different fonts in the figure. At the top tier, the links are oversubscribed because of the significant VM traffic which result in poor network performance for all tenants. A poor placement of VMs causes network bottleneck. As the system scales from single rack, it is important to consider the network knowledge for VM placement. Considering network knowledge for VM placement can increase traffic locality to reduce network bottlenecks and decrease latency, that leads to improved application performance and support for increased number of applications.

In virtualization-based Data Centers, the critical issue which attracted significant attention is the problem of VM placement. The VM placement problem uses resource requirements of VMs as input and based on the available capacity of the physical servers, VMs are placed on to servers sharing the resources in accordance with the Data Center

2. BACKGROUND AND LITERATURE REVIEW

Figure 2.6: Illustration of VM placement

policies [67]. As the number of VMs and physical hosts increases in a Data Center, automation is required to do VM to server mapping. The number of possible mappings to be evaluated for a given set of virtual and physical machines (servers) is on the order of $O(n^m)$ where n is the number of physical servers and m is the number of virtual machines. Such a large solution space makes it impossible for any brute force VM-host mapping algorithm to give exact placement plans in practically acceptable times. Hence, intelligent placement heuristics are needed to narrow down the search to obtain a near-optimal placement solution.

There are mainly three kinds of works focusing on the VM placement problem. One is to consider how to place VMs with the constraints of the physical server's limited resources and the second one is to consider network resource optimization while placing the VMs and the third one is to consider energy saving while placing the VMs. In the field of VM placement, there are lots of outstanding achievements. However, we consider only the significant ones.

2.4.1 VM Placement with Server Resource Constraints

In recent years, VM placement problem has been studied by several researchers. Initially, researchers considered the placement of VMs with the resource constraints of the physical servers only. The works carried out in [39, 40, 41] addressed the VM placement problem considering only the server resources and not the networking aspects. In [39], Sindelar *et al.*, demonstrated that VM placement based on memory share that can save the memory resources of a server. In their proposed placement scheme, VMs with the most shared pages are co-located in the same physical server. In [40], Bobroff *et al.* adopted prediction techniques while minimizing the number of active physical machines (PMs), and present mechanisms for dynamic migration of VMs based on a workload forecast. Different from other VM placement methods, these works consider user's preference in resource utilization and they consider the importance of reducing the number of servers used and balancing the resources used, but they do not consider network topology and VM communication traffic. As a result, there is a possibility that communicating VMs are placed on servers separated by long paths with high communication cost.

2.4.2 Network-aware VM Placement

With the increasing traffic within a Data Center, researchers working on VM placement started to focus on traffic engineering mechanisms. The location of the communicating VM pairs in the network topology and the network links utilization in between them affect the overall performance of the workload inside the VMs [42, 43, 44, 45]. Considering the resource constraints of the servers [39, 40, 41] will affect the performance of communication between VMs [47]. The following VM placement schemes have considered the problem of improving overall network performance by balancing the load across the communicating VMs.

In [42], the problem of joint VM placement and routing in Data Center has been studied. This work considers the impact of VM placement on routing and traffic engineering. It provides a solution for the management of network resources and the VM placement under the dynamically changing traffic environment. The proposed algorithm is based on Markov approximation and required small number of VM migrations. This work can be adopted in Fat-Tree and BCube topologies [43].

2. BACKGROUND AND LITERATURE REVIEW

VM placement technique for heterogeneous infrastructure has been proposed in [45]. The proposed scheme was implemented in two phases. The first phase identifies a potential group of PMs which are named as cohort, and the second phase maps the VM-to-PM based on the characteristics of cohort and the user level requests. This scheme can scale up to several hundreds of physical nodes and the latency among deployed VMs is reduced. The deployment is unaffected by the size of the physical infrastructure. This is suitable for LAN and STAR network [43].

An application-aware virtual machine placement in Data Centers has been proposed in [67] which suggest reducing the number of physical machine and communication distance between VMs using convex optimization theory in a scalable manner. For placing the VMs, this algorithm considers server constraints as well as the dependency relations between the VMs.

In [46], the authors studied the problem of Traffic-aware VM placement to reduce the aggregate traffic between the VMs in a Data Center. It is an NP hard [46] problem and they used a Local optimal Pairwise Interchange (LOPI) and simulated annealing heuristic approach. Traffic matrices and cost among VMs are taken as input. It uses different cost function and places only one VM on each host without considering link capacity constraints which will lead to congestion on links in the Data Center. However, this work does not consider a dynamic online problem which uses the network state with residual link capacities for VM placement and traffic mapping. They proposed the off-line algorithms for clustering of servers and VMs. It is important to note that, application of an offline framework to solve the problem online can result in inefficient migration of VMs. Apart from that, they did not consider the VM migration traffic, leading to a near complete shuffling of almost all VMs in each round.

In [94], the Min Cut Ratio-aware VM Placement (MCRVMP) problem has been studied to determine the location of VMs in a Data Center based on the network bandwidth, CPU, and memory resources. The formulation is complex and does not scale to the size of Data Centers given that the number of VMs in a Data Center can be as large as thousands of or tens of thousands of VMs [48].

Remedy [95] balances the network by migrating the VMs depending upon bandwidth utilisation statistics collected from intermediate switches. However, Remedy provides load-balancing across the data centre network, rather than routing traffic over lower level, and lower cost links in the network to improve pairwise locality for VMs.

2.4.3 Power-aware VM Placement and Migration

Energy Efficiency [69] is becoming important in Data Center networks. Most research on early work focused on the energy savings on server side [96] and cooling systems [97] without giving consideration to network aspect. Since, today's networking elements are not energy proportional [98], it has been found that the ideal state network element energy consumption is 85% more than that of the working state [99, 100]. Energy-aware routing is one possible way to save energy in the Data Center by the partial shutdown of servers or network elements when it is at idle state [46, 69]. Several research studies are done on VM consolidation to reduce the traffic load and also reduce power consumption in Data Centers [46, 101].

ElasticTree [69] is designed for energy saving in Data Centers. It reduces energy consumption of a Data Center network by turning off the links and switches that are not required to meet the varying traffic demands at the time. It uses small subset of switches and links to serve the traffic flows in the data center network and the remaining unused network elements can be turned off for power saving. It proposed three algorithms namely, Linear Programming (LP)-based formal model, greedy bin packing heuristic, and topology-aware heuristic to find the subset of networking elements to satisfy the traffic demands. Shang et al. [102] proposed a routing technique for reducing the number of switches used for routing a given traffic matrix over a given Data Center topology. Though these two techniques [69, 102] focused on energy-aware traffic routing, they did not integrate it with the appropriate placement of VMs.

In [44], authors considered a Network-Aware Virtual Machine Placement (NAVP) problem named *VMFlow*. They provided a heuristic approach that places VMs, one by one to minimize the data transfer delay. Their VM migration scheme increases locality, network performance and reduces the energy cost. However, their approach leads to the situation where the VMs cannot be placed near to their communicating pairs due to the powered down of networking equipments. This results in the degradation of network performance. Apart from that their one-to-one VM placement mapping assumption may not be valid for several applications [70].

Mahadevan et al. [68] proposed a system Urja, which monitors and analyzes the network traffic and power usage. Based on the measurement results, various approaches for saving energy in Data Centers such as server load consolidation, VM placement

2. BACKGROUND AND LITERATURE REVIEW

optimization are suggested by the authors. However, their implementation techniques do not hold well in practice.

The basic idea of VMPlanner[70] is to optimize both virtual machine placement and traffic flow routing so as to turn off as many unwanted network elements as possible for power saving. VMPlanner considers network-aware virtual machines placement inside the Data Center, and the power efficient traffic flow routing between the virtual machines.

2.4.4 Discussion

Though, the above mentioned VM placement schemes focus on improving the overall network performance by balancing the load across the network by strategic VM placement and followed the off-line method [46]. There is a research gap in the VM placement problem that is able to do traffic routing so that traffic can be protected against failures in addition to reducing congestion. The use of link-disjoint paths ensures the availability of at least one path for the traffic upon a link failure, thus guaranteeing a certain bandwidth associated with the surviving paths. Our proposed scheme differs by considering online approach for allocation of VMs and routes traffic across multiple paths for reduced occurrence of congestion while satisfying a certain protection grade which is defined as the fraction of rate (or bandwidth) guaranteed to be available in the event of single link failures, without using additional bandwidth. Our proposal on power-aware migration saved energy by considering server and network components. It aggregates virtual machines to a few physical machines to reduce server power consumption and also communicating virtual machines would be migrated into close locations to save power consumption of link and switches. Our proposal is different from others by reducing the Data Center network traffic and improving the energy consumption by proper routing and VM allocation.

2.5 Summary

This chapter surveys the various Data Center topologies, software defined networking and mile stone research techniques on Data Center traffic engineering, VM placement strategies documented in various literatures. A brief discussion describing the focus of our work and approach is given. In summary, the different techniques have been

designed to achieve different goals with their respective strengths and limitations as discussed in this chapter.

2. BACKGROUND AND LITERATURE REVIEW

Chapter 3

Adaptive Layer-2 Multi-path Forwarding

This chapter presents the proposed adaptive load balancing framework. The chapter describes various functional modules used in the framework to accomplish the load balancing task based on the dynamic network state. Details of the load balancing algorithm, simulation set up and the performance study are described. The proposed Adaptive Layer-2 algorithm differs from the previous algorithms like TRILL [55], Shortest Path Bridging (SPB)[103], Portland [72] and SPAIN [57] by distributing the traffic over multiple paths and adjusting the traffic distribution in the real time manner to reduce congestion. The previous proposals use ECMP or single path to route a traffic flow. The proposed mechanism is computationally simple and can be easily adopted in the Data Center. The research work presented in this chapter has been published in [5].

3.1 System Architecture

We consider a 3-tier hierarchical architecture, which has equal cost multiple paths between every pair of edge nodes. We note that, the proposed method can be used with any arbitrary topology which has multiple paths between the end hosts. Figure 3.1 depicts a 3-tier architecture which consists of core switches at the root of the tree followed by aggregate switches which are at the next higher level, and finally the edge switches at the lowest level in the hierarchical architecture. The existence of multiple

3. ADAPTIVE LAYER-2 MULTI-PATH FORWARDING

Figure 3.1: Three Tier Fat tree architecture.

paths between host C and host J is illustrated with an example. The multiple paths which exist between these hosts are : $host\ C - E_2 - A_2 - C_3 - A_6 - E_5 - host\ J$, $host\ C - E_2 - A_2 - C_4 - A_6 - E_5 - host\ J$, $host\ C - E_2 - A_1 - C_2 - A_5 - E_5 - host\ J$, $host\ C - E_2 - A_1 - C_1 - A_5 - E_5 - host\ J$. We use E_i , A_j and C_p to denote edge switch i , aggregate switch j and core switch p , respectively. Note that host C is connected to switch E_2 and host J is connected to switch E_5 . We choose one of the paths based on the load metrics to achieve maximum bandwidth utilization and minimize congestion.

3.1.1 Addressing Scheme

The poor scalability due to the traditional Layer-2 flat MAC address space motivates us to use a hierarchical addressing scheme. We group the aggregate and edge switches to form a pod or zone. All core switches are assigned to the same Pod. The address of the host is denoted as $podId.layerId.switchId.hostId$, where $podId$ denotes the pod number containing the host, $layerId$ denotes the layer of the switch (core switches are

Figure 3.2: HMAC addressing scheme

assigned 3, aggregate switches are assigned 2, edge switches assigned 1 and hosts are assigned 0), *switchId* denotes the switch's position in the layer (hosts are assigned 0) and *hostId* denotes the position of the host in the Pod. We refer to this location-specific addressing scheme as *Hierarchical MAC Addresses* (HMAC). The format of our HMAC scheme is shown in Figure 3.2. This addressing scheme can scale up to 4.1M hosts with 1:1 oversubscription ratio at both the aggregate layer and edge layer. This is explained as below.

- Maximum number of pods = $256-1$ (for core pod) = 255; No of. pods must be even; so Maximum number of Pods = 254;
- number of edge switches in a pod = $254/2 = 127$
- number of hosts connected to each switch = 127

3. ADAPTIVE LAYER-2 MULTI-PATH FORWARDING

- Total number of hosts this addressing scheme can scale up to with 1:1 oversubscription ratio = $254 \times 127 \times 127 = 4,096,766$ approximately 4.1M

The addressing scheme used by us for the fat tree described above is different from the addressing scheme proposed by [49]. The latter uses a scheme similar to IP addressing with longest prefix matching procedure. In our scheme, the HMAC is attached to the packet by the edge switch. This HMAC is used by various switches for routing particularly for deciding interpod or intrapod communication.

3.1.2 Chief Supervisor

The network is centrally controlled by Chief Supervisor (CS) as shown in Figure 3.1. The CS is similar to SDN controller such as NOX [51]. It keeps track of the soft state about the whole network topology and has the connectivity to all the switches in the network. The CS collects information about the load on various links as and when needed. When a link fails, it will notify all the edge switches about the failure so that they can quickly react to the failure. The CS does not maintain any table hence it is stateless.

3.2 Proposed Load Balancing Framework

The main crux of the algorithm is to measure the load on the possible paths and distribute the traffic load across multiple paths in proportion to the load information estimated. We use this load-balanced routing only when the load exceeds a certain threshold, for scalability reasons. When the load is low, we forward packets using known hash based routing technique.

The main modules in our load balancing framework are,

- The *Route* computation module computes possible candidate paths between any pair of end hosts.
- The *Load Measurement* module measures the load on various links and sends this information to edge switches through the chief supervisor.
- The *Load-aware Routing* module distributes the packets to various paths based on the load information obtained for the path.

- The *Fault-Tolerant routing* module redistributes the traffic on the failed links to the other working paths.

The following subsections describe the functionalities of each module in detail.

3.2.1 Route Computation

There are two kinds of routing namely interpod and intrapod routing. In intrapod routing, the two end hosts are located in the same pod. In this case, the paths between these hosts traverse the edge and aggregate switches that are residing only within the pod. In interpod routing, the two end hosts are located in two different pods. So, the paths between these hosts traverse the edge, aggregate and also the core switches. Here, each of the paths from a source host reaches one core switch in the upward direction and there exists a path from a core switch to a destination host in the downward direction. We pre- compute the possible routes between the end hosts.

Table 3.1: Multiple paths between host E and host H

Path	Route
1	$E_3 - A_3 - E_4$
2	$E_3 - A_4 - E_4$

Table 3.2: Multiple paths between host C and host P

Path	Route
1	$E_2 - A_2 - C_3 - A_8 - E_8$
2	$E_2 - A_1 - C_2 - A_7 - E_8$
3	$E_2 - A_1 - C_1 - A_7 - E_8$
4	$E_2 - A_2 - C_4 - A_8 - E_8$

The following example is used to illustrate the intrapod routing in Figure 3.1. Consider host E and host H which are in the same pod. The possible routes are shown in Table 3.1. Note that these paths do not traverse any core switch. Next, we consider the case of routing between host C and host P which are in different pods, which require interpod routing as shown in Figure 3.1. The possible routes are shown in Table 3.2.

3. ADAPTIVE LAYER-2 MULTI-PATH FORWARDING

3.2.2 Load Measurement

To route the packets along the lightly loaded path, the load on the links along the possible paths needs to be determined. After computing the possible paths between the source and destination hosts, the source edge switch sends the *path_load_req* control packet to the chief supervisor requesting for the required load information on the links along various paths. This control packet carries information that includes source edge switch, destination edge switch, and the paths between them. Upon receiving this control packet, the chief supervisor first extracts the details of various switches and links along the paths specified in the control packet called *link_load_req*. The chief supervisor collects link load information from various switches along the path by sending a control packet named *link_load_req*. The control packet *link_load_req* contains the details of the switch and its associated links whose load information is requested. Upon receiving a *link_load_req* packet, a switch returns the load information of the specified links based on its queue length information. This load information is sent back to the chief supervisor using the control packet namely *link_load_reply*. After receiving *link_load_reply* control packets from various switches, the chief supervisor calculates the path load for each of the paths as required by the source edge switch. We measure the link load and path load in terms of number of packets per second. The path load on a path is computed as the maximum load on any of the links traversed by the path. The CS then passes the link load details to the edge switch by using the control packet *path_load_reply*, with the details of paths and the associated loads.

The load measurement mechanism is illustrated with an example. We consider the case of routing between edge switch E_2 and edge switch E_8 . First, the control packet *path_load_req* (E_2, E_8, P) is sent from the source edge switch E_2 to the CS. Here, P represents the set of paths p_1, p_2, p_3, p_4 , where p_1 is $A_2 - C_4 - A_8$, p_2 is $A_1 - C_2 - A_7$, p_3 is $A_2 - C_3 - A_8$, and p_4 is $A_1 - C_1 - A_7$. Upon receiving the *path_load_req* packet from E_2 , the chief supervisor first extracts the details of various switches $A_1, C_1, A_2, C_2, A_3, C_3, A_4$ and C_4 , and links along the paths specified in the control packet.

The chief supervisor prepares control packets *link_load_req*(s, L) for each switch S . Here, L is the set of required links at S . One such packet is *link_load_req* ($A_1, \{A_1 - C_1, A_1 - C_2\}$). Upon receiving a *link_load_req*, switches $A_1, C_1, A_2, C_2, A_3, C_3, A_4$ and

C_4 return the load information of the specified links based on the queue length information. The load information for each of the specified links is sent by switch s back to the CS using the control packet namely `link_load_reply`($s, \{< link, load >\}$). For example, switch A_1 sends the packet `link_load_reply` ($A_1, \{< A_1 - C_2, 4 >\}, \{< A_1 - C_1, 3 >\}$). After receiving the `link_load_reply` control packets from various switches, the chief supervisor calculates the path load for each of the paths as the maximum load on any of the links traversed by the path. Mathematically this is represented as path load pl_{ij}^m where i and j are edge switches and m is the path identifier.

$$pl_{ij}^m = \max_g(link_g^m) \forall g \in \text{links on path } m \quad (3.1)$$

The chief supervisor prepares the control packet `path_load_reply` ($< p_1, pl_1 >, < p_2, pl_2 >, < p_3, pl_3 >, < p_4, pl_4 >$), where p_i and pl_i represent path i and its path load, respectively. As an example, we show the path and path load information in Table 3.3. When edge switch E_2 receives the `path_load_reply` packet, it stores the information in the `load_information` Table as shown in Table 3.3. The load measurement protocol operations are shown in Figure 3.3.

Table 3.3: Path load information table at switch E_2

Path	Route	Pathload
1	$E_2 - A_2 - C_3 - A_8 - E_8$	2
2	$E_2 - A_1 - C_2 - A_7 - E_8$	1
3	$E_2 - A_1 - C_1 - A_7 - E_8$	4
4	$E_2 - A_2 - C_4 - A_8 - E_8$	1

3.2.3 Load-Aware Adaptive Routing Algorithm

The proposed *Load-Aware Adaptive Routing* (LAAR) algorithm is described in this section. Using the information in `load_information` table, a switch determines the proportion of the traffic to be sent along various possible paths. Let p_i be path i , pl_i be the path load for path i , and k be the number of paths between a given pair of edge switches. The fraction f_i , $i = 1, 2, \dots, k$ of the traffic to be sent through path i is calculated as follows:

$$f_i = \frac{\frac{1}{pl_i}}{\sum_{i=1}^k \frac{1}{pl_i}} \quad (3.2)$$

3. ADAPTIVE LAYER-2 MULTI-PATH FORWARDING

Figure 3.3: The load measurement protocol

The traffic is split among the paths according to the fractions calculated as shown in equation 3.2. The path id information is attached to the packet, so that other switches will know the path or output link through which the packet is to be forwarded. Table 3.4 below shows the fractions calculated for the paths between host C and host P, in Figure 3.1. The pseudocode is shown in Algorithm 1.

Table 3.4: Fraction of the traffic transmitted along the paths based on the load

Path	Route	Pathload	Fraction
1	$E_2 - A_2 - C_3 - A_8 - E_8$	2	0.25
2	$E_2 - A_1 - C_2 - A_7 - E_8$	1	0.5
3	$E_2 - A_1 - C_1 - A_7 - E_8$	4	0.125
4	$E_2 - A_2 - C_4 - A_8 - E_8$	4	0.125

Algorithm 1 Load-aware adaptive routing.

```

1: /* Here  $i$  and  $j$  are edge switches,  $k$  is the number of edge switches and  $m$  is the
   path identifier */
2: for each flow  $F$  do
3:   if flow  $F$  is new then
4:     if load in source edge switch  $<$  threshold  $T$  then
5:       Perform hash routing
6:     else
7:       if load  $l_{ij}^m$  has been determined  $\forall m$  then
8:         for  $m=1$  to  $k$  do
9:            $f_{ij}^m = \frac{\frac{1}{p_{ij}^m}}{\sum_{m=1}^k \frac{1}{p_{ij}^m}};$ 
10:        end for
11:       for every  $n$  packets do
12:         Send  $n * f_1$  packets through path 1;
13:         Send  $n * f_2$  packets through path 2;
14:         ...
15:         Send  $n * f_m$  packets through path  $m$ ;
16:       end for
17:     end if
18:   end if
19: end if
20: end for

```

3.2.4 Fault Tolerance

This section describes how our LAAR algorithm redistributes the load in the event of a link failure. When a link fails, the switch sends the control packet called *failure_notification* to the CS about the link failure containing the identifier of the two switches connecting the failed link. The Chief Supervisor upon receiving this *failure_notification* packet forwards the packet to all the edge switches. An edge switch upon receiving the *failure_notification* packet checks if this link is traversed by any of the paths in its *load_information* table. Thus, the number of messages exchanged during the failure scenario is proportional to the number of the edge switches in the network.

To facilitate redistribution of the traffic on the failed paths to the working paths, the fraction of traffic to be distributed on the paths are recalculated. Without the loss

3. ADAPTIVE LAYER-2 MULTI-PATH FORWARDING

Figure 3.4: Illustration of link failure

of generality. Suppose that path pl_k fails among the k paths between a given pair of edge switches. The fraction f_i of traffic for $path_i$, $i = 1, 2, \dots, k-1$ is calculated as follows:

$$f_i = \frac{\frac{1}{pl_i}}{\sum_{m=1}^{k-1} \frac{1}{pl_i}} \quad (3.3)$$

Suppose that link $A_2 - C_4$ fails in the network as shown in Figure 3.4. The corresponding switches A_2 and C_4 send the $notification_failure(A_2 - C_4)$ packet to the chief supervisor. Upon receiving this packet, the chief supervisor forwards it to all the edge switches E_1 through E_8 . Edge switch E_2 , now finds that path 4 traverses this failed link $A_2 - C_4$ from Table 3.4. It then calculates the fractions of the traffic to be distributed among the paths other than path 4. The updated values are shown in Table 3.5.

Table 3.5: Fraction of the traffic distributed along the paths after failure

Path	Route	Pathload	Fractions
1	$E_2 - A_2 - C_3 - A_8 - E_8$	2	0.286
2	$E_2 - A_1 - C_2 - A_7 - E_8$	1	0.571
3	$E_2 - A_1 - C_1 - A_7 - E_8$	4	0.143
4	$E_2 - A_2 - C_4 - A_8 - E_8$	4	0

3.3 Performance Study

The performance of the proposed LAAR algorithm has been evaluated by using Omnet++ 4.0, an open-source, extensible, modular, component-based C++ simulation library and framework [104].

We use different traffic patterns, namely, $Stride_1$, $Stride_2$, $Stride_4$, $Stride_8$ and Bijective mapping. Such traffic patterns have earlier been used as benchmark in the literature [49]. The results obtained for LAAR is compared with that of the static hash-based algorithm which is a well known ECMP algorithm. The performance metrics considered are bisectional bandwidth, load distribution, and link utilization. The traffic patterns considered are described below.

- Stride1 - in this pattern, each $host_i$ sends packets to $host_{(i+1) \bmod 16}$
- Stride 2 - in this pattern, each $host_i$ sends packet to $host_{(i+2) \bmod 16}$
- Stride 4 - in this pattern, each $host_i$ sends packet to $host_{(i+4) \bmod 16}$
- Stride 8 - in this pattern, each $host_i$ sends packet to $host_{(i+8) \bmod 16}$
- Bijective - here every host randomly selects a destination host and each host is a source for only one traffic flow and destination for only one traffic flow.

For each of the traffic patterns considered, each host sends 10000 packets, to the destination host. Each switch in the three tier fat tree architecture considered is a four port switch and there are 16 hosts in total. Figure 3.5 below shows the simulation setup using Omnetpp 4.0 simulator. For the calculation of bisection bandwidth, the network is split into two halves by a vertical divider line such that each half of the network has equal number of nodes and the two halves are symmetric. The bisection

3. ADAPTIVE LAYER-2 MULTI-PATH FORWARDING

Figure 3.5: Simulation setup using Omnetpp simulator

bandwidth is the sum of the bandwidths of the links crossing the divider line. In the topology considered here, the divider line which splits the network into two equal halves is between C_2 and C_3 . Hence for the calculation of bisection bandwidth the links considered are $C_1 - A_6, C_1 - A_7; C_2 - A_6, C_2 - A_8; C_3 - A_2, C_3 - A_3; C_4 - A_2, C_4 - A_4$. Figure 3.6 depicts the divider line for the topology considered in this chapter. For the sake of simplicity, the CS is not shown in Figure 3.6.

Figure 3.7 plots the bisection bandwidth obtained using various traffic patterns considered under study. From Figure 3.7, it is clear that the bisection bandwidth of LAAR is higher for most of the traffic patterns compared to the well known ECMP routing. This is due to the fact that in LAAR, traffic is equally distributed across all the switches and roughly equal amount of data flows across the divider line which is not the case with the ECMP. For stride 8 and bijective pattern, the bisection bandwidth of LAAR is about 3 times higher than ECMP. The plots in Figure 3.8 and Figure 3.9 show the load distribution across the core switches by our LAAR and the ECMP algorithm respectively. It can be observed that using LAAR the load gets distributed more evenly

Figure 3.6: Divider line dividing the network into two equal halves

Figure 3.7: Comparison of ECMP and our proposed LAAR algorithm in terms of bisection bandwidth [5]

3. ADAPTIVE LAYER-2 MULTI-PATH FORWARDING

Figure 3.8: Load distribution using LAAR [5]

Figure 3.9: Load distribution using ECMP [5]

across the core switches thereby not overloading a particular core switch. The ECMP routing performs worse for the traffic patterns considered. As seen in Figure 3.9, it overloads core switches C_1 and C_4 and core switches C_2 and C_3 are lightly loaded.

Figure 3.10: Load distribution for stride 8 traffic pattern using LAAR and ECMP algorithms [5]

To get a better understanding of the load utilization, we use the traffic pattern stride 8 and the result of the two algorithms are compared and plotted in Figure 3.10. As expected, LAAR has even load distribution when compared to ECMP routing. Two of the core switches C_1 and C_4 are heavily loaded whereas the other two core switches C_2 and C_3 are lightly loaded. Our LAAR algorithm reduces the stress of core switches C_1 and C_4 upto 25% when compared to ECMP.

Figure 3.11 plots the link utilization using LAAR for the traffic patterns, stride 4 and stride 8. For the stride 4 pattern the link utilization averages to 655 Mbps for the links considered as shown in the Figure 3.11. Similarly for the stride 8 pattern, the link utilization averages to 625 Mbps.

Figure 3.12 compares the scenario of a link failure with that of the normal working condition. From Figure 3.12, it is clear that during the normal operation, the load is more evenly distributed across the core switches. The figure also depicts the load

3. ADAPTIVE LAYER-2 MULTI-PATH FORWARDING

Figure 3.11: Link utilization using LAAR algorithm [5]

Figure 3.12: Load distribution without and with a link failure [5]

distribution for the failure of the link $A_2 - C_4$. Here the packets reaching C_4 are affected and thus packets arriving at C_4 are less compared to the other switches. Further, we can observe that the packets are more evenly re-distributed among the other core switches i.e. C_1 , C_2 and C_3 .

3.4 Summary

In this chapter, an adaptive layer-2 load balancing framework has been developed and its functional modules, load balancing algorithm and simulation setup have been described. The framework dynamically distributes traffic across multiple paths based on the path load conditions and effectively redistributes the traffic across the working paths in the event of link failures. The effectiveness of the proposed load balancing mechanism has been demonstrated by simulations using various traffic patterns.

3. ADAPTIVE LAYER-2 MULTI-PATH FORWARDING

Chapter 4

SDN-based Edge-to-Edge Rerouting

This chapter presents an SDN-based adaptive routing framework developed with the objective of dynamically reallocating different flows in Data Center networks in the event of congestion and faults. The chapter describes various functional modules used in the framework to accomplish the load balancing task based on the dynamic network state. Details of the rerouting scheme, prototype implementation and the performance study on various topologies are described. Recent multipath solutions Hedera [27], MicroTE [58] performed flow level traffic engineering based on predicted traffic demands without considering the current network state. As we discussed in chapter 2, it is difficult to predict the traffic demands accurately before the traffic enters the network. This inaccurate traffic demand leads to undesired performance in network provisioning [93]. The advantages of SDN including fine-grained flow control, software based routing management and resilient from failures motivated us to develop a SDN-based TE framework which differs from others [27, 58, 91] by considering the current network state for path selection and flow forwarding so that it can dynamically adapt to varying traffic conditions in a Data Center. The research work presented in this chapter has been published in [6].

Figure 4.1: OpenFlow testbed

4.1 Routing Control Framework

We consider an Openflow-enabled network as shown in Figure 4.1. The network is centrally managed by a dedicated controller which communicates with the OpenFlow-enabled switches using the OpenFlow protocol. With OpenFlow, it is possible to use various routing algorithms within the controller to generate the forwarding tables that govern the flow based edge-to-edge re-routing of large flows in the data plane.

Let us assume that Host 1 is generating a mix of large flows and small flows to Host 2. The set of candidate shortest paths from Host1 to Host 2 are $A-B-D-F$, $A-C-E-F$, $A-C-D-F$ and $A-B-E-F$. In Figure 4.1, suppose the routes $A-B-E-F$ and $A-C-E-F$ are selected for routing traffic flows from Host 1 to Host 2, link E-F may be overloaded. We propose an edge-to-edge rerouting scheme in SDN -based Data centers to avoid this problem. The basic idea is to use least loaded route first. When congestion occurs on a link, we dynamically re-route one or more large flows to alternate paths and allow small flows to continue their traversal because of their short-lived nature. We developed a prototype with functional modules implemented in the NOX-Classic controller [51]. Our rerouting framework and its functional modules are shown in Figure 4.2.

Figure 4.2: OpenFlow-based re-routing control framework

4.1.1 Framework Modules

The proposed Re-routing framework has four modules namely Monitor, Host Tracker, Routing Engine and Route Tracker. The functions of each module are described in this section.

4.1.1.1 Monitor

The key idea behind our algorithm is to route the packets along the least-loaded path. In order to find the least loaded path, the load on the links along the possible paths needs to be determined by collecting the switch statistics. The main functionality of the *Monitor* module is to collect the statistics from all the OpenFlow switches by polling them at fixed intervals. The per-table, per-flow and per-port statistics are gathered from the OpenFlow switches and are stored in memory as a snapshot object. Each snapshot is identified by a sequence number, which increments by 1 at a regular time interval. The two most recent snapshots are maintained in memory at any instant of

4. SDN-BASED EDGE-TO-EDGE REROUTING

time. The *Routing Engine* module makes use of this statistics to compute the load on various links and determine the least loaded path.

4.1.1.2 Host Tracker

The *Host Tracker* module maintains the list of all the hosts that are detected on the network. It stores the MAC address of the interface of the hosts connected to the network, IP address assigned to the interface, the OpenFlow switch id to which each of the hosts is connected and the OpenFlow switch port number to which each of the hosts is connected. Table 4.1 shows an example of the entries stored in this module for the case of two hosts connected to port 1 of the switches A and F respectively as shown in Figure 4.3.

Table 4.1: Host tracker entries

MAC Address	IP	Switch ID	Port No.
21 – AA – 32 – FF – 1C – 4A	10.200.40.1	A	1
16 – EE – 21 – ED – 2D – 9B	10.200.40.2	F	5

4.1.1.3 Routing Engine

The *Routing Engine* module is the brain of our framework and interacts with all the other modules to determine the least loaded candidates paths between any pair of end hosts. When a packet from a new flow from a host arrives at an OpenFlow switch the first time, it is forwarded to the controller because there is no match with its installed flow entries. Upon receiving the packet, the *Routing Engine* module in the controller will extract the flow's source and destination host addresses. The Routing Engine will then contact the *Host Tracker* to obtain the information about the switch connected to the source host and the switch connected to the destination host. Upon receiving this, it computes all the possible shortest routes. Once the set of all shortest routes R is computed, the statistics from the Monitor module will be used to compute the least loaded route.

The mechanism of the *Routing Engine* is illustrated with an example. The case of routing between Host 1 and Host 2 is considered as shown in Figure 4.3. Here, R represents the set of candidate shortest paths $\{r_1, r_2, r_3, r_4\}$ where r_1 is $A - B - D - F$,

r_2 is $A - C - E - F$, r_3 is $A - B - E - F$, and r_4 is $A - C - D - F$. At the initial phase, if the amount of statistics collected is insufficient, a route is randomly picked from R . The total cost of each route $r_i \in R$ is defined as,

$$\gamma_i = \alpha_i + \beta_i \quad (4.1)$$

where α_i is the total fixed costs of each link in the route, $link_k \in r_i$, and β_i is the total variable costs of $link_k \in r_i$. Each $link_k$ has a predetermined fixed weight w_k , a link load of L_k bps computed from the statistics, and link capacity C_k bps. Currently in our algorithm, all link weights are set to 1. The value of L_k is estimated from the change in byte count for all flow entries in a switch between the two most recent snapshots from the *Monitor* module. After computing γ_i from all $r_i \in R$, the controller picks the route that has the minimum γ_i and installs the flow entries needed into the series of switches in the shortest path. Mathematically the total fixed cost and the variable cost of route r_i is given as,

$$\alpha_i = \sum_k^N w_k; \quad (4.2)$$

$$\beta_i = \sum_k^N \frac{L_k}{C_k} * \frac{w_k}{\alpha_i} \text{ where } link_k \in r_i \quad (4.3)$$

The cost of each link is shown in Figure 4.3. The Routing Engine maintains the path and the cost information in the NOX controller as shown in Table 4.2. Based on the total cost, the least cost route A-C-E-F will be chosen. The Routing Engine will then pass the information of this chosen route to the *Route Tracker* module which will install this route as flow entries at the switches A, C, E and F.

Table 4.2: Cost information

i	Path	Total Cost(γ_i)
1	A-B-D-F	3.85
2	A-C-E-F	3.1
3	A-B-E-F	3.25
4	A-C-D-F	3.60

The load across all the links will be checked periodically by *Routing Engine* module. We set the threshold for congestion as 75% of the link capacity. At any instant, if any

4. SDN-BASED EDGE-TO-EDGE REROUTING

Figure 4.3: Example of path cost comparison

one of the link load exceeds this threshold value T , the controller will re-route one or more large flows across the link to the alternate path one by one. To identify large flows, we use the flow statistics mechanism. Each flow is monitored at the first switch traversed by the flow. Statistics are collected from the switches by the controller at a fixed interval of p seconds and are used to classify the large flows. It is computed as follows:

$$\psi_t = \frac{b_t - (b_{t-p})}{p} \quad (4.4)$$

where ψ_t is the estimated flow size at time t , and b_t is the total number of bytes of the flow received by the switch at time t . In our implementation, p is set to 5 seconds.

The re-route mechanism is illustrated with an example. Let F represent the set of a mix of large flows $\{f_{1L}, f_{3L}, f_{4L}\}$ and small flows $\{f_{2S}, f_{5S}\}$ generated by Host 1. According to Table 4.2, if the routes A-C-E-F and A-B-E-F are selected for routing of traffic from Host 1 to Host 2, the path load of link E-F may exceed the congestion threshold. When the path load of link E-F nears the congestion threshold, the *Routing Engine* module re-routes the large flow f_{3L} , which is determined by the flow

statistics mechanism, to other least loaded alternative path A–C–D–F. Before choosing the alternative path, it makes sure that the large flow f_{3L} at its current size will not overload A–C–D–F. Upon flow completion, the controller will again be notified of the flow removal by the switches after a certain period of flow entry inactivity. When the controller receives this *flow entry removal message*, the *Route Tracker* will check which of the installed flows is to be removed and remove those flows from the list maintained by it.

4.1.1.4 Route Tracker

The Route Tracker module is responsible for the installation of the new route chosen by the *Routing Engine* into the series of OpenFlow switches as flow entries. This module maintains the information about all the installed routes as well, as shown in Table 4.3. Each entry in the table contains the information about the flow attributes like source MAC, Destination MAC, Type and the route installed for this particular flow entry. In the event of a link failure, this module will redirect the affected installed routes.

Table 4.3: Route tracker records

Entry	S.MAC	D.MAC	Type	IP PROTO	Route
1	11-AA-23-FE-1C-4A	22-BB-22-EA-2D-9B	IP	TCP	A-B-D-F
2	22-BB-22-EA-2D-9B	11-AA-23-FE-1C-4A	IP	UDP	F-D-C-A

4.2 Fault Tolerance

This section describes how the load will be redistributed in the event of a link failure. When a link fails, the *Route Tracker* module will be notified by the built-in NOX components called *Discovery*. The *Route Tracker* reacts by preparing the list of affected flows according to its record of installed route entries. For each of these affected flows, the *Routing Engine* will start to find alternative routes while making use of the working segment of the current (broken) route as much as possible. It will do this by considering the node closest to the destination with the end node of the failed link as the starting point for rerouting. From that node, it will attempt to find a loop-free shortest route to the destination node, if it exists. If multiple loop-free shortest routes to the destination node exist, then the least-loaded route is chosen. If no loop-free route to the destination

4. SDN-BASED EDGE-TO-EDGE REROUTING

node exists, then it considers the node located just before the currently considered node in the route as the start node and attempts to find a loop-free shortest route from that node to the destination node again.

The fault-tolerant mechanism is illustrated using the Figure 4.1 . We assume that there are two flows f_1 and f_2 that are initially installed, with flow f_1 from Host 1 to Host 2 using the route A–B–E–F and flow f_2 from Host 2 to Host 1 using the route F–E–C–A as shown in Figure 4.4. Suppose that link E–F fails. For flow f_1 , the first node to be considered as the start node for rerouting is node E which has a route to node F through E–D–F. With this alternative route, flow f_1 will be taking the new route A–B–E–D–F. For flow f_2 , the first node to be considered as the start node is node F. Note that node F has several shortest routes to A, namely, F–D–B–A and F–D–C–A. The algorithm will take the least-loaded route among these two alternate routes. For simplicity, assume that route F–D–C–A route is chosen for flow f_2 . The old and new routes are shown in Figure 4.4.

Figure 4.4: Illustration of route selection when link F-E fails

Let us now consider the scenario where another link failure happens after rerouting. Assume that now link B–E fails. Flow f_2 is not affected by the link failures. Flow f_1 , on the other hand, is affected. The first node to be considered as the start node to

reroute flow f_1 is B. The shortest route from B to F is B-D-F. With this alternate route, flow f_1 will be rerouted to A-B-D-F. The state of the network now is as shown in Figure 4.5.

Figure 4.5: Illustration of route selection when link B-E fails

4.3 Performance Evaluation

A testbed using six OpenFlow switches, two hosts and a controller connected as the topology as shown in Figure 4.1 is used. All the OpenFlow switches are software-based OpenFlow switches, with OpenFlow reference implementation version 1.0, running on Linux server machines with 1 Gigabit Ethernet interfaces [105]. The controller is a Linux server machine running the NOX-Classical controller, Destiny branch. Our routing algorithm is running at the NOX controller which we call *Dynamic Routing* algorithm. The performance of this algorithm is compared with the well known multipath ECMP routing algorithm and the single shortest path routing algorithm.

Traffic pattern which are a mix of large and small flows are used in the experimentation. The traffic generator has been used to generate various traffic patterns by adjusting the packet size as 64, 512 and 1500 bytes under 100% network load. The

4. SDN-BASED EDGE-TO-EDGE REROUTING

performance metrics used are load distribution, throughput, and link utilization. For large flows, the size of 100KB and more and for small flows, the size less than 100KB are considered.

Figure 4.6: Comparison of our proposed dynamic routing algorithm with ECMP and single path routing in terms of load distribution [6]

The plots in Figure 4.6 show the load distribution at the switches in the OpenFlow testbed. It can be observed that switches A and F are overloaded for all algorithms because they are ingress and egress points. The focus will be on traffic load distribution in the intermediate switches, B, C, D, and E. Using our Dynamic Routing algorithm, the load gets distributed more evenly across the other switches, B, C, D and E. However, ECMP overloads switches B and E while single path routing overloads the switch B and D whereas switches C and E are underutilized for the traffic patterns considered. The proposed algorithm reduces the stress of core switches upto 50% when compared to single path routing and upto 40% when compared to ECMP.

Figure 4.7 plots link utilization using our Dynamic Routing algorithm for the traffic patterns generated from the traffic generator under 100% network load, the link utilization averages to 80% which is equal to 800 Mbps whereas for the single path routing, the link utilization averages to 70% which is equal to 700 Mbps and for ECMP, the link utilization averages to 72% which is equal to 720 Mbps.

Figure 4.7: Comparison of our proposed dynamic routing algorithm with ECMP and single path routing in terms of link utilization [6]

Figure 4.8: Link utilization using the proposed approach before and after a link failure [6]

For testing the performance of the fault tolerance mechanism, two flows on the topology as shown in Figure 4.1. and one link failure are considered. The first flow, f_1 , is traversing through route A–B–D–F while the second flow, f_2 , is traversing through route A–C–E–F. Figure 4.8 depicts the load distribution before and after the failure of link B–D. We note that failure of link B–D, causes f_1 to be rerouted to A–B–E–F, thus increasing the utilization of links B–E and E–F and reducing the utilization of links B–D and D–F.

4.4 Simulation Results

In this section we evaluate the performance of the proposed algorithm by simulation on a Fat tree topology and a mesh topology. The simulation was performed using Omnet++ 4.0, an open-source, extensible, modular, component-based C++ simulation library and framework, with an Eclipse-based IDE and graphical runtime environment [104]. The traffic patterns considered are as below.

- Stride 1 : in this pattern, each $host_i$ sends packets to $host_{(i+1) \bmod 16}$
- Stride 2 : in this pattern, each $host_i$ sends packet to $host_{(i+8) \bmod 16}$
- Bijective : here every host randomly selects a destination host and each host is a source for only one traffic flow and destination for only one traffic flow.

For each of the traffic patterns considered, each host sends 10000 packets each of size 1500 bytes to the destination host. Each switch in the three tier fat tree architecture considered is a four port switch and there are 16 hosts in total. The links are assumed to be 1 Gbps links. Figure 4.9 shows the simulation setup using Omnetpp 4.0 simulator [104].

4.4.1 Results for Fat Tree Topology

In this experiment, we consider Fat tree topology to evaluate the performance of the proposed algorithm. Its performance is compared with the single path algorithm and ECMP algorithm.

Figure 4.9: Simulation setup using Omnet++ simulator(fat tree topology)

4.4.1.1 Load Balancing

Figure 4.10 shows the load distribution across the core switches for the stride2 traffic pattern. It can be observed that using our proposed dynamic algorithm, the load gets distributed more evenly across the core switches thereby not overloading any particular core switch. However ECMP based routing performs poorly for the traffic patterns considered. Single path algorithm resulted in uneven load distribution for the stride pattern.

4.4.2 Mesh Topology

In this experiment, we consider a mesh topology as shown in 4.11 with 16 switches and 24 hosts.

4. SDN-BASED EDGE-TO-EDGE REROUTING

Figure 4.10: Load distribution

Figure 4.11: Simulation setup using Omnet++ simulator(Mesh topology)

4.4.2.1 Load Balancing

Figure 4.12: Mesh topology - load balancing across core switches

Figure 4.12 shows load distribution across the core switches for the stride1, stride2 and bijective traffic patterns with our dynamic algorithm. It can be observed that the load gets distributed more evenly across the core switches thereby not overloading any particular core switch.

4.4.2.2 Fault Tolerance

Figure 4.13 compares the scenario of a link failure with that of the normal working condition for stride 2 pattern. It is clear that during the normal operation of the network, the load is more evenly distributed across the core switches. The figure also depicts the load distribution for the failure of the link connecting core1 and core4. Here the packets reaching core4 are affected and thus packets arriving at core4 are less compared to the other switches. Further, we can observe that the packets are more evenly re distributed among the other core switches i.e. core1, core2 and core3.

Figure 4.13: Mesh topology - traffic distribution upon a link failure

4.5 Summary

In this chapter, an OpenFlow-based re-routing control framework, rerouting algorithm and prototype implementation have been described. The framework has been developed to address the problem of managing different flows in Data Center networks in the event of congestion and faults. The framework enables adaptive re-routing of flows to efficiently spread the load in accordance with the changing load conditions. We also considered the case of link failures wherein the traffic on the failed paths are evenly redistributed across the working paths. The framework has been implemented in the OpenFlow testbed and demonstrated the effectiveness of the proposed re-routing mechanism has been demonstrated in comparison with other methods. The performance of the proposed mechanism has been studied on Fat tree and a mesh topology using simulation results.

Chapter 5

SDN-based Data Replication

This chapter studies how our SDN-based load balancing framework can be extended to support the data replication in Data Centers. The SDN-based data replication framework with the routing method, functional modules, prototype evaluation are provided in this chapter. Data replication is an effective approach [60, 61, 62, 63] for achieving high data availability and durability in Data Centers. Data replication is a method in which copies of data are placed at two or more storage nodes attached to different racks in a Data Center. It ensures that at least one copy of data is available for continuous operation in the event of a rack switch failure or rack power failure or cooling problems. However, maintaining consistency between the replicas in a synchronised manner is a complicated task.

Companies such as IBM, EMC, Veritas, etc., provide their own proprietary solutions for remote mirroring [106, 107, 108, 109]. Existing data replication techniques [107, 109] are primarily storage-based in which disk contents are synchronized in an incremental timely manner. Different from storage based replication, we explore the network based replication at Data Center. Network-based replication uses a switch or appliance attached to a network in the data path to handle replication. The key limiting factors of network based data replication are network bandwidth, latency [110] and host stack modification [111, 112] apart from the requirement of vendor specific special hardware.

The adoption of network based replication is now possible with the use of SDN technologies such as OpenFlow which redirects the packets in the network and do not require modification of the end hosts stack. We develop an SDN based system which will

5. SDN-BASED DATA REPLICATION

perform network based replication in the Data Centers. We develop OpenFlow NOX controller service that determines the least loaded route based on the current network state to the primary destination of a flow as well as the replica servers. Different from conventional flow routing, the data replication requires routing of many flows from the same client to the primary server and each of the replication servers. The routing rules in the controller service ensure that the packets reach the primary and replica servers without causing congestion in the network. We evaluate the performance of our implementation with the traditional iSCSI (Internet Small Computer System Interface) based implementation for network-based service replication. The research work presented in this chapter has been published in [7].

5.1 Proposed SDN-based Data Replication Framework

Our SDN-based replication framework is shown in Figure 5.1. The network is comprised of switches interconnected in three layers: Top of Rack (ToR), aggregate, and core layers as shown in the figure. The architecture includes the replication host (which initiates the replication operation), the primary site where the primary copy of data is stored in a server and backup site where the backup copy of the data is stored in a server. We generate two flows for a replication operation - one flow between the replication host and primary server and the other flow between the replication host and backup server.

5.1.1 Dynamic Routing Method

With the objective of balancing the load, the proposed dynamic routing method selects paths based on load information. It uses a set of shortest-hop paths as candidate paths. We define the cost of a path as the sum of utilization of links along the paths. We then choose the minimum cost path among the candidate paths. In case where two or more paths with the minimum cost exists, the path with minimum load (equivalent to maximum residual capacity) is chosen. The load of a path is defined as the maximum load on any of the links traversed by the path.

5.1.1.1 Functional Modules

The SDN based Data replication Testbed topology is shown in Figure 5.2. The networks, namely, control network (CN) and network under test (NUT) are setup to

Figure 5.1: SDN based data replication architecture

deploy SDN-based remote replication. The CN is the network used for management of the OpenFlow switches. The NUT is used as the OpenFlow data path where the storage network is deployed and tested. Our dynamic load-balancing algorithm is deployed at the NOX controller [51] which is used as the base controller for the HP series OpenFlow switches. The system components and functional modules in the proposed data replication framework are described below.

To test the performance of data replication using our dynamic routing algorithm, we use two storage systems namely Primary Storage server and Replica Storage server from different network paths of the NUT to the Data replication host. We configured the data replication host as an iSCSI initiator and the two storage servers connected to different network paths as iSCSI targets. Replica Storage server acts as a backing store for the Primary Storage Server. We use IOmeter [113] as the work load generator.

The functional modules namely, (i) Monitor and (ii) Routing module are the two

5. SDN-BASED DATA REPLICATION

Figure 5.2: SDN based data replication testbed

key modules in the Centralized NOX controller. In order to achieve dynamic load balancing, the packets are routed along the least loaded path. The packet counts and flow count information from the Monitor module together with capacity of each link along the paths are used to compute the least loaded path by the Routing Module.

The functionalities of each module in our dynamic routing framework are described in the following sections.

Monitor Module: This component is responsible for collecting statistics from the OpenFlow switches that will be used by the Routing Component to compute and compare the load on various links. These statistics are collected at fixed intervals and stored in NOX Controller as snapshot objects. Routing module can access these snapshot objects directly to calculate the least loaded path.

Routing Module: The main functionality of this module is to route the flows along the least loaded path. Based on the most recent load statistics collected by the

5. SDN-BASED DATA REPLICATION

Figure 5.4: Shortest routes from replication host to backup server

utilization of a link is calculated as $\frac{L}{C}$. The cost of a route is calculated as the sum of the utilization of the links along the path. After computing $route_cost_i \forall r_i \in R$, the controller picks the route that has the minimum $route_cost_i$. If there are more than one route with the minimum $route_cost_i$, the route with maximum residual capacity will be chosen.

5.1.2 Network Under Test

We used a hardware switch [17] (HP ProCurve E5412-96G zl switch with K.15.05.5001 firmware) for NUT to measure the SDN-based data replication performance. We have configured the switch to support the FAT tree Data Center architecture. To achieve this, we have assigned groups of 1GigE ports to 7 sets of VLANs controlled by a single instance of our forwarding algorithm running on top of the NOX OpenFlow controller. We used VLAN numbers 701 to 707 to emulate the switches of the FAT Tree topology. VLAN 701 represents the core switch of the Data Center. VLAN 702 and 703 represent the aggregate switches. VLAN 704, 705, 706, and 707 represent the ToR switches where the Data Center storage servers are connected as shown in Figure 5.2.

The individual components of NUT are briefly described below:

1. Primary Storage Server

The Primary Storage Server represents the primary storage site for replication. It acts as iSCSI target 1. It is connected to an external RAID5 storage system with 750GB capacity. The primary server is connected to TOR switch in the Data Center.

2. Replica Storage Server

The Replica Storage Server(Backup server) represents the secondary storage site for replication. It acts as iSCSI target 2. It has a direct attached storage. This is the backup store of Primary Storage Server.

3. Replication Host

This host is the data replication host with iSCSI initiator configuration. This host is connected to both CN and NUT. After starting the iSCSI initiator service, two new storage devices (e.g. /dev/sdb and /dev/sdc) will be detected by this host.

5.2 Performance Study

In order to test the data replication performance, we run a benchmark called *IOmeter*[113], which is an industry standard I/O benchmark tool. IOmeter can generate workloads of various characteristics including request size, read/write ratio, queue size and maximum number of outstanding requests. The IOmeter creates one worker thread for each CPU core. For each block size we ran several read, write, and read/write tests. We set up a testbed using a HP OpenFlow switch, three hosts and an OpenFlow controller, with the topology as shown in Figure 5.2. The hardware configurations of our experiments are shown in Table 5.1 and Table 5.2.

5.2.1 Throughput

We evaluate the effectiveness of our proposed SDN-based dynamic routing mechanism in terms of throughput achieved by comparing it with the traditional iSCSI based approach (without SDN) for read and write operations. We use synchronous operations wherein the operations on primary and backup copies are performed simultaneously.

5. SDN-BASED DATA REPLICATION

Table 5.1: Switch configuration

Manufacturer	Hewlett Packard
Model	HP E5412-96G zl Switch
Firmware	K.15.05.5001
# of GigE Ports	96

Table 5.2: Replication host, Primary server, Backup server, and NOX configuration

	Replication Host	Primary server	Backup server	NOX Controller
Operating System	Fedora 16 (Linux 3.3.8-1.fc16.i686)	Fedora 16 (Linux 3.4.4-4.fc.16.i686)	Fedora 16 (Linux Linux 3.4.7-1.fc16.i686)	Ubuntu 10.04 (Lucid) Linux 2.6.32.33-server
Memory	3.0GiB	3.4GiB	2.2GiB	1.5GiB
Processor	AMD Athlon 64 X2 Dual Core 4200+	Intel Xeon CPU W3530 @ 2.8GHz	Intel Xeon CPU E5630 @ 2.53Ghz	Intel Core2 CPU 6400 @ 2.13GHz
# of Processor	2 Cores	8 Cores	8 Cores	2 Cores

The throughput achieved by our method and iSCSI method for various request sizes are shown in Figure 5.5. We can observe that our SDN-based method achieves higher throughput demonstrating the effectiveness of dynamic routing. The figures shows that our SDN-based replication technique achieves performance gain of up to 20% when compared to iSCSI method.

5.2.2 Performance for Different I/O Queue Size

The storage server's queue length is an important parameter that affects small I/O and getting maximum performance out of storage subsystem. The queue length has an impact on the number of input/Output operations per second (IOPS) generated. So, we measure the performance with varying storage server queue lengths. The impact of the queue length on synchronous replication between Primary server and Backup server is shown in Figure 5.6. From the figure, we can observe that throughput increases with the queue length until the queue length equals to 8. This is because the maximum number of parallel operations that can be handled by the storage system used is limited by 8.

Figure 5.5: Performance of SDN vs. iSCSI(without SDN) based schemes [7]

Figure 5.6: Throughput for different I/O queue size [7]

5.2.3 Load Distribution

Figure 5.7 shows the load distribution at the VLANs in the OpenFlow testbed. It can be observed that using our algorithm, the load gets distributed more evenly across the

5. SDN-BASED DATA REPLICATION

Figure 5.7: Load Distribution across aggregate switches

aggregate switches, i.e. across VLANs 702 and 703. This shows the effectiveness of our algorithm in balancing the load. Our replication method reduces the stress of core switches up to 25% when compared to iSCSI method.

5.2.4 Throughput at Primary Server

The snapshot taken at the primary server is shown in Figure 5.8. It shows the throughput measured in units of million Bytes per second (MiB/s) for the data bytes sent and received. We can observe that the throughput of about 600 Mbps for 1Gbps ports can be achieved.

5.3 Summary

In this chapter, the problem of load balancing for data replication in Data Centers was studied. A dynamic routing method was developed for choosing the paths for the flows to the primary and replication servers used for replication. This routing method is based on dynamic load information collected in the proposed SDN-based architecture. An OpenFlow based testbed was built to carry out the performance study.

Figure 5.8: Screenshot from the primary server during replication [7]

The experimental results demonstrated the effectiveness of the proposed SDN-based scheme when compared to the traditional iSCSI-based scheme .

Chapter 6

Flow Cache Design and Maintenance

Chapter 6 describes the proposed flow cache architecture with dynamic indexing scheme and replacement strategies to address the problems due to limited switch size and mice flooding. Though SDN offers benefits like flexible routing management and fine grained traffic control, it has a drawback of limited switch memory due to high cost and energy consumption leading to unfair eviction of elephant flows as we discussed in chapter 2. Mice flooding in a Data Center network will cause the elephant flows to be evicted prematurely from the flow table, particularly when a burst of many mice flow packets arrive at a switch between elephant flow packet arrivals [65, 87]. Flow table misses cause additional delay due to round-trip time to the controller and in addition, an extra *packet-in* event is generated and sent to the controller, incurring additional load on the controller.

To handle this issue, we propose a flow cache layer in between the SDN switch and the controller. For a scalable solution, caches can be used with each flow cache storing records for a number of switches. Thus a cache is comprised of a number of flow caches each corresponding to a flow table of a switch. When required, a switch first consults the flow cache instead of directly contacting the controller. Our work deals with dynamically growing/shrinking buckets according to the dynamic network traffic. The proposed differentiated caching framework has several attractive features. The cache architecture and hash function used are simple, thus enabling easy of implementation. The dynamically growing/shrinking feature of the flow cache enables better sharing of

6. FLOW CACHE DESIGN AND MAINTENANCE

cache memory by different flow caches. The differentiated index approach improves fairness for elephant flows reducing the number of their evictions. Also, our placement, lookup, and replacement mechanisms ensure fast flow processing with high hit rate. The research work presented in this chapter has been published in [8].

6.1 Caching Architecture and Maintenance

The proposed flow cache architecture and various maintenance operations are described in this section. As shown in Figure 6.1, the cache is shared by more than one switch. But, a switch is mapped to only one cache so that the need for coordination between the cache modules in a large network is avoided. For simplicity, we will only use one flow cache in our discussion in the following sections.

Figure 6.1: Cache architecture

In the proposed architecture, a cache corresponding to a switch has dynamically growing and shrinking number of blocks (or buckets) with associated indices. The size

of the index depends on the current number of buckets. The index value is obtained by hashing the relevant fields of a flow, called dynamic-index hashing which is similar to extendible hashing [114]. We propose a differentiated flow cache mechanism wherein we associate different indices for elephant and mice flows, and also for the buckets to ensure fair treatment of elephant flows. The use of multiple buckets in the cache allows us to keep the number of entries in a bucket that need to be searched as small as possible. The search (or lookup) operation with fewer cache entries helps reduce the lookup time when realized with SRAM. In the proposed framework, we used a localized least recently used (LLRU) policy as the cache entry replacement strategy, which is shown to be more effective than other strategies by our experiments.

The interactions between OpenFlow switch, cache and the controller are shown in Figure 6.2. When there is no space in the flow table or due to timeout’s mechanism, an entry in the flow table may be evicted. Such evicted entries will be stored in the associated flow cache. When a packet arrives at an OpenFlow-enabled switch, a lookup is performed first on the OpenFlow switch’s flow table. The pseudo code for the Lookup operation is shown in Algorithm 2. If there is no entry in the flow table, called as a *flow table miss*, triggers the lookup of the flow cache; if there is a match in the flow cache, the associated action for packet forwarding will be executed; otherwise a *packet_in* event is sent to the controller leading to an insertion of an entry in the flow table. If a cache bucket is full and the maximum number of buckets has already been used up, new flow entries cause a cache miss leading to the eviction of existing flows.

We now discuss the communication overhead incurred. As we mentioned earlier, when the flow entry is not in the flow table at a switch, look up operation is performed in the cache module which is distributed and located close to the switch. When there is a cache hit, there is no message sent to the controller, only a few messages will be exchanged between the switch and the cache, but there is no need to send the messages for installing the rules at other switches on the route. We note that when there is no cache used, *packet_in* messages are sent from the switch to the controller which in turn sends many messages to install the forwarding rules at all the switches on the chosen route. This would incur a heavy traffic and computational load on the controller. On the other hand, the use of cache and a high cache hit requires only a small number of messages exchanged.

6. FLOW CACHE DESIGN AND MAINTENANCE

Algorithm 2 Algorithm for *Lookup* operation at Flow Cache

```

1: Read a Packet that arrives at the OpenFlow switch ;
2: Check the OpenFlow switch's flow table for matched entry;
3: if found then
4:   Perform action for packet forwarding;
5: else
6:   check the flow cache;
7:   if cache_hit then
8:     Perform action for packet forwarding;
9:   else
10:    Send packet_in message to controller;
11:   end if
12: end if

```

Figure 6.2: Interactions between the controller, cache and flow table

In a dynamic network environment, where the controller performs dynamic re-routing based on the real-time traffic load, the path of the large flow could have been changed by the controller to avoid congestion [6]. In that scenario, the controller will send *flow modification* messages to both flow cache as well as switch to update them. The insertion of the flow cache layer in the architecture results in exchange of a few messages like *flow_evict*, *packet_in* messages between cache layer and the controller as shown in Figure 6.2.

The main components of our framework are described in the following sections.

6.1.1 Elephant Flow Detection

The two main components that we designed in the NOX controller are *Monitor* and *Packet-in Tracker*. The purpose of the Monitor component is to query and collect the per-table, per-flow and per port statistics from all the connected OpenFlow switches at fixed intervals. The collected statistics are stored in memory as a snapshot object. Each snapshot is identified by a sequence number, which increments by 1 after each interval.

The elephant flows are identified using the collected flow statistics. The mechanism is explained as follows. Each flow is monitored at the first switch traversed by the flow. The statistics are collected from the switches by the controller at a fixed interval of p seconds and used to classify the large flows. It is computed as follows:

$$\psi_t = \frac{b_t - (b_{t-p})}{p} \quad (6.1)$$

where ψ_t is the estimated flow size at time t , and b_t is the total bytes of the flow received by the switch at time t . In our implementation, we set p to 5 seconds. We classify flows as elephant flows if the flow size is equal or greater than 100KB. We follow the similar approach as in Mahout [87] to use the VLAN priority code bit (PCB) bits (set as 001 for elephant and 000 for Mice) to indicate the elephant flows as shown in Figure 6.3.

The *Packet-in Tracker* component is responsible to process the *packet_in* messages that are generated after the flow cache miss. The main functionality of this component is to calculate the least loaded route and then inserts the corresponding flow entry in the flow table.

6. FLOW CACHE DESIGN AND MAINTENANCE

Switch port	MAC src	MAC dst	Eth Type	VLAN	IP Src	IP Dst	IP Port	TCP sport	TCP dport	Action
				PCB-001						

Figure 6.3: Using VLAN PCB bits to classify flows

6.1.2 Flow Cache and Dynamic-index Hashing

The flow cache is organized into two separate buckets; one for mice flows and one for elephant flows. While the maximum number of buckets in a flow cache is constrained, the number of buckets used by the elephant flows and mice flows are dynamically determined based on the requirement. We use dynamic-index hashing to map a flow to a bucket. The k -bit index associated with a flow is dynamic and the value of k depends on the number of buckets currently used for that type of flow (mice or elephant). Associated with each bucket there is an index which stores the flows (either mice or elephant, but not both) with the same index. We use the popular Secure Hash Algorithm (SHA) [115] and extract x bits (for forming the index) from the random hash value which is more likely to keep the buckets balanced. Since SHA is a strong hashing method, it is highly likely that equal number of flows will be mapped to different buckets.

A directory is used to store the index values and the pointer to the corresponding bucket. The index directory is in the form of an array with at most 2^b entries for the indices with each entry storing a bucket address. The variable b is called the maximum depth of the directory corresponding to the maximum permissible number of buckets. The index directory and 1-bit index buckets for mice and elephant flows are shown in Figure 6.4. Initially, the cache line has only two buckets termed as Bucket A with index 0 for storing mice flows and Bucket B with index 1 for storing elephant flows.

The placement of the flows into the flow cache is carried out in the following steps which are shown in Figure 6.4.

Figure 6.4: Illustration of placement of flow records

1. When a packet arrives, the hash value for the packet is calculated from its flow identifying fields using the secure hash algorithm SHA-1[115]. The advantage of using SHA hashing is that it randomly maps the hashed key values to a space of 160-bit binary numbers. The first $k-1$ bits ($k=1,2,..$) will be extracted and a 1-bit prefix is added with bit '0' used for mice flows and bit '1' for elephant flows to form a k -bit index.
2. By using the index constructed as above, the pointer to the bucket is obtained from the index directory.
3. Lookup for the matched entry in the chosen bucket is carried out using the full 160-bit hashed value.
4. If there is no match in step-3, a new entry is made in the bucket, subject to space availability. If there is space constraint, a new bucket will be created using the bucket expansion mechanism as explained in the next section.

6.1.3 Overflow Handling- Bucket Expansion

If a bucket overflows, the bucket's size is doubled with the new index using an extra bit. For example, if 1-bit index is used originally, after expansion 2-bit index will be used. As a result, the entries in an original bucket (say, with index 0) will be distributed among two buckets (with index 00 and index 01). The index directory will also be doubled, i.e., if it originally contains 2^k index values, it will now contain 2^{k+1} entries, implying that the depth increases to $k+1$. We note that these $k+1$ bits are formed from a random hash value as explained before. As a result, it is likely that the new buckets are more balanced. If the 0-indexed mice bucket exceeds its capacity (1024, for example), it will be expanded into 2-bit indexed buckets: 00 and 01 thus increasing its maximum capacity to 2048. If there is a need for further expansion, it will be expanded to 3-bit indexed buckets: 000, 001, 010, and 011. A similar procedure is used for elephant buckets with prefix 1.

We note that the number of index bits for mice flows and elephant flows need not be the same. Since the number of mice flows is much more when compared to the number of elephant flows, it is highly likely that the index size is larger for mice flows than elephant flows.

Bucket expansion is illustrated in Figure 6.5. Here we assume a bucket size of 4000. When the 4001st mice flow arrives, bucket A overflows and a new bucket A' is used and the 4000 entries (including the new one) are distributed over the buckets A and A' each of which now uses a 2-bit index. Similarly, if any bucket with a 2-bit index is full, a new bucket is added when a new flow arrives. The buckets will now have 3-bit indices.

6.1.4 Cache Replacement Strategy

The total number of concurrent flows in the system is much larger than the cache size. hence, there is the need for a replacement policy. Since we maintain different buckets for mice and elephant flows, our replacement policy is designed in such a way that a new mice flow can evict only a mice flow and an elephant flow can evict only an elephant flow. We use localized least recently used (LLRU) policy to evict the flows wherein, the least recently used flow in the local bucket is evicted. By doing so, we preserve the mapping of a flow to the bucket and ensure fast processing. The least recently used flow

Figure 6.5: Illustration of bucket expansion

entry will be evicted in the event of a cache miss and permissible number of buckets have been used up. We note that although we use localized replacement, the hit ratio is likely to be high as evident from our performance study presented in Section 6.2.

6.1.5 Bucket Shrinking

Flows with long inactive period will be deleted from the cache and it might cause a bucket to become empty. In that case, a check is made for possible bucket shrinking wherein the bucket index shrinks by 1 bit and the number of buckets is halved and flow entries are shuffled accordingly as shown in Figure 6.6. The figure shows the buckets with index '01' has a very few entries and bucket with index '00' is not full. In that case, these two buckets are managed to form a new bucket with one bit index '0'. Similarly buckets with index '10' and '11' are merged to a single bucket with index '1'. Since there are multiple elephant flows in one bucket, it is unlikely for all of them to stay inactive for a long period of time. Further, bucket shrinking occurs only if the number of entries falls below half of bucket size which is less likely to occur frequently.

6. FLOW CACHE DESIGN AND MAINTENANCE

Figure 6.6: Illustration of bucket shrinking

6.2 Performance Evaluation

We present the results of our performance study carried out using software implementation of the cache in this section. We compare the performance of our proposed method with the OpenFlow based wildcard-aware linear search table [85] and with different flow cache replacement policies such as Random and First-In-First-Out (FIFO). While Random method selects the flow to evict from the cache at random, FIFO evicts the flow which arrived first. Following the observations in [86], we generate the Data Center bursty traffic pattern which is a mix of 80% mice flows and 20% elephant flows using Ostinato traffic generator [116].

6.2.1 Performance in Terms of Cache Hit Rate

Figure 6.7 shows the cache hit rate achieved by the four cache replacement strategies: LLRU, Wild-card linear, Random and FIFO. We can observe that our proposed LLRU method performs better than OpenFlow wildcard-aware linear table [85] replacement, Random replacement and FIFO replacement method. Also note that although the number of mice flows is large, the performance of elephant flows is not affected due to the use of different buckets for mice and elephant flows.

Figure 6.7: Comparison of cache hit rate [8]

6.2.2 Impact of Cache Bucket Size

The impact of cache bucket size of 1k, 2k, 4k and 8k flow entries on the performance in terms of look-up time has been studied. Figure 6.8 shows the effect of the cache size on the lookup time with wildcard-aware linear flow table and our proposed dynamic index hashing on the new architecture. We can observe that the lookup time increases with increasing bucket size. Further, by organizing the cache into different buckets, our method achieves significant performance improvement in terms of lookup time when

6. FLOW CACHE DESIGN AND MAINTENANCE

compared to wildcard-aware linear flow table. In our simulation, it was shown that our cache design LLRU reduces the lookup time up to 50% when compared to wild-card linear method.

Figure 6.8: Impact of cache bucket size [8]

6.2.3 Performance in Terms of Lookup Time

The performance of the proposed method is compared with wildcard-aware linear table in terms of lookup time. We measure the lookup time based on simulation PC with custom implementation. The lookup time includes the time taken to carry out hashing and the corresponding bucket look up time. Whereas in the Wild card method, the look up time refers to the time for linear table walk time. In Figures 6.9, 6.10, and 6.11, we show the Cumulative Distribution Function (CDF) lookup time comparisons for 500, 1000, 1500, 3000, and 5000 entries. From the graphs, we can observe that the lookup time by our method for both mice and elephant flows are considerably smaller than that by the wildcard-aware linear method.

With the wildcard-aware linear method, there is no differentiation between mice and elephant flows, and it is highly likely that the elephant flows are in the higher

percentile region resulting in unfairness for elephant flows. This unfairness problem is reduced in our method. Also, we can observe that the lookup time for mice flows is higher than that for elephant flows. This is because, there are more entries inside a mice bucket than an elephant bucket.

Figure 6.9: Lookup time for wild-card linear method [8]

6.2.4 Bucket Expansion Overhead

In this section, the overhead due to bucket expansion is studied. We assume the number of the flows that the bucket can accommodate is 1024(1K) and the mice and elephant flows are in the proportion 80:20. When the bucket expands, some of the entries in the original bucket will be moved to a new bucket incurring the overhead which are given in detail in the Table 6.1. The bucket expansion overhead values are plotted for three different values of the number of flows in Figure 6.12. We note that when the number of flows exceeds 1024, we need a 2-bit index and when it exceeds 2048 we need a 3-bit index. As such, for the case of 5120 entries (total), the mice flow buckets need to be expanded from 1-bit to 2-bit index then to 3-bit index; and for elephant flow buckets to expand from 1-bit to 2-bit index. For the case of 3000 entries (total) the mice flow

6. FLOW CACHE DESIGN AND MAINTENANCE

Figure 6.10: Lookup time for mice flows [8]

Table 6.1: Bucket expansion overhead

No. of Flows	Bucket Expansion					
	No. of Buckets		Bucket index(No. of. Flows)		Bucket Expansion	
	Mice	Elephant	Mice	Elephant	Mice	Elephant
1500	2	1	00(1024), 01(176)	1(300)	Y	N
3000	3	1	000(1024), 001(1024), 01(352)	1(600)	Y	N
5120	4	2	000(1024), 001(1024), 010(1024), 011(1018)	10(1024), 11(6)	Y	Y

Figure 6.11: Lookup time for elephant flows [8]

bucket needs to expand from 1-bit to 2-bit index and then to 3-bit index. For the case of 1500 entries (total) the mice flow bucket needs to expand from 1-bit to 2-bit index. We can observe that the overhead is small but it increases with the increasing number of index bits. At the same time, we note that such an expansion occurs less frequently compared to the lookup operation.

6.3 Summary

In this chapter, we addressed the problem of improving fairness for elephant flows in a Data Center as they suffer from frequent eviction because of the flooding of mice flows. We adopted a differentiated approach and proposed a flow cache architecture with dynamic-index hashing for placing the flow records onto various buckets. The simulation results show that the proposed localized LRU-based replacement strategy, gives better performance than other methods. Overall, the differential flow cache architecture provides a simple and effective means to address the overload issue on the controller due to flow table exhaustion at the OpenFlow switch.

Figure 6.12: Bucket expansion overhead [8]

Chapter 7

Virtual Machine Placement with Two path Routing

Chapter 7 studies how benefits can be achieved by VM placement and traffic distribution in terms of congestion and traffic protection. Server virtualization adopted in Data Centers enable flexible and efficient resource utilization. Virtual machine (VM) placement which defines a VM-to-server mapping, is an important problem in Data Centers. An intelligent placement of VMs will bring in several benefits such as reduced congestion, fault tolerance and energy efficiency. With the increasing trend of traffic pattern shifting towards “east-west“, large amount of traffic is routed between VMs within a Data Center. Therefore, it is essential that the impact of VM placement on network traffic be given due consideration. An improper placement of VMs might result in traffic traversing longer paths and/or bottle-necked links. In the literature, while some works [39, 40, 41] studied the VM placement problem considering only the server resources without considering the impact on the networking resources, some [46, 94] studied the VM placement problem taking into consideration of the network resources. While such works in the literature address the problems of VM placement and traffic routing with the objectives of improved resource utilization and reduced congestion, they do not study these problems from the perspective of fault tolerance. Different from other works in the literature, in this chapter, we address these problems with the objective of reduced congestion while guaranteeing a certain protection grade for the traffic in the event of failures.

Link failures in Data Centers are critical leading to severe consequences. A possible

7. VIRTUAL MACHINE PLACEMENT WITH TWO PATH ROUTING

way to provide fault tolerance is to reserve additional network resources, called backup bandwidth which is readily available in the event of failures. By doing so, we can fully protect the traffic, i.e., upon a link failure, the same amount of bandwidth will be made available for the failed traffic as was available for the working traffic before the failure. But, it is expensive to reserve such extra backup resources. On the other hand, for a cost-effective solution, traffic can be protected partially wherein reduced bandwidth is guaranteed to be available in the event of failures. We note that, although the performance might degrade with partial protection due to smaller bandwidth, it could be acceptable for most applications. Further, more bandwidth than the minimum guaranteed bandwidth might be available upon failures. Also, the downtime is small as the failures in a Data Center can be repaired quickly when compared to the failures in wide area networks (where cables are usually laid under the ground and sea). Thus, partial protection is an attractive and cost-effective solution for providing reliable services in Data Centers.

In this chapter, we investigate the problem of VM placement and traffic routing in Data Centers so as to reduce congestion while satisfying server and network resource requirements and guaranteeing a certain protection grade. The fraction of rate (or bandwidth) guaranteed to be available upon a failure defines the protection grade. We develop a greedy algorithm to place VMs for flow demands with specific computing, memory, and bandwidth resource requirements. In the proposed traffic splitting approach, a flow is split and routed over two link-disjoint paths with appropriate rates based on the protection grade requirements. Unlike the multipath routing algorithm ECMP which chooses a path based on hashing, the proposed algorithm chooses a path based on current utilization of the links and also the protection requirements. We also provide a discussion on how protection grade can be improved by using more paths for traffic routing. The research work presented in this chapter has been published in [9].

7.1 VM Placement and Routing Framework

This section provides an overview of the proposed framework for VM placement and traffic routing. The framework with different components is shown in Figure 7.1. The virtual machines in physical server communicate to physical network interfaces through a virtual switch inside the hypervisor. The network is a 3-tier multi-rooted tree with

Figure 7.1: VM placement framework.

edge, aggregate and core switches. The aggregate and edge switches are grouped to form a pod. The hypervisor on a host allows a VM to be deployed on the host having sufficient resources. A VM communicates through the physical network adapter of the hosting server. The tasks of VM management and routing decisions for the VM traffic are carried out by the controller. The functionalities of each component are described in the following sections.

7.1.1 Hypervisor

Associated with each server, there is a hypervisor. The main function of a hypervisor is to host multiple VMs on the associated server. Each VM is assigned a destination hypervisor with the condition that there are sufficient resources available to accommodate it. Each VM has a set of parameters specifying the required computing, memory, and bandwidth resources. A VM sends and receives traffic through the network adapter of the server in which it is hosted. The details of the VMs on a server network interfaces, and the ToR switches connected to them are maintained by the hypervisor.

7. VIRTUAL MACHINE PLACEMENT WITH TWO PATH ROUTING

7.1.2 Controller

The controller is the central manager which carries out key tasks of VM management and routing for VM traffic. As shown in Figure 7.1, the controller has three logical modules, 1) Monitor 2) VM placement and route selection and 3) VM and route deployment. The Monitor collects the resource availability status from hypervisors and network link status (bandwidth availability and failure notification) from switches. It also receives flow demands and their resource requirements. This module passes the server and network state information to the module *VM placement and route selection*.

The VM Placement and Route Selection module chooses servers for the VMs and the routes for the communicating VMs based on the current state information received from the Monitor module. It uses an algorithm to select servers and routes satisfying the server resource and network resource requirements while making better use of the resources. It then passes the details of servers chosen for VMs and paths with the associated rates to the module *VM and route deployment*. In our work, we aim to reduce the network traffic load, followed by the number of servers used and server resource utilization, while guaranteeing the specified protection grade requirements. Our algorithm and other possible algorithms that can be used are listed below.

1. GVMTPR algorithm: This is our proposed algorithm. It adopts a greedy approach to choose the hypervisor for each VM to achieve best possible utilization of resources. For a flow, it chooses two paths and determines the fraction of traffic to be assigned to each of the two paths to meet the protection grade requirements. The details of the algorithm will be described in the following sections.
2. Random Selection: This algorithm chooses a server (hypervisor) randomly among all servers that can satisfy the VMs resource requirements. It routes traffic over a shortest-hop path.
3. First Fit Selection: This algorithm chooses the first server (with the smallest identifier) among all possible servers that can satisfy the VMs resource requirements. It routes traffic over a shortest-hop path.

The *VM and Route Deployment module* takes the responsibility of placing VMs on the chosen servers and installing the route information on the network switches. This module has two components: (i) VM deployment and (ii) Route deployment.

The *VM deployment* component sends the details of chosen servers for VMs to the hypervisors to enable placement of VMs at the servers. It provides the details of CPU, memory, ingress/egress bandwidth requirements, and the fraction of traffic to be assigned to flows. The *Route* component is responsible for installing routing/forwarding information at the network switches along the chosen paths.

7.1.3 Implementation Issues:

The proposed work can be implemented with an SDN control framework, for example, the popular openflow protocol [50]. The Openflow NoX controller [51] carries out the functionalities of routing. It also installs the flow forwarding information at various switches. Similarly the flow rates can be controlled at the switches with the latest OpenFlow v1.3 protocol. In the event of errors, these rates can be adjusted. It is also possible to control the rates at the source by using known methods such as leaky-bucket mechanisms. In this case, Upon receiving failure notification messages, the controller will instruct the hypervisors to direct the traffic through only one path, i.e., the working path.

7.2 Problem Definition and Formulation

We consider a set of n servers $S = \{S_1, S_2 \dots S_n\}$ and a set of m VM types $V = \{VT_1, VT_2 \dots VT_m\}$. The resources of a server host S_i is represented as $S_i = \{S_com_i, S_mem_i, S_bw_i\}$, where S_com_i is the computing capacity, S_mem_i is the capacity of memory, and S_bw_i is the bandwidth, i.e., ingress/egress capacity. A VM flow (or simply a flow) originates at a VM and ends at another VM. A job's resource requirement is specified as a vector $J = \{x_1, x_2 \dots x_m\}$, where x_k is the number of VMs of type VT_k . The resource requirement of VM type VT_i is represented as $VT_i = \{VT_com_i, VT_mem_i, VT_bw_i\}$, where VT_com_i is the required computing resource, VT_mem_i is the required memory resource, and VT_bw_i is the required bandwidth. The protection grade γ required is also given. We note that different applications have different kinds of CPU and memory requirements. We also note that in a Data Center there is a mix of traffic flows requiring different amounts of bandwidth [20, 117, 118]. Therefore, consideration of different types of VMs in our work has practical significance.

7. VIRTUAL MACHINE PLACEMENT WITH TWO PATH ROUTING

We study the VM placement problem with two-path traffic routing to reduce the occurrence of congestion while satisfying a certain protection level. The problem is to find a mapping between VMs and hosts that satisfies the resource requirements of VMs while trying to minimize the network congestion and providing protection guarantee of grade γ ($0 \leq \gamma \leq 1$), the fraction of bandwidth guaranteed to be available for a flow upon a link (or port) failure. We use maximum load on any link as a measure of congestion. We represent a Data Center network as a directed graph $G = (V, E)$, in which each node $v \in V$ corresponds to a network switch or host (physical server) and each directed edge $(u, v) \in E$ corresponds to a physical link between the switches or between the switches and the hosts. Each link (u, v) has a nonnegative capacity $c(u, v) \geq 0$. We implicitly assume that $c(u, v) = 0$ for $(u, v) \notin E$. Each link carries one or more flows. On link $(u, v) \in E$, we reserve bandwidth $b_{i,u,v}$ for flow i . For each job, we need to find host machines to support its physical resource requirements and a set of paths in G to route its traffic among VMs. Each path traverses a set of links that inter-connect switches and host machines. In our work, we consider two-path routing wherein the traffic is mapped to two paths with a certain traffic proportion so as to minimize network congestion while providing a specified protection grade. As stated earlier protection grade is defined as a fraction of bandwidth guaranteed to be available for the flow upon a link failure. If flow demand i with the bandwidth requirement of B_i units is split and sent across two link disjoint paths at the rate of $b_{i,1}$ and $b_{i,2}$ units (where $B_i = b_{i,1} + b_{i,2}$) respectively, then upon any link failure $\min(b_{i,1}, b_{i,2})$ units of bandwidth is guaranteed to be available for flow demand i and the protection grade is given by

$$\frac{\min(b_{i,1}, b_{i,2})}{b_{i,1} + b_{i,2}} \quad (7.1)$$

In this chapter we study the case of two path routing for the reasons of simple management and reduced impact of traffic splitting. It is possible to increase the protection grade further by distributing traffic to more than two paths. In general, if b units of bandwidth is split across n link-disjoint paths with b_{max} is the maximum bandwidth used among the n paths, then the minimum protection grade guaranteed is $[b - b_{max}]/b$ which happens when a link on the path with b_{max} units has failed. As the traffic is split across more paths, the implementation complexity increases. Further, we

can provide 100% protection guarantee using spare resources on a backup path with the reservation of b_{max} amount of bandwidth at the expense of excessive spare resources.

Below, we formulate the problem of VM placement with traffic routing on two paths. We present the objective function followed by the constraints that need to be satisfied for placement of virtual machines with traffic routing on two paths with the specified protection guarantee. The VM placement problem is complex and variants of this problem considering network traffic has been shown to be NP hard[46]. The consideration of protection guarantee makes it more complex. Therefore, in this section we only develop problem formulation for better description and understanding of the complexity of the problem. In the next section, we present fast heuristic algorithm developed by us to solve the problem.

We use the following notations.

Notations:

$P_{i,j} \in 0, 1$: indicates if VM i is placed at server host j .

B_i : bandwidth of flow i .

$b_{i,1}$: bandwidth of flow i on path 1.

$b_{i,2}$: bandwidth of flow i on path 2.

s_i : source switch of flow i , i.e., the ToR switch to which the server hosting the source VM is connected.

d_i : destination switch of flow i , i.e., the ToR switch to which the server hosting the destination VM is connected.

$x_{i,1,u,v} \in \{0, 1\}$: indicates if flow i on path 1 traverses link (u, v) .

$x_{i,2,u,v} \in \{0, 1\}$: indicates if flow i on path 2 traverses link (u, v) .

$b_{i,u,v}$: bandwidth of flow i on link (u, v) .

b_min_i : $\min(b_{i,1}, b_{i,2})$ for flow i .

$l_{u,v}$: total bandwidth used (load) on link (u, v) .

f_max : maximum load on any link.

Objective:

Our problem objective is to minimize the maximum load (f_max) on any link while providing protection grade of at least γ for each demand. We present the objective function below followed by the protection constraint and other constraints related to VM placement and two path routing.

7. VIRTUAL MACHINE PLACEMENT WITH TWO PATH ROUTING

Minimize f_max

Constraints:

1. The maximum load f_max is defined by

$$f_max \geq l_{u,v} \quad \forall u, v \in V \quad (7.2)$$

2. Load on links: The total load on any link (u, v) is calculated as the sum of the traffic of each flow i traversing the link. It is given by

$$l_{u,v} = \sum_i b_{i,u,v}, \quad \forall u, v \in V \quad (7.3)$$

3. Capacity constraints: The total resource requirement of the VMs placed on a host should not exceed its capacity. For a given server j , the sum of the resource requirements of all VMs placed on it should be less than or equal to the total available capacity of the host. As we specified above, we consider three types of resources-computing, memory, and bandwidth.

$$\left. \begin{aligned} \sum_i V_com_i \times p_{ij} &\leq S_com_j & \forall \text{ server } j \\ \sum_i V_mem_i \times p_{ij} &\leq S_mem_j & \forall \text{ server } j \\ \sum_i V_bw_i \times p_{ij} &\leq S_bw_j & \forall \text{ server } j \end{aligned} \right\} \quad (7.4)$$

4. Placement constraint: Each virtual machine is mapped to one server and all virtual machines are placed.

$$\sum_j p_{ij} = 1 \quad \forall vm \ i \quad (7.5)$$

5. Bandwidth constraint: The aggregate bandwidth used on link (u, v) should not exceed its capacity $c(u, v)$

$$\sum_i b_{i,u,v} \leq c(u, v) \quad \forall u, v \in V \quad (7.6)$$

6. No loss constraint: For every switch u that is not the source switch s_i or the destination switch d_i of flow i , the traffic flowing into it (incoming flow bandwidth) must be equal to the traffic flowing out of it (outgoing flow bandwidth).

$$\sum_{v \in V} b_{i,v,u} - \sum_{w \in V} b_{i,u,w} = 0, \quad \forall \text{ flow } i \text{ and } \forall u \in V - \{s_i, d_i\} \quad (7.7)$$

7. Traffic splitting constraint: Traffic on flow i is split into two with bandwidth $b_{i,1}$ and $b_{i,2}$ to be mapped onto two paths.

$$b_{i,1} + b_{i,2} = B_i, \quad b_{i,1} \geq 0, \quad b_{i,2} \geq 0, \quad \forall \text{ flow } i \quad (7.8)$$

8. Traffic routing on link-disjoint paths: A link (u, v) is traversed by none or only one of the two paths of flow i . This also ensures that the two paths use disjoint sets of links.

$$\left. \begin{aligned} x_{i,1,u,v} + x_{i,2,u,v} &\leq 1, \quad \forall \text{ flow } i \text{ and } \forall u, v \in V \\ b_{i,u,v} &= x_{i,1,u,v} \times b_{i,1} + x_{i,2,u,v} \times b_{i,2} \leq B_i \quad \forall \text{ flow } i \text{ and } \forall u, v \in V \end{aligned} \right\} \quad (7.9)$$

9. Traffic splitting at the source: The total traffic rate flowing out of the source switch S_i onto its links is equal to the flows bandwidth demand B_i .

$$\sum_{v \in V} b_{i,S_i,v} = B_i \quad \forall \text{ flow } i \quad (7.10)$$

10. Traffic merging at the destination switch: The total traffic rate flowing into the flows destination switch d_i through its input links is equal to the flows bandwidth demand B_i .

$$\sum_{v \in V} b_{i,v,d_i} = B_i \quad \forall \text{ flow } i \quad (7.11)$$

11. Protection constraint: The fraction of bandwidth available for flow i upon a link failure is at least γ .

$$\left. \begin{aligned} b_min_i &\leq b_{i,1} \quad \forall \text{ flow } i \\ b_min_i &\leq b_{i,2} \quad \forall \text{ flow } i \\ b_min_i / B_i &\geq \gamma \quad \forall \text{ flow } i \end{aligned} \right\} \quad (7.12)$$

7.3 Proposed GVMTPR Algorithm

In this section, we will present a detail description of our algorithm known as *Greedy VM placement with Two Path Routing (GVMTPR)*. The GVMTPR algorithm uses the greedy algorithm and works in two phases.

phase 1: In this phase, VMs are placed at appropriate servers taking into consideration the resource efficiency and the impact of placement on network traffic with the assumption of single shortest path routing. It is noted that network congestion is largely influenced by the placement of VMs. This is because, routing a flow adds load to every link on the chosen path in the network which could possibly increase the congestion (maximum link load). Also, consideration of the number of active servers and resource usage within servers could possibly help satisfy resource requirements for future demands. We consider all the above factors in this phase.

Phase 2: In this phase, we split the traffic in a certain proportion and route them across two link-disjoint paths with the objective of reducing congestion on the paths subject to a specified protection grade.

7.3.1 VM Placement

Algorithm GVMTPR uses a greedy method to map each VM to a server. We label a VM as permanently placed, when a server has been chosen for its placement. For each VM vm_i , an iterative procedure is used to choose a server for placement. The same procedure is repeated for every VM. In iteration j , we construct a tentative mapping j , by tentatively placing vm_i at server j and temporarily placing other VMs (which are not yet permanently placed) at the servers. For the tentative placement of a VM, we first check if its adjacent VM has already been placed in some server. If so, the VM is placed on the same server as its adjacent VM subject to available capacity. If not, the VM is placed at the closest server with maximum residual capacity based on the resource usage cost defined later. The closeness between two servers is defined by the distance between them. We therefore check the servers (i) attached to the same edge switch, (ii) in the same pod, and (iii) different pods, in that order. If the adjacent VM has not yet been placed, we place the VM at a server with maximum residual capacity. By doing so, we use a new server only if no currently-used server has the required capacity and we use a new pod only if all the servers in currently-used pods

cannot accommodate this VM. Once the tentative mapping is completed, the cost of the tentative mapping is calculated. We mainly consider three cost factors– resource usage cost, server usage cost, and bandwidth usage cost.

We define **Resource Usage (RU)** cost for each server S_p , RU_p as the fraction of the resources used in server p . If u is the amount of resource used by all the VMs allocated to S_p and v is the total amount of the resource provided at server S_p , then the resource usage cost is computed as u/v . The resource usage cost is calculated for each of the three resource types– computing, memory, and ingress/egress bandwidth– and RU_p is computed as the average resource usage cost. The resource usage cost RU is the mean resource usage cost considering the usage cost of all servers.

We define **Server Usage (SU)** cost of the Data Center as the ratio between the number of servers currently used and the total number of servers in the Data Center.

We define bandwidth cost for a server pair as the bandwidth required for the traffic from server S_x to server S_y weighted by the hop distance $h_{x,y}$. If $B'_{x,y}$ is the total bandwidth required for all VMs allocated to S_x that communicate to the VMs allocated to S_y , then the bandwidth cost for this server pair is calculated as

$$h_{x,y} * B'_{x,y}. \quad (7.13)$$

We define **Bandwidth Usage (BU)** cost of the Data Center as the ratio between the sum of the bandwidth cost of all the pairs of servers $\langle x, y \rangle$ and the total bandwidth requirement of the traffic (B_T). It refers to the weighted average number of hops for unit traffic, and is given by

$$\sum_{x,y} h_{x,y} * B'_{x,y} / B_T \quad (7.14)$$

We calculate the cost of a tentative placement as $\alpha * RU + \beta * SU + \gamma * BU$, where α , β , and γ are constants. We choose $\alpha \ll \beta \ll \gamma$ so that the cost due to network traffic across switches is minimized followed by that due to the number of active servers and by the usage of server resources. We repeat the above tentative mapping procedure for VM_i for all possible servers and choose the server which yields minimum tentative mapping cost. Now that VM_i permanently placed, we proceed to permanently place other VMs by adopting the same procedure. The pseudo code of this phase of VM placement is given in Algorithm 3 and Algorithm 4.

7. VIRTUAL MACHINE PLACEMENT WITH TWO PATH ROUTING

Algorithm 3 VM placement(i)

```
1: min_cost =  $\infty$ 
2: for each server  $j$  do
3:   tentatively place VM  $i$  at server  $j$ ;
4:   construct Tentative_Mapping ( $i, j$ ) and obtain its cost  $C(i, j)$ ;
5:   if cost  $C(i, j) < \text{min\_cost}$  then
6:     min_cost = cost  $C(i, j)$ ;
7:     chosen-server =  $j$ ;
8:   end if
9: end for
10: place VM in chosen-server;
```

Computational complexity: We now determine the worst case complexity of our algorithm for N servers and M VMs. For a given VM, constructing a tentative mapping to one server and computing its cost requires $O(NM)$ time in the worst case. Permanent mapping of a VM requires tentative mapping to each of N servers and thus takes $O(N^2M)$ time in the worst case. Permanently placing all the M VMs thus takes $O(N^2M^2)$ time which is the worst case time complexity.

Similar work [42] that considers bandwidth between VM clusters for VM placement using the the Markov approximation technique developed in [119] to find the near optimal configuration (in terms of placement of VMs and finding the less-congested paths) . The statespace of the Markov Chain is a function of the number of VMs, servers, as well as the number of paths between the servers. This essentially means there can be millions of states for a large data center with thousands of nodes hosting hundreds of VMs and having multiple paths between servers, raising doubt on the scalability of the approach. In another work [94], the researchers proposed a min-cut clustering-based heuristic algorithm whose runtime complexity is $O(n^4)$, where n is the number of VMs. Our GVMTTPR algorithm performs better when compared to these existing methods.

7.3.2 Two Path Routing

This section details the proposed algorithm for choosing paths for the flows and the traffic proportion to be assigned to the paths. We leverage on the availability of multiple paths between source VM s_j and destination VM d_j , with the goal of distributing the

Algorithm 4 Tentative_Mapping (i, j)

```

1: for each VM  $k$  not yet permanently placed do
2:   if neighbor VM  $k'$  has already been placed in server  $S$  then
3:     if server  $S$  has sufficient capacity then
4:       place VM  $k$  in  $S$ ;
5:       update usage data;
6:     else
7:       choose server  $S'$  closest to  $S$  with sufficient capacity;
8:       update usage data;
9:     end if
10:  else
11:    choose a server  $S'$  with sufficient capacity giving priority to
12:    partially-used server and pod in that order;
13:    update usage data;
14:  end if
15: end for
16: compute the cost of Tentative_Mapping denoted by cost  $C(i, j)$ ;

```

traffic across the paths and partially protecting the traffic from single link failures. We compute two least-congested disjoint path for each pair of communicating VMs.

Traffic flows can be split at flow level or packet level. When packet-level traffic splitting occurs, some packets generated by a flow are sent through one path while other packets are sent through the other path. The effect of packet reordering due the traffic splitting is less pronounced, because, the diameter of a Data Center network is short and the hop lengths of the paths are mostly the same. When splitting is used at flow level, the entire traffic of some flows are sent through one path while the entire traffic on other flows are sent through the other path. The flow traffic is split only at the edge and the traffic follows the pre-allocated paths without requiring any further splitting along the path traversed.

Computational complexity

If w is the number of switches in the network, the computational complexity of the path selection algorithm is given by $O(w^2)$. This complexity can be reduced if we use pre-computed set of alternative paths between pairs of switches at the expense of performance degradation. Suppose that we consider K alternative paths. The longest

7. VIRTUAL MACHINE PLACEMENT WITH TWO PATH ROUTING

path between two edge switches traverses an aggregate switch, a core switch and another aggregation switch. The hop length of such a longest path is four which is independent of the number of switches in the network. Therefore it takes only $O(K)$ time to choose two least-congested paths. By choosing a sufficiently large number of alternative paths such degradation can be made acceptably small.

7.3.3 Illustration of Routing and Path Load

This section illustrates the path selection and path load calculation for two cases of routing: Intra-pod and Inter-pod routing. As an example, suppose that VM_1 in H_1 wants to send a flow to VM_2 in H_2 , both of which are in the same pod. In this example scenario, the two possible link-disjoint paths between H_1 and H_2 are shown in Figure 7.3. The links are marked with the current load and those links and nodes in the network which are not on the paths are not shown for the purpose of clarity. While path $S_1 - S_2 - S_4$ has a load of 10, path $S_1 - S_3 - S_4$ has a load of 15. We note that the path load is the maximum load on any link traversed by the path. Inter-pod routing searches for paths between different pods in the network. We now consider a case of inter-pod routing as shown in Figure 7.3. Suppose that VM_1 placed in H_1 in Pod 1 wants to send a flow to VM_2 placed in H_2 in Pod 4. Those links and nodes that are not along the communication paths are omitted for clarity. The links are marked with the current load. The two possible paths between H_1 and H_2 are $S_1 - S_2 - S_4 - S_8 - S_{10}$, $S_1 - S_3 - S_6 - S_9 - S_{10}$, with the path loads 10 and 15, respectively.

7.3.4 Traffic Splitting Policy

Our traffic splitting policy selects the fraction of traffic assigned to each of the two paths. The maximum protection grade that is achieved is 0.5, when the traffic is split equally. When there is no traffic splitting, the protection grade becomes zero. We note that minimizing congestion does not necessarily result in maximum protection grade. We illustrate the impact of splitting on congestion and protection grade in Figure 7.4. Consider three link-disjoint paths, p_1 , p_2 , and p_3 , with their load, 50, 35, and 15, respectively, as shown in Figure 7.4 (a). Suppose that a new flow with traffic rate of 100 units needs to be routed. We consider three cases of splitting: (i) 50:50, (ii) 60:40, and (iii) 55:45. When the traffic is split in the ratio 50:50, the congestion is 85 with protection grade 0.5 as shown in Figure 7.4 (b). When the traffic is split in the

Figure 7.2: Intra-pod routing

Figure 7.3: Inter-pod routing.

7. VIRTUAL MACHINE PLACEMENT WITH TWO PATH ROUTING

Figure 7.4: Illustration of traffic splitting and protection grade. (a) Traffic Load across three paths. (b) 50:50 Traffic splitting. (c) 60:40 Traffic splitting. (d) 55:45 Traffic splitting

ratio 60:40, the congestion is 75 with protection grade 0.4 as shown in Figure 7.4 (c). When the traffic is split in the ratio 55:45, the congestion is 80 with protection grade 0.45 as shown in Figure 7.4 (d). If the objective is to maximize protection grade (with no consideration of the congestion) then case (i) is the right choice. If the objective is to minimize congestion (with no consideration of the protection grade) then case (ii) is the right choice. If the objective is to minimize the congestion subject to a certain constraint, say, for example, at least 45% protection grade, then case (iii) is the right choice. Suppose the minimum protection grade required is 30%. Splitting the traffic 30:70 results in a load of 65, and 85, on p_2 and p_3 , respectively with the congestion being 85. A better choice is to split traffic in the ratio 40:60 as in case (ii) that result in congestion of 75 only and meet the protection requirement as well.

Having illustrated the relationship between traffic protection and congestion as a

result of traffic splitting, we now explain our traffic splitting policy. Let r be the protection grade required. We begin with splitting the traffic in the ratio $r:(100-r)$ and calculating the resulting congestion. We then proceed with calculating congestion for different values of r in step of 5 until $r=50$, and choose the split ratio that result in minimum congestion.

7.4 Performance Evaluation

7.4.1 Simulation Setup

A Data Center with 3-layer Fat Tree topology [33, 73] consisting of 128 server hosts, each with 1000 units of resources of each type CPU and memory is generated using ns-3 simulator [120, 121]. The servers ingress/egress bandwidth and link bandwidth are set to 1000 units. The jobs arrive randomly as a Poisson process with the mean inter-arrival time of 1 second. We consider broadly two types of VMs: large VM and small VM. The resource requirement of a large VM is in the range of 18%–22% of the total amount of resources(CPU, memory, ingress/egress bandwidth) in a host and that of a small VM is in the range of 1%–1.5% of host resources. We choose a certain number of large VMs and small VMs for a job in a certain proportion, keeping the total amount of bandwidth units fixed.

The performance of the proposed GVMTPR algorithm is compared with the following well-known VM placement heuristics and a routing algorithm.

1. Random Selection Among the servers which can satisfy the VM's resource requirements, this heuristic chooses a server randomly to place a given VM.
2. First Fit - Among the servers which can satisfy the VM's resource requirements, this heuristic chooses the first server based on its identifier.
3. Single path routing - This method selects the shortest path with the minimum hop count.

7.4.2 Bandwidth Cost

The performance of the algorithms is compared in terms of bandwidth cost which is calculated as the amount of traffic units sent between servers weighted by the hop

7. VIRTUAL MACHINE PLACEMENT WITH TWO PATH ROUTING

Figure 7.5: Performance evaluation based on bandwidth cost [9]

count. Figure 7.5 shows that GVMTPR method has much lower bandwidth cost for varying demand sizes. We choose the number of large VMs and small VMs in a job based on the fraction of large VMs with the total amount of bandwidth units set to a fixed value. Our method shows about 50% improvement when compared to Random and First-Fit methods. It can also be observed from the figure that the bandwidth cost decreases slowly with the increasing fraction of large VMs. The reason for this behavior is as follows. For a fixed-total amount of bandwidth, when the fraction of large VMs is small, there are only a small number of large flows and a large number of small flows. This increases the possibility of more number of small flows taking longer hop routes leading to higher bandwidth cost. When the fraction of large VMs increases, the number of small flows decreases leading to less number of small flows taking longer hop routes.

7.4.3 Impact of VM Size Distribution

Our next study is to evaluate the impact of VM size distributions on the performance of the algorithms. The size is defined by the amount of resources required by the VMs in a

Figure 7.6: Performance comparison with varying VM sizes [9]

job. We use the resource requirement patterns similar to that in [42]. Applications with uniformly-large demand use large VMs. Applications with uniformly small demand use only small VMs. Applications with a random demand require an amount uniformly distributed between 25% and 100% for each VM. The bimodal demand is generated from two normal distributions $N(0;3;0:1)$ and $N(0;7;0:1)$ with equal probability. We used the metric *maximum core switch utilization* which is defined as the maximum value of the utilization of the links connected to the core switches. From Figure 7.6, it is observed that our GVMTPR algorithm is effective with varying VM size distributions when compared to Random and First-Fit methods.

7.4.4 Impact of Flow Sizes

The performance of the proposed algorithm is studied for varying flow sizes. We generated elephant flows with the traffic rate in the range of 180–220 units, and mice flows with the traffic rate in the range of 1–2 units. For a fixed total traffic load, we varied the fraction of elephant flows. Since mice flows are short and could be latency-sensitive, we split only the elephant flows. Figure 7.7 shows the performance of the algorithm.

7. VIRTUAL MACHINE PLACEMENT WITH TWO PATH ROUTING

Figure 7.7: Performance under varying flow sizes [9]

From the figures, it is observed that when the fraction of elephant flows is small, a large number of mice flows are generated since the total amount of traffic generated is fixed. This increases the possibility of some flows passing through core switches. When the fraction of elephant flows increases, more traffic is carried out by elephant flows and the number of mice flows decreases. Therefore, the possibility of flows passing through core switches decreases with the increasing percentage of elephant flows resulting in the decreasing trend of the maximum core utilization.

7.4.5 Load Balancing

The performance of the proposed algorithm in terms of the load measured in units of Bps (Bytes per second) that pass through different core switches is studied. We consider equal proportion of large and small VMs. It is observed from Figure 7.8 that, the proposed algorithm not only results in smaller load on switches but also in even distribution of loads across the switches when compared to other algorithms. The simulation results show that our GVMTPR algorithm, reduces the stress of core switches by up to 40% when compared to random and first-fit methods.

Figure 7.8: Traffic load across core switches [9]

7.4.6 Demand Acceptance Ratio

The performance of the proposed GVMTPR algorithm in terms of demand acceptance ratio is compared with Random and First-Fit allocation methods. We define the demand acceptance ratio as the ratio between the number of demands (requests) accepted and number of demands arrived. We recall that 1000 units of server resources are used in our experiments. We study the performance with varying amount of server sources available for VMs. The amount of server resources available is normalized as a fraction γ of the default value of 1000 units, where $0 < \gamma < 1$. Figure 7.9 shows that our GVMTPR algorithm achieves higher demand acceptance ratio than the other algorithms.

7.4.7 Performance with Impact of Workload

The performance of our proposed algorithm in terms of maximum link utilization for varying workload with the inter-arrival-time $\tau=0.5$ sec and $\tau=0.25$ sec is analysed. We consider VM sizes uniformly distributed with 3 different ratios ($\epsilon=1.00$; 0.50 ; and 0.25) of large and small VMs. From Figure 7.10, It is observed that with the in-

7. VIRTUAL MACHINE PLACEMENT WITH TWO PATH ROUTING

Figure 7.9: Demand acceptance ratio [9]

Figure 7.10: Maximum core link utilization for varying workloads [9]

Figure 7.11: Protection grade [9]

creasing load, the maximum link load increases. The proposed GVMTPR algorithm outperforms the other two algorithms with considerable performance improvement for different workloads.

7.4.8 Protection Grade

The performance of the proposed GVMTPR algorithm in terms of the protection grade guaranteed by varying fractions of large VMs is shown in Figure 7.11. The percentage of large VMs is varied from 0 to 1 in steps of 0.25. The minimum required protection grade is set as 45%. It is noted that the maximum possible protection grade is 50% with two path routing. The figure shows that the proposed algorithm achieves more than the specified protection grade. It also shows that protection grade slowly increases with the increasing percentage of large flows. As explained before, when the percentage of large flows is small, there will be large number of small flows since we fixed the total amount

7. VIRTUAL MACHINE PLACEMENT WITH TWO PATH ROUTING

of traffic. If the bandwidth of a (small) flow is much less than the difference between the loads on two paths then there is a high possibility of unequal splitting closer to the minimum specified protection grade (in our case, it is 45%) in an attempt to minimize the maximum load on the paths. For example, suppose that a flow requires 10 units of bandwidth and the loads on two paths are 90 and 60 units, respectively. Here the difference in path loads is much higher than the required bandwidth. The split ratio of 45:55 will result in the path loads of 94.5 and 65.5 with the congestion of 94.5 which is the minimum congestion possible meeting the protection grade requirement of 45%. Therefore the protection grade is low when the percentage of large flows is small.

7.5 Summary

In this chapter, the VM placement and traffic routing problem to reduce congestion while guaranteeing a certain protection grade for traffic flows was investigated. We developed an efficient algorithm for this problem, analyzed the computational complexity, and carried out performance study. The algorithm uses greedy method for intelligently placing VMs onto physical servers so as to use the resources efficiently and reduce the network traffic. The traffic is routed on two link-disjoint paths with a certain proportion so as to reduce congestion while satisfying the protection grade requirement. We demonstrated the effectiveness of the proposed algorithm through simulations considering various scenarios and performance metrics. Some interesting directions for future work include consideration of more alternative paths for improved traffic protection and also reservation of backup resources for full traffic protection and the impact on resource efficiency.

Chapter 8

Power Efficient VM placement and Migration

This chapter studies the benefits of considering the network knowledge in VM placement and migration in terms of energy efficiency. Data Center operating cost is influenced by its power consumption. Recent studies [122, 123] have shown that equipment including servers, storage and network devices consume around 45% of the total power in Data Centers. Various models of servers and switches consume power at different levels. According to [124], a high-end server (IBM powersaver 7 with multicores) consumes from 2,000–15,000 Watts and a volume server chassis consisting of 14 blade servers consumes around 4,500 Watts under full workload [125, 126]. Similarly, a high-end switch such as Cisco Catalyst 6500 series consumes 400–4,000 Watts while an edge switch such as Dell PowerConnect series consumes around 100 Watts only. The switch's power consumption depends on the number of active links and their rates [127]. Data Center power consumption can be reduced by minimizing the number of active servers and switches.

To optimize the Data Center performance in terms of power consumption, it is required to design an efficient strategy for placing virtual machines (VMs) among the physical machines (PMs) in such a way that only a few servers and switches are activated. This can be achieved by letting a source-destination VM pair that are close to each other communicate through edge switches instead of core switches. By migrating VMs of communicating parties to a close location, core switches usage can be reduced and energy can be saved. Thus, location affinity of source-destination VMs contributes

more to power saving than simply aggregating traffic. We design a SDN-based VM placement and migration solution that ensures power efficiency in Data Centers. It attempts to use fewer PMs and schedule VMs to migrate when necessary. The research work presented in this chapter has been published in [10].

8.1 Power Efficient Virtual Machine Placement and Migration Scheme

We first describe the terms used in our design and then illustrate our power efficient VM placement and migration scheme with traditional virtual machine placement algorithms such as *First Fit*[128, 129], *Best Fit* [129] and *Random*.

A **job** is denoted by $\{ JobID, JobPriority, VMlist \}$. *JobID* denotes the job sequence number depending on job arrival time. *JobPriority* denotes job priority level, which corresponds to requirements on response time, delay and throughput. *VMlist* specifies number of VMs for each type that are required by this job. Each list entry is specified in the format of *VMTypeID : VMnumber*.

A **Virtual Machine** (VM) requires several system resources like CPU, RAM, storage and bandwidth. Different types of VMs differ in system resource demands. A type of VM is defined by parameters $\{ CPU, RAM, Storage, Bandwidth, VMTypeID \}$. The parameters *CPU*, *RAM*, *Storage*, *Bandwidth* denote respectively, resources required by the type of VM that is specified in the *VMTypeID* field.

A **hypervisor** is running on a PM and creates VMs on it. It tracks the use of PM resources. A hypervisor is identified by its $\{ Hypervisor ID, CPU, Memory, Storage, Bandwidth, Local Capped Utilization \}$ specifying the total CPU, memory, storage and bandwidth units of its hosting PM. The last field specifies a percentage as the threshold to indicate that utilization is becoming close to full capacity.

The proposed framework is shown in Figure 8.1. It has two major components: centralized controller and SDN controller. The centralised controller serves as a hypervisor responsible for resource allocation for VMs and orchestrates PMs and VMs to host applications. The SDN controller is responsible for dynamic routing and management of network devices. When a new job arrives, the centralized controller deploys the VMs required to process the job. A job is processed through two stages namely, VM placement and VM migration. In VM placement stage, the *PM checker* component

8.1 Power Efficient Virtual Machine Placement and Migration Scheme

checks whether the current resources on the PM must be sufficient to meet the VM requirement for VM deployment. If sufficient resources are available, the *VM Deployment* component deploys the VMs. If any one of the VMs cannot be deployed, this job will be suspended for a random time before checking again. The current network traffic volume between any two VMs can be monitored by SDN controller and sent it to the centralized controller. The centralized controller periodically checks job status, PM utilization, and network traffic volume. Based on these information, the controller schedules VM migration when necessary. Our VM migration scheme reschedules the VMs in a such a way to form more inner pod flows (than inter pod flows) so that less core switches are needed.

Figure 8.1: Power aware VM placement and migration illustration.

8.2 Problem Formulation

We model a Data Center network using a directed graph $G=(V,E)$, in which each node $v \in V$ represents a network switch, PM (host) or an external client. Each directed edge $(u, v) \in E$ represents a physical link between the switches or between the switches and the hosts. Each link (u, v) has a nonnegative capacity $c(u,v) \geq 0$. We implicitly assume that $c(u, v) = 0$ for $(u, v) \notin E$.

We consider a set of t hosts with their resource capacities given along memory, CPU, storage and network bandwidth dimensions. There are m virtual machines to be placed. The requirements of these virtual machines are given along the dimensions of memory, CPU storage and network bandwidth. We have to find a mapping between VMs and PMs which satisfies the VM's resource requirements while minimizing the number of PMs used, minimizing data center network energy.

Input: Let there be t PMs $S = \{S_1, S_2 \dots S_t\}$ and each host $S_i = \{c_i, r_i, s_i, b_i, u_i\}$ with different resource capacity attributes: c_i for CPU, r_i for RAM, s_i for storage, and b_i for bandwidth. Each server has a capped resource utilization value, u_i beyond which no more resources are allocated.

We consider a set of m different virtual machine types depending on their resource needed (VMs) $V = \{V_1, V_2 \dots V_m\}$. Each VM type $V_i = \{c_{vi}, r_{vi}, s_{vi}, b_{vi}\}$ with different resource usage attributes: c_{vi} for CPU, r_{vi} for RAM, s_{vi} for storage, and b_{vi} for bandwidth.

We are given a set of n jobs from tenants that need to be assigned to a Data Center, each job J_i with a list of VM types and number of requested VMs for that type. $J = \{J_1, J_2 \dots J_n\}$ where $J_i = \{x_1, x_2 \dots x_m\}$ and $\forall k, x_k$ is the number of VM type V_k needed by Job J_i .

The network traffic source or destination is one of the m VMs. Here, over a link $(u, v) \in E$, we reserve bandwidth $b_i(u, v)$ for demand n .

$$b(u, v) = \sum_{i=1}^{i=n} b_i(u, v) \quad (8.1)$$

is then the aggregated bandwidth over the link (u, v) . For each job j , we need to find m_k feasible PMs to support its physical resource requirements, as well as a set of paths in G to route its traffic among VMs. Each path consists of a set of links that inter-connect switches and PMs.

We define $Host_i = \{host_{i1}, \dots, host_{im}\}$ for each server S_i that denotes the set of VMs assigned to that server, where $host_{im} = 1$ if node S_i is hosting the VM m and $host_{im} = 0$ otherwise. The power consumed by server S_i depends on its physical component as well as on the set of VMs present:

$$P(S_i) = P(S_i, Host_i) \quad (8.2)$$

The objective function for power-efficient VM placement and migration can be written as,

$$\min (P(S_i)) \quad (8.3)$$

We consider the capacity constraints, placement constraints, bandwidth constraints, bandwidth demand constraints as discussed in Chapter 7 that need to be satisfied during placement of virtual machines with traffic routing.

In next section, we present heuristic algorithms to tackle the problem of VM placement and migration.

8.3 Heuristic algorithms for VM placement and Migration

8.3.1 Virtual Machine Placement

The proposed power-efficient VM placement and migration algorithm is described along with three traditional deployed VM placement algorithms, namely *First Fit*, *Best Fit* and *Random*.

The proposed power-efficient placement mechanism aims to find out an appropriate host(PM) to deploy a VM. It first checks if the VM can be assigned to any of active hosts. Since this mechanism first targets on active PMs, it not only aggregates traffic to a smaller number of active PMs, but also assures the active PMs are close enough to full utilization before activating a new PM. The *First Fit*, *Best Fit* and *Worst Fit* algorithms are considered for the power aware VM placement. In a First Fit algorithm, the centralized controller activates a single host at a time and each VM gets deployed in the first suitable host(PM) according to a predefined order between the hosts[128]. We call this *First Fit Placement* as *Powerefficient – FirstFit*. In a Best Fit algorithm,

8. POWER EFFICIENT VM PLACEMENT AND MIGRATION

list of all active hosts(PMs) is maintained at all times and the VMs can be deployed to the PM where it best fits among the active hosts. The centralized controller computes weights for PMs as follows:

$$w_i = \Delta c_i / c_i + \Delta r_i / r_i + \Delta s_i / s_i + \Delta b_i / b_i \quad (8.4)$$

where Δc_i , Δr_i , Δs_i , and Δb_i denote the unutilized resource of CPU, RAM, storage, and bandwidth respectively. The centralized controller deploys the VM in the PM with lowest weight in Best Fit algorithm. We call this *Best Fit Placement* as *Powerefficient – BestFit*. In the *Worst – Fit* algorithm, centralized controller calculates the weight and deploys the VM in the PM with the highest weight. We name this *Worst Fit Placement* as *Powerefficient – WorstFit*.

In a Random placement algorithm, the centralized controller selects a random host to accommodate the VMs. If the selected host cannot satisfy the VM requirements, the mechanism continuously explores other hosts for T times. The parameter of T is called *times – to – try*. We name this *Random Placement* as *Random*.

After the completion of the job, a VM will be released. After VM release, active hypervisors check the VMs assigned to its hosting PM. If no VMs are assigned, the hypervisor instructs the PM to switch into the inactive mode.

8.3.2 Virtual Machine Migration

VM migration is invoked when a PM cannot accommodate the incoming request. In this stage, two communicating VMs might be moved into the same pod. Since each pod contains N^2 end stations [33], we use the PM index to check whether they are in the same pod. The mechanism first checks if the source VM and destination VM are in the same pod. Otherwise, they attempt to migrate either source VM or destination VM to an active PM in the same pod of the other VM. If neither source pod nor destination pod can accommodate the other VM, the algorithm tries to allocate either VM to an inactive PM. They return true when a VM can be migrated to a new hosting PM, otherwise it returns false. Depending on which algorithm is implemented, the migration mechanism varies. Generally, they follow the steps as shown in Algorithm 5.

The three algorithms differ in selection of an active PM to host the VMs. In FF, the controller checks sequentially and migrates the VM to the first one with sufficient

resource. In the contrary, the BF algorithm calculates the weight first, and migrate to the PM with the lowest weight value and WF algorithm calculate the weight first, and migrate to the PM with the largest weight. We assume to employ pre-copy memory migration [130]. In pre-copy migration, the hypervisor firstly copies all the memory from the migration source to the migration destination, while the VM is still operating on the migration source. The VM will be resumed on the migration destination when the copy at the destination is sufficient for VM execution.

Algorithm 5 Migration algorithm

```

Calculate src_pm_id, namely PM id of source VM
Calculate des_pm_id, namely PM id of destination VM
▷ Check whether in the same pod id
if  $src\_pm\_id/N^2 \neq des\_pm\_id/N^2$  then
  ▷ check whether can migrate to an active PM
    if source pod can accommodate destination VM to an active PM then
      Schedule to the active PM
      Return true
  ▷ check whether can migrate to an inactive PM
    else
      if can migrate src/Dest VM to an inactive PM then
        Schedule to the inactive PM
        Return true
      end if
      Return false
    end if
  end if
  Return false

```

8.4 Simulation Experiments

In this section, the performance of the proposed algorithm is studied with two sets of simulation experiments. In the first experiment, the simulation runs on an $N = 4$ fat tree topology with 128 PMs. In the second experiment, the simulation runs on an $N = 8$ fat tree topology with 1024 PMs. In each experiment, we evaluate them from two perspectives, namely *active number of PMs* and *inner pod flow percentage*. Active

8. POWER EFFICIENT VM PLACEMENT AND MIGRATION

number of PMs indicates the server power consumption level in a Data Center. *Inner pod flow percentage* shows the percentage of flows that travels within a pod, called intra-pod flows. Thus, a higher *inner pod flow percentage* indicates lesser core switch power consumption.

Experimental parameters are shown in Table 8.1. In the first experiment, 20 jobs are randomly generated. Each job requires at most 10 VMs and each VM may belong to a set of 15 VM types. Further, each job contains at most 10 flows. In the second experiment, most parameters are the same with the previous one with the exception that the number of jobs is increased. Similarly, in this experiment, 400 jobs are generated with at most 10 VMs from the set of 20 VM types. Each job contains at most 10 flows. The simulation time is capped at 800 seconds.

Table 8.1: Parameter setting in Experiment I and Experiment II

Parameters	Experiment I	Experiment II
Number of Jobs	20	400
Duration	1-20	1-20
Number of VMs per Job	1-100	1-100
Flow per Job	1-10	1-10
Times to Try t	10	10

The active number of PMs for both $N = 4$ and $N = 8$ fat tree are shown in Figure 8.2 and Figure 8.3, respectively. We focus on the performance comparison of *Random mechanism*, *Power efficient -First fit mechanism* and *Power efficient -Best fit mechanism*. From these two figures, we can see that the active number of PMs with the power-efficient mechanism is always smaller than the Random mechanism. During 5-15s and 20-30s, a significant difference can be observed between Random mechanism and Power-efficient mechanisms.

For power-efficient mechanism, the current weight difference function w may not be sufficient to guarantee full utilization. As for *Power efficient-Best Fit algorithm*, the hole (we refer holes as unutilized system resources) may not be small enough. We can observe that generally *Power efficient-First Fit algorithm* performs better with small number of active hypervisors (PMs). The reason is that *Power efficient-First Fit algorithm* tends to allocate VMs used in a job in the same set of PMs. Therefore, when a job ends, a PM can be deactivated. In *Power efficient-Best Fit algorithm*

Figure 8.2: Active number of hypervisors for N=4 fat tree.

Figure 8.3: Active number of hypervisors for N=8 fat tree.

8. POWER EFFICIENT VM PLACEMENT AND MIGRATION

and *Power efficient-Worst Fit algorithm* algorithms, VMs belonging to the same job might be allocated to different PMs. When a job ends, PMs which host VMs from other jobs cannot be deactivated. Considering the above observations, we just compare *Power efficient-First Fit algorithm* with *Random mechanism* in terms of *inner pod flow percentage* in the following study.

The drop in Figure 8.3 at around 200s is due to the rescheduling of jobs when system capacity is full. As in the *Power efficient First-Fit* and *Power efficient Best-Fit* mechanisms, the centralized controller first attempts to fill in all the active PMs. These VMs with fewer resource requirements can be scheduled to active PMs before starting another PM. Therefore, this mechanism firstly fills the *holes* in active PMs. Comparatively, the *Random mechanism* does not give priority for filling in the *holes* than starting an inactive PM. Even when there are smaller VMs that can be assigned to an active PM, the VMs may be allocated to some inactive PMs. When a VM requires more system resources it is more probable that none of the PMs has enough resources to deploy it. Thus, there are bigger holes in PMs and less VMs can be hosted in the *Random* mechanism. As the *Random* mechanism reaches full utilization earlier, it delays more jobs to a later time. When previous jobs complete, there are bigger gaps and delays before job retransmission. Therefore, it reacts slower when system resource is released and results in longer delays.

Thirdly, we consider the job delay for both N=4 and N=8 experiments. Table 8.2 shows the total time of job delay in both experiments. We can see from the results that for the first experiment, the delay for all the algorithms are same. They have the same delay because there is only one job delay due to full utilization. However, for N=8, the difference between the algorithms becomes obvious, in which *Power efficient First-Fit* has the shortest overall delay of 55.06s while *Random* has the longest delay of 76.04. That means, it is much easier for *Power efficient First-Fit* to find sufficient resources for processing a job and its delay is about 26% is lesser than others.

Fourthly, we show the inner pod flow percentage in Figure 8.4 and Figure 8.5 for N=4 and N=8 fat tree, respectively. The two figures show the percentage of number of inner pod flows to the total number of flows for both *Random* and *Power efficient First-Fit* mechanisms with or without VM migration. In a VM migration mechanism, flows are categorized to inner pod flow and inter pod flow. An inner pod flow transits within a pod and does not go through core switches, while an inter pod flow transits

across two different pods and must go through a core switch to reach the destination. An inter pod flow which requires a core switch tends to consume more power.

Table 8.2: Job Delay in Experiment I and Experiment II

Mechanism	Experiment I	Experiment II
Random	6.61s	76.04s
First Fit	6.61s	55.06s
Best Fit	6.61s	63.25s
Worst Fit	6.61s	64.85

Figure 8.4: Inner pod flow percentage for N=4 fat tree [10]

We present results for four schemes, namely, Random, Power efficient-First fit, Random Migration and Power efficient-First fit Migration. We show the inner pod flow percentage obtained from 5 to 35 seconds with an interval of 5s in Figure 8.4, and from 100 to 700 seconds with an interval of 100s in Figure 8.5. We can see from the figures that with Migration algorithm, more than half of the flows can be migrated to form inner pod flows. Thus, fewer core switches are needed in these schemes. The lack of data in Figure 8.5 at 600s and 700s is due to job completion in case of power-efficient

Figure 8.5: Inner pod flow percentage for N=8 fat tree [10]

schemes.

8.5 Summary

In this chapter, the proposed power-efficient solution with virtual machine placement and migration was described. Our proposal saves energy by aggregating virtual machines to a few physical machines to reduce server power consumption. Communicating virtual machines are migrated into close locations to avoid using too many components thus saving the power consumption. The simulation results show that our solution uses less number of physical machines and completes job faster. It achieves up to 50% reduction in the number of physical machines used and around 26% reduction in job delay as well as up to 50% flows migrating to the same pods.

Chapter 9

Conclusion

In this chapter, we summarize the overall contents of the thesis and lists the main achievements. We then discuss the limitations of our work and possible directions for continuing research in these areas.

9.1 Summary

The growing volume of the data transfer and the new traffic types in today's Data Centers make Data Center networks to suffer from frequent link congestions, delayed recovery from switch or link failures. Considering the issues of load balancing, fault tolerance, fairness and energy efficiency, this thesis is focussed on the design and evaluation of dynamic traffic engineering techniques based on SDN capabilities to improve network bandwidth utilization and congestion reduction in DCNs.

In Chapter 3, we considered the issue of scalability limitation of the traditional Ethernet due to the spanning tree protocol, which limits network bandwidth utilization and make it prone to congestion. This is unacceptable in today's Data Center traffic. Thus, there is a need for Layer-2 Multipath solution to distribute the traffic and ease congestions. Most of the existing proposals [TRILL, VL2, Portland] use the ECMP protocol, which route packets to all available paths between each source-destination pair equally without considering the existing traffic load. To overcome this, we proposed a layer-2 multi path solution considering the dynamic network state using computationally simple methods. The proposed load balancing framework uses an intelligent centralized controller which collects the network state information. Our proposed method LAAR

9. CONCLUSION

balances the load across the network, thereby reducing the occurrence of congestion leading to efficient utilization of bi-section bandwidth in Data Center networks. With extensive simulations, we have shown that our proposal outperforms ECMP in terms of bisection bandwidth utilization and balanced load distribution. In our simulation, it was shown that our LAAR algorithm reduces the load of core switches up to 25% when compared to the well known ECMP. This research work has been summarized and published in [5].

Continuing the work in chapter 4, we developed an SDN based re-routing control prototype in chapter 4, for balancing the load by dynamically reallocating different flows in Data Center networks in the event of link congestion and faults. Our prototype is suitable for Data Centers with unpredictable traffic characteristics. We implemented the framework with NOX controller in the OpenFlow testbed and demonstrated that our re-routing mechanism is better than ECMP and single path routing schemes in terms of load balancing and throughput. Also we demonstrated the effectiveness of our algorithm on a large scale using simulations. With our prototype, it was shown that our SDN-based edge-to-edge rerouting technique reduces the stress of core switches up to 40% when compared to the well known ECMP. This research work has been summarized and published in [6].

In chapter 5, we extended our SDN-based dynamic routing prototype to support applications like data replication to keep multiple instances of synchronized files or databases for fail over or disaster recovery purpose in Data Centers. In our prototype, it was shown that our SDN-based replication technique improves the performance up to 20% and reduces the stress of core switches up to 25% when compared to the traditional iSCSI method. Our results proved that our SDN-based approach outperforms the traditional iSCSI based approach. This research work has been summarized and published in [7].

Although, SDN provides an excellent support for setting intelligent routing decision, it is still a challenge for SDN to provide a fair treatment for elephant flows in a Data Center due to the limitation in SDN switch's forwarding table size. Since, SDN switch's forwarding table can support only a few entries, elephant flows suffer from frequent evictions because of the flooding of mice flows. In chapter 6, we addressed this problem by adopting a differentiated approach and proposed flow cache architecture with dynamic-index hashing for placing the flow records onto various buckets. We also

proposed a localized LRU-based replacement strategy, which gives good performance in terms of lookup time and hit ratio. Overall the differential flow cache architecture provides a simple and effective means to address the overload on the controller due to flow table exhaustion at the OpenFlow switch. It was shown that our proposal reduces the lookup time up to 50% when compared to the wild-card linear method. This research work has been summarized and published in [8].

Our SDN framework, which has excellent support for introducing new control policies and services, motivated us to explore how the consideration of network knowledge in VM placement and traffic distribution help us to achieve performance benefits in terms of congestion and traffic protection. In chapter 7, we developed an algorithm called GVMTPR for VM placement and two path traffic routing to minimize congestion while providing the specified protection grade. The algorithm uses a greedy method for intelligently placing VMs onto physical servers in such a way to have reduced traffic load on the network. The traffic is routed on two link-disjoint paths with a certain proportion so as to reduce congestion while satisfying the protection grade requirement. We demonstrated the effectiveness of the proposed algorithm through simulations considering various scenarios and performance metrics. It was shown that our GVMTPR improves the bandwidth cost up to 50% and reduces the stress of core switches up to 40% when compared to baseline methods. This research work has been summarized and published in [9].

In chapter 8, we explored how the consideration of the network knowledge can help to achieve performance benefits in terms of energy efficiency. We proposed a power-efficient solution for virtual machine placement and migration. Our proposal saves energy by aggregating virtual machines to a few physical machines to reduce server power consumption and ensures the number of migrations is small and the placement decisions are fast, which makes it practical for large Data Centers. It achieves up to 50% reduction in the number of active physical machines, up to 26% reduction in job delay as well as up to 50% flows migrating to the same pods, when compared to baseline methods. This research work has been summarized and published in [10].

9.2 Future Work

The field of Data Center networking is still a growing area, and new and innovative ideas appear continuously. Here, we discuss limitations and the open problems related to the work in this dissertation as well as future directions for ongoing research in the areas of VM placement with traffic protection, energy efficient VM placement and virtual network abstraction in Data Centers.

Due to the non-availability of real data centers and their traffic traces for the purpose of experimentation due to privacy and security concerns, we used synthetic patterns along the lines of traffic distributions considered in the literature [27, 69] for evaluating our techniques. The experimentation of our approaches with large scale real Data Centers is hence limited.

We have studied traffic engineering problem with VM placement, adaptive routing and congestion improvement mechanisms for a Data Center. We also used SDN for adaptive control of Data Center network with appropriate routing and rerouting mechanisms. We can further extend this work to consider a global and integrated control framework considering the virtualized resources of both network and server (computing and storage) systems. Recently in [131], such an integrated framework called software defined infrastructure has been presented and mechanisms have been proposed to enable SDN applications in such a framework. The problem of satisfying different kinds of service level agreements (SLAs) cutting across network and server resources for guaranteed services requires new solutions, such an integrated approach has an advantage of improved resource usage, lesser overhead and predictable performance for tenants. For example, global monitoring of resources will lead to better migration strategies to decide between path migration (rerouting) and VM migration for a tradeoff between resource utilization and overhead.

Data Centers provide three kinds of resources -computing, storage and networks. They use different kinds of technologies from multiple different providers like VMware, Microsoft Hyper-V, cisco, brocade, Juniper, riverbed, etc. There is a shift in trend of network services from running on a few physical appliances moving to hundreds of virtual appliances. The management of this heterogeneous environment is complex and a possible solution is to have a multi-virtualization and multi-cloud management platform. An important issue is the placement of virtual appliances to determine if virtual

appliance VMs can share resources with tenant VMs or require dedicated resources. Such placement and resource allocation would require effective traffic engineering for improved resource sharing.

The current practice is to reserve the computing and storage resources for the tenants. The network bandwidth is not reserved for the tenants. This leads to unpredictable performance of the applications [132]. Further, because of insufficient bandwidth availability, the applications may use VM resources for a long time implying higher cost for the tenant. Recently, there has been proposals to provide virtual network abstractions to the tenants. A virtual network is comprised of a set of nodes representing a group of VMs which are connected by a set of links representing communication between nodes. Also, associated with a virtual network, there is a guarantee of resources (computing, storage and bandwidth) to satisfy tenants requirements. The dedicated network abstractions may not accurately model the communication patterns of real applications. To overcome such a poor abstraction, in [133], possible application abstractions in the form of tenant application graph have been proposed. An important problem is how such abstractions and the associated resource requirements can be mapped to the Data Center servers and network links in a way to guarantee predictable performance for the tenants while ensuring improved resource usage for the providers. Since, there are traffic requirements between various nodes (VM groups) in such application abstractions, effective traffic engineering is required for improved resource usage.

In our work on VM placement with traffic protection, we considered traffic distribution on two paths which could provide protection grade up to 50%. It would be interesting to consider more alternative paths for improved traffic protection beyond 50% and its implications on computational complexity and implementation issues. The protection grade can be improved by distributing the traffic over increased number of paths. Suppose a traffic flow requiring b units of bandwidth is distributed among n link-disjoint paths with b_{max} as the maximum. Here, the protection grade guaranteed is $\frac{b-b_{max}}{b}$. if n increases, b_{max} is more likely to decrease, leading to increased protection grade. Another possible work is to consider reserving spare resources (backup resources) on a path to increase the protection grade even to full traffic protection. Such spare resource reservation will lead to decreased number of admitted flows. Developing mechanisms to achieve a trade-off between traffic protection, congestion and acceptance ratio will be an important future study.

9. CONCLUSION

Our work on power saving with VM placement and migration can be extended by considering energy proportional network switches. While the present commodity switches are not energy proportional, energy proportional switches are likely to be popular in near future. For example, recently in Energy Efficient Ethernet (IEEE 802.3az) [134], energy efficiency enhancements have been introduced wherein energy consumption depends on the amount of traffic and the timing of frame arrivals. In such switches, the energy consumption on a port or link is proportional to the current traffic load. Study of energy efficient techniques for networks with such switches will be an interesting future work.

List of Publications

Journal Publications:

1. K. Renuga, B. S. Lee, R. Vasanth, and K. M. M. Aung, **Adaptive routing for Layer-2 load balancing in Data Center networks**, *AICIT International Journal on Advances in Information Sciences and Service Sciences*, Vol. 4, No. 15, pp. 1 - 12, 2012.
2. K. Renuga, B. S. Lee, N. T. Dat, N. M. Luke, and K. M. M. Aung, **Virtual machine placement with two-path traffic routing for reduced congestion in Data Center networks**, *Elsevier Computer Communications Journal*, Vol. 53, pp. 1-12, November 2014.

Conference Publications:

3. K. Renuga, N. Luke, K. M. M. Aung, B.S. Lee, and Heryandi, **OpenFlow based control for re-routing with differentiated flows in Data Center Networks**, in *Proc. of IEEE International Conference on Networks, ICON'12*, 2012.
4. S. Fang, K. Renuga, B. S. Lee, C. H. Foh, and K. M. M. Aung, **Power-efficient virtual machine placement and migration in Data Centers**, in *Proc. of IEEE Greencom'13*, 2013.
5. B. S. Lee, K. Renuga, and K. M. M. Aung, **An efficient flow cache algorithm with improved fairness in software-defined Data Center networks**, in *Proc. of IEEE Cloudnet'13*, 2013.
6. K. Renuga, B. S. Lee, N. M. Luke, N. T. Dat, and F. M. Rodel, **Software defined network based adaptive routing for data replication in Data Centers**, in *Proc. of IEEE ICON'13*, 2013.

References

- [1] CISCO. **Cisco Global Index**. http://www.cisco.com/en/US/solutions/collateral/ns341/ns525/ns537/ns705/ns1175/Cloud_Index_White_Paper.html. ix, 3, 4, 23, 29
- [2] NICHOLAS JOHN LIPPIS. **A Simpler Data Center Fabric Emerges**. http://www.cisco.com/c/dam/en/us/products/collateral/switches/nexus-7000-series-switches/Data_Center_Net_Fabric.pdf, 2010. ix, 3, 17
- [3] C. GUO, H. WU, K. TAN, L. SHI, Y. ZHANG, AND S. LU. **Dcell: a scalable and fault-tolerant network structure for data centers**. *SIGCOMM Computer Communication Review*, **38**(4):75–86, 2008. ix, 17, 20, 21
- [4] C. GUO, G. LU, D. LI, H. WU, X. ZHANG, Y. SHI, C. TIAN, Y. ZHANG, AND S. LU. **BCube: A high performance, server-centric network architecture for modular data centers**. *SIGCOMM Computer Communication Review*, **39**(4):63–74, 2009. ix, 17, 20, 22
- [5] K. RENUGA, B.S. LEE, R. VASANTH, AND K.M.M. AUNG. **Adaptive routing for Layer-2 load balancing in Data Center networks**. *AICIT International Journal on Advances in Information Sciences and Service Sciences*, **4**(15), 2012. ix, 37, 49, 50, 51, 52, 136
- [6] K. RENUGA, N. LUKE, K. MI MI, B.S. LEE, AND HERYANDI. **OpenFlow based control for re-routing with differentiated flows in Data Center Networks**. In *Proc. of IEEE International Conference on Networks, ICON'12*, 2012. x, 55, 64, 65, 87, 136
- [7] K. RENUGA, N. M. LEE, B. S. AND LUKE, N. T. DAT, AND F. M. RODEL. **Software defined network based adaptive routing for data replication in Data Centers**. In *Proc. of International Conference on Networks*, 2013. x, 72, 79, 81, 136
- [8] B. S. LEE, K. RENUGA, AND K. M. M. AUNG. **An efficient flow cache algorithm with improved fairness in software-defined Data Center networks**. In *Proc. of IEEE Cloudnet*, 2013. x, xi, 84, 93, 94, 95, 96, 97, 98, 137

REFERENCES

- [9] K. RENUGA, B. S. LEE, N. M. DAT, N. T. AND LUKE, AND K. M. M. AUNG. **Virtual machine placement with two-path traffic routing for reduced congestion in Data Center networks.** *Computer Communications*, 2014. xi, 100, 116, 117, 118, 119, 120, 121, 137
- [10] K. FANG, S. AND RENUGA, B. S. LEE, C. H. FOH, AND K. M. M. AUNG. **Power-efficient virtual machine placement and migration in Data Centers.** In *Proc. of IEEE Greencom*, 2013. xi, 124, 133, 134, 137
- [11] HUAN LIU. **Amazon Data Center Size.** <http://huanliu.wordpress.com/2012/03/13/amazon-data-center-size/>, 2012. 3
- [12] YEVGENIY SVERDLIK. **Google Invests \$600 Million to Expand in South Carolina.** <http://www.datacenterknowledge.com/archives/2013/01/18/google-puts-another-600m-in-south-carolina/>. 3, 23
- [13] A. GREENBERG, J. HAMILTON, D.A. MALTZ, AND P. PATEL. **The cost of a cloud: research problems in data center networks.** *ACM SIGCOMM Computer Communication Review*, **39**(1):68–73, 2008. 3
- [14] RICH MILLER. **Facebooks \$1 Billion Data Center Network.** <http://www.datacenterknowledge.com/archives/2012/02/02/facebook-1-billion-data-center-network/>., 2012. 3
- [15] RICH MILLER. **Dimon: JPMorgan Spends \$500 Million per Data Center.** <http://www.datacenterknowledge.com/archives/2012/08/13/dimon-jpmorgan-spends-500-million-on-its-data-centers/>, 2012. 3
- [16] KEVIN McLAUGHLIN. **Microsoft Building \$500 Million Data Center In Virginia.** <http://www.crn.com/news/cloud/227200088/microsoft-building-500-million-data-center-in-virginia.html>, 2010. 3
- [17] HP. **Configuring HP Procurve.** <http://www.hp.com/rnd/support/manuals/5400z1.htm>. 3, 23, 25, 76
- [18] JUNIPER. **Virtualization and its impact on network.** <http://www.adtechglobal.com/Data/Sites/1/marketing/Virtualization-and-itsimpact-on-the-network.pdf>. 3
- [19] DAVID FLOYER. **VMware Dominant in Multi-Hypervisor Data Centers.** http://wikibon.org/wiki/v/VMware_Dominant_in_Multi-Hypervisor_Data_Centers, 2013. 3
- [20] S. KANDULA, S. SENGUPTA, A. GREENBERG, P. PATEL, AND R. CHAIKEN. **The nature of data center traffic: measurements & analysis.** In *Proc. of SIGCOMM conference on Internet measurement conference, IMC 2009*. ACM, 2009. 4, 23, 25, 26, 103

-
- [21] QI ZHANG, LU CHENG, AND RAOUF BOUTABA. **Cloud computing: state-of-the-art and research challenges**. *J. Internet Services and Applications*, pages 7–18, 2010. 4
- [22] DAVID ZATS, TATHAGATA DAS, PRASHANTH MOHAN, DHRUBA BORTHAKUR, AND RANDY KATZ. **DeTail: Reducing the Flow Completion Time Tail in Datacenter Networks**. pages 139–150, 2012. 4, 5
- [23] **IEEE Computer Society. Information technology - telecommunications and information exchange between systems local and metropolitan area networks common specifications part 3: Media access control bridges**. *IEEE Standard 802.1D*, 1998. 5, 26, 27
- [24] **IEEE. 802.1w - Rapid Reconfiguration of Spanning Tree**. Available at <http://www.ieee802.org/1/pages/802.1w.html>. 5
- [25] C.E. HOPPS. **RFC 2922-Analysis of an equal-cost multi-path algorithm**. Available at <http://tools.ietf.org/html/rfc2992>, 2000. 5, 16, 26
- [26] J. MOY. **RFC2328-OSPF Version 2**. Available at <http://tools.ietf.org/html/rfc1247>, 1998. 5
- [27] M. AL-FARES, S. RADHAKRISHNAN, B. RAGHAVAN, N. HUANG, AND A. VAHDAT. **Hedera: Dynamic flow scheduling for data center networks**. In *Proc. of USENIX Symposium on Networked Systems Design and Implementation, NSDI 2010*. USENIX, 2010. 5, 10, 11, 16, 22, 23, 27, 28, 55, 138
- [28] JIAXIN CAO, RUI XIA, PENGKUN YANG, CHUANXIONG GUO, GUOHAN LU, LIHUA YUAN, YIXIN ZHENG, HAITAO WU, YONGQIANG XIONG, AND DAVE MALTZ. **Per-packet Load-balanced, Low-latency Routing for Clos-based Data Center Networks**. In *Proc. of the Ninth ACM Conference on Emerging Networking Experiments and Technologies*, 2013. 5
- [29] IAN F. AKYILDIZ, AHYOUNG LEE, PU WANG, MIN LUO, AND WU CHOU. **A roadmap for traffic engineering in SDN-OpenFlow networks**. *Computer Networks*, 71:1–30, 2014. 5, 23, 26
- [30] A. GREENBERG, P. LAHIRI, D.A. MALTZ, P. PATEL, AND S. SENGUPTA. **Towards a next generation data center architecture: scalability and commoditization**. In *Proc. of the ACM workshop on Programmable Routers for Extensible Services of Tomorrow*. ACM, 2008. 6, 7
- [31] T. BENSON, A. ANAND, A. AKELLA, AND M. ZHANG. **Understanding data center traffic characteristics**. In *Proc. of ACM SIGCOMM Workshop: Research on Enterprise Networking*, pages 608–613, 2009. 6, 22, 26

REFERENCES

- [32] Y. ZHANG AND A. NIRWAN. **On Architecture Design, Congestion Notification, TCP Incast and Power Consumption in Data Centers.** *IEEE Communications surveys and Tutorials*, **15**(1), 2013. 6, 17
- [33] C. LEISERSON. **Fat-Trees: universal networks for hardware efficient supercomputing.** *IEEE Transactions on Computers*, **34**(10):892–901, 1985. 6, 17, 26, 115, 128
- [34] MEG WALRAED-SULLIVAN, KEITH MARZULLO, AND AMIN VAHDAT. **Scalability vs. Fault Tolerance in Aspen Trees.** <http://research.microsoft.com/apps/pubs/default.aspx?id=183910>, 2013. 6
- [35] PHILLIPA GILL, NAVENDU JAIN, AND NACHIAPPAN NAGAPPAN. **Understanding Network Failures in Data Centers: Measurement, Analysis, and Implications.** *SIGCOMM Comput. Commun. Rev.*, **41**(4):350–361, 2011. 6
- [36] KIRILL LEVCHENKO, GEOFFREY M. VOELKER, RAMAMOHAN PATURI, AND STEFAN SAVAGE. **XI: An Efficient Network Routing Algorithm.** *SIGCOMM Comput. Commun. Rev.*, **38**(4):15–26, 2008. 6
- [37] JUNDA LIU. **Data-driven network connectivity.** In *Proceedings of the 10th ACM Workshop on Hot Topics in Networks*. ACM, 2011. 6
- [38] KARTHIK LAKSHMINARAYANAN, MATTHEW CAESAR, MURALI RANGAN, TOM ANDERSON, SCOTT SHENKER, AND ION STOICA. **Achieving Convergence-free Routing Using Failure-carrying Packets.** *SIGCOMM Comput. Commun. Rev.*, **37**(4):241–252, 2007. 6
- [39] M. SINDELAR, R. K. SITARAMAN, AND P. SHENOY. **Sharing-aware algorithms for virtual machine colocation.** In *Proc. of Symposium on Parallelism in Algorithms and Architectures, SPAA 2011*. ACM, 2007. 6, 31, 99
- [40] N. BOBROFF, A. KOCHUT, AND A. BEATY. **Dynamic placement of virtual machines for managing SLA violations.** In *Proc. of IFIP/IEEE International Symposium on Integrated Network Management, IM 2007*. IFIP/IEEE, 2007. 6, 31, 99
- [41] C. ISCI, J. E. HANSON, AND I. WHALLEY ET. AL. **Runtime demand estimation for effective dynamic resource management.** In *Proc. of the International Conference on Network Operations and Management Symposium, NOMS 2010*. IEEE, 2010. 6, 31, 99
- [42] J. W. JIANG, CT. LAN, S. HA, M. CHEN, AND M. CHIANG. **Joint VM Placement and Routing for Data Center Traffic Engineering.** In *Proc. of IEEE INFOCOM'12*, 2012. 7, 11, 29, 31, 110, 117

-
- [43] KASHIF BILAL, SAIF UR REHMAN MALIK, OSMAN KHALID, ABDUL HAMEED, ENRIQUE ALVAREZ, VIDURA WIJAYSEKARA, RIZWANA IRFAN, SARJAN SHRESTHA, DEBJYOTI DWIVEDY, MAZHAR ALI, USMAN SHAHID KHAN, ASSAD ABBAS, NAUMAN JALIL, AND SAMEE ULLAH KHAN. **A taxonomy and survey on Green Data Center Networks.** *Future Generation Comp. Syst.*, **36**:189–208, 2014. 7, 20, 21, 29, 31, 32
- [44] V. MANN, A. KUMAR, P. DUTTA, AND S. KALYANARAMAN. **VMFlow: leveraging VM mobility to reduce network power costs in data centers.** In *Proc. of IFIP conference on Networking'11*, 2011. 7, 12, 29, 31, 33
- [45] TSAKALOZOS, KONSTANTINOS, ROUSSOPOULO, MEMA, DELIS, AND ALEX. **VM Placement in non-Homogeneous IaaS-Clouds.** *ICSOC, Lecture Notes in Computer Science*, **7084**(2):172–187, 2011. 7, 29, 31, 32
- [46] X. MENG, V. PAPPAS, AND L. ZHANG. **Improving the scalability of data center networks with traffic-aware virtual machine placemen.** In *Proc. of of IEEE INFOCOM'10*, 2010. 7, 11, 32, 33, 34, 99, 105
- [47] GUOHUI WANG AND T. S. EUGENE NG. **The Impact of Virtualization on Network Performance of Amazon EC2 Data Center.** In *Proc. of INFOCOM*, 2010. 7, 31
- [48] A. GREENBERG, J.R. HAMILTON, N. JAIN, S. KANDULA, C. KIM, P. LAHIRI, D.A. MALTZ, P. PATEL, AND S. SENGUPTA. **VL2: A Scalable and Flexible Data Center Network.** In *Proc. of SIGCOMM 2009 conference on Data Communication*. ACM, 2009. 7, 16, 17, 19, 25, 26, 32
- [49] M. AL-FARES, A. LOUKISSAS, AND A. VAHDAT. **A scalable, commodity data center network architecture.** In *Proc. of SIGCOMM 2008 conference on Data Communication*. ACM, 2008. 7, 16, 17, 18, 22, 40, 47
- [50] N. MCKEOWN, T. ANDERSON, H. BALAKRISHNAN, G. PARULKAR, L. PETERSON, J. REXFORD, S. SHENKER, AND J. TURNER. **OpenFlow: enabling innovation in campus networks.** *SIGCOMM Computer Communication Review*, **38**(2):69–74, 2008. 7, 23, 25, 28, 103
- [51] A. TAVAKOLI, M. CASADO, T. KOPONEN, , AND S. SHENKER. **Applying NOX to Data center.** In *Proc. of SIGCOMM Workshop on Hot Topics in Networks, HotNets 2009*. ACM, 2009. 7, 23, 40, 56, 73, 103
- [52] HP. **HP ProCurve 5400 zl switch series.** http://h17007.www1.hp.com/us/en/products/switches/HP_E5400_zl_Switch_Series/index.aspx. 7
- [53] JEFFREY C. MOGUL AND PAUL CONGDON. **Hey, You Darned Counters!: Get off My ASIC!** In *Proc. of the First Workshop on Hot Topics in Software Defined Networks*, 2012. 7

REFERENCES

- [54] P. MAHADEVAN, P. SHARMA, S. BANERJEE, AND P. RANGANATHAN. **A power benchmarking framework for network devices**. In *Proc. of the International IFIP Networking Conference. NETWORKING 2009*. IFIP, 2009. 7, 25
- [55] AVAYA INC. **Compare and contrast spb and trill**. Available at <http://www.avaya.com/uk/resource/assets/whitepapers/SPB-TRILL-Compare-Contrast-DN4634.pdf>, 2010. 10, 27, 37
- [56] C. KIM, M. CAESAR, AND J. REXFORD. **Floodless in seattle: a scalable ethernet architecture for large enterprises**. In *Proc. of SIGCOMM 2008 conference on Data Communication*. ACM, 2008. 10, 22, 27
- [57] J. MUDIGONDA, P. YALAGANDULA, M. AL-FARES, AND J.C. MOGUL. **SPAIN: COTS data-center Ethernet for multipathing over arbitrary topologies**. In *Proc. of USENIX Symposium on Networked Systems Design and Implementation, NSDI 2010*. USENIX, 2010. 10, 26, 27, 37
- [58] THEOPHILUS BENSON, ASHOK ANAND, ADITYA AKELLA, AND MING ZHANG. **MicroTE: Fine Grained Traffic Engineering for Data Centers**. In *Proc. of the Seventh COnference on Emerging Networking EXperiments and Technologies*. ACM, 2011. 10, 11, 23, 28, 55
- [59] Z. ZHAO, T. QIN, F. XU, R. CAO, X. LI, AND G. WANG. **CAWRM: A Remote Mirroring System Based on AoDI Volume**. In *Proc. of IEEE conference on Dependable Systems and Networks Workshops*. IEEE, 2001. 10
- [60] H. WEATHERSPOON, G. LAKSHMI, T. MARIAN, B. MAHESH, AND K. BIRMAN. **Smoke and Mirrors: Reflecting Files at a Geographically Remote Location without Loss of Performance**. In *Proc. of 7th USENIX Conference on File and Storage Technologies*. USENIX, 2009. 10, 71
- [61] M. JI, A. VEITCH, AND J. WILKES. **Seneca: remote mirroring done write**. In *Proc. of USENIX Annual Technical conference*. USENIX, 2003. 10, 71
- [62] H. PATTERSON, S. MANLEY, M. FEDERWISCH, D. HITZ, S. KLEIMAN, AND S. OWARA. **SnapMirror: File system based asynchronous mirroring for disaster recovery**. In *Proc. of USENIX Conference on File and Storage Technologies*. USENIX, 2002. 10, 71
- [63] DOTHILL. **Secure data protection with dot hills batch remote replication**. *Dot Hill's White paper*, pages 1–8, 2011. 10, 71
- [64] M. ZADNIK, M. CANINI. **Evolution of cache replacement policies to track heavy-hitter flows**. In *Proc. of PAM*. IEEE, 2011. 11

-
- [65] T. PAN, X. GUO, C. ZHANG, W. MENG, AND B. LIU. **ALFE: A Replacement policy to cache elephant flows in the presence of mice flooding.** In *Proc. of IEEE ICC*. IEEE, 2012. 11, 25, 83
- [66] M. YU, J. REXFORD, M. J. FREEDMAN, AND J. WANG. **Scalable flow-based networking with DIFANE.** In *Proc. of SIGCOMM*. ACM, 2010. 11
- [67] F. SONG, D. HUANG, H. ZHOU, AND I. YOU. **Application-aware virtual machine placement in Data Centers.** In *Proc. of IEEE International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS'12)*, 2012. 11, 30, 32
- [68] PRIYA MAHADEVAN, SUJATA BANERJEE, PUNEET SHARMA, AMIP SHAH, AND PARTHASARATHY RANGANATHAN. **On energy efficiency for enterprise and data center networks.** *IEEE Communications Magazine*, **49**(8):94–100, 2011. 12, 33
- [69] B. HELLER, S. SEETHARAMAN, P. MAHADEVAN, Y. YIAKOUMIS, P. SHARMA, S. BANERJEE, AND N. McKEOWN. **ElasticTree: Saving energy in data center networks.** In *Proc. of USENIX symposium on Networked Systems Design and Implementation, NSDI 2010*. USENIX, 2010. 12, 16, 33, 138
- [70] LUCA CHIARAVIGLIO, WEIWEI FANG, XIANGMIN LIANG, SHENGXIN LI, AND NAIKUE XIONG. **VMPlanner: Optimizing Virtual Machine Placement and Traffic Flow Routing to Reduce Network Power Costs in Cloud Data Centers.** *Computer Networks*, 2012. 12, 33, 34
- [71] CISCO. **Data center: Load balancing data center services.** 2004. 15, 22
- [72] R. NIRANJAN MYSORE, A. PAMBORIS, N. FARRINGTON, N. HUANG, P. MIRI, S. RADHAKRISHNAN, V. SUBRAMANYA, AND A. VAHDAT. **PortLand: a scalable fault-tolerant layer 2 data center network fabric.** In *Proc. of SIGCOMM 2009 conference on Data Communication*. ACM, 2009. 16, 17, 19, 20, 22, 27, 37
- [73] C. CLOS. **A study of non-blocking switching networks.** *Bell System Technical Journal*, **32**(2):406–424, 1953. 17, 19, 26, 115
- [74] RUI ZHANG-SHEN AND NICK McKEOWN. **Designing a Predictable Internet Backbone with Valiant Load-balancing.** In *Proc. of the 13th International Conference on Quality of Service*. Springer-Verlag, 2005. 19
- [75] MD. FAIZUL BARI, RAOUF BOUTABA, RAFAEL PEREIRA ESTEVES, LISANDRO ZAMBENEDETTI GRANVILLE, MAXIM PODLESNY, MD. GOLAM RABBANI, QI ZHANG, AND MOHAMED FATEN ZHANI. **Data Center Network Virtualization: A Survey.** *IEEE Communications Surveys and Tutorials*, **15**(2):909–928, 2013. 19, 20, 26, 27
- [76] J. HAMILTON. **Architecture for modular data centers.** In *Proc. of 3rd Biennial Conference on Innovative Data Systems Research (CIDR)*, 2007. 20, 22

REFERENCES

- [77] K. V. VISHWANATH, A. GREENBERG, AND D.A. REED. **Modular Data Centers: How to design them.** In *Proc. of ACM workshop on Large- Scale System and Application Performance, (LSAP 09)*, 2009. 20
- [78] M. M. WALDROP. **Data center in a box.** *Scientific American*, **297**(2):90–93, 2007. 20
- [79] C. GUO, G. LU, H. J. WANG, S. YANG, C. KONG, P. SUN, W. WU, AND Y. ZHANG. **SecondNet: a data center network virtualization architecture with bandwidth guarantees.** In *Proc. of ACM Co-NEXT’10*, 2010. 21
- [80] FAN YAO, JINGXIN WU, GURU VENKATARAMANI, AND SURESH SUBRAMANIAM. **A Comparative Analysis of Data Center Network Architectures.** In *Proc. of IEEE International Conference on Communications - Next Generation Networking*, 2014. 22
- [81] TOP500. **Home — TOP500 Supercomputer Sites.** 2013. 22
- [82] MALATHI VEERARAGHAVAN YANG LIU, JOGESH K. MUPPALA. **A Survey of Data Center Network Architectures.** <http://www.ece.virginia.edu/mv/pubs/recent-samples/Data-center-Survey.pdf>. 22
- [83] OPEN NETWORKING FOUNDATION. **Software-Defined Networking: The new norm for networks.** In *white paper*. Open Networking Foundation., 2012. 23
- [84] OPEN NETWORKING FOUNDATION. **Open Networking Foundation.** <https://www.opennetworking.org/>, 2013. 23
- [85] OPEN NETWORKING FOUNDATION. **OpenFlow switch specification v1.0v1.4.** <http://www.openflowswitch.org/sdn-resources/onf-specifications>, 2008. 23, 24, 92, 93
- [86] T. BENSON, A. ANAND, A. AKELLA, AND M. ZHANG. **The case for fine-grained traffic engineering in data centers.** In *Proc. of Internet network management conference on Research on enterprise networking*, pages 2–3. USENIX Association, 2010. 25, 26, 92
- [87] A. R. CURTIS. **Mahout:Low overhead datacenter traffic Management using end-host based elephant detection.** In *Proc. of IEEE INFOCOMM*. IEEE, 2011. 25, 83, 87
- [88] NING WANG, KIN HON HO, GEORGE PAVLOU, AND MICHAEL HOWARTH. **AN OVERVIEW OF ROUTING OPTIMIZATION FOR INTERNET TRAFFIC ENGINEERING.** *IEEE COMMUNICATIONS SURVEYS - THE ELECTRONIC MAGAZINE OF ORIGINAL PEER-REVIEWED SURVEY ARTICLES*, **10**(1-4):36–56, 2008. 26

-
- [89] HAO WANG, HAIYONG XIE, LILI QIU, YANG RICHARD YANG, YIN ZHANG, AND ALBERT GREENBERG. **COPE: Traffic Engineering in Dynamic Networks.** *SIGCOMM Comput. Commun. Rev.*, **36**(4):99–110, 2006. 26
 - [90] CHRISTOPHE BETOULE, THOMAS BONALD, REMI CLAVIER, DARIO ROSSI, GIUSEPPE ROSSINI, AND GILLES THOUENON. **Adaptive Probabilistic Flooding for Multipath Routing.** *CoRR*, abs/1104.5636, 2011. 27
 - [91] FUNG PO TSO AND DIMITRIOS P. PEZAROS. **Improving Data Center Network Utilization Using Near-Optimal Traffic Engineering.** *IEEE Transactions on Parallel and Distributed Systems*, **24**(6):1139–1148, 2013. 28, 55
 - [92] DENG PAN YU LI. **OpenFlow based load balancing for Fat-Tree networks with multipath support.** In *Proc. of International conference of communications*, 2013. 28
 - [93] HAMID HAJABDOLALI BAZZAZ, MALVEEKA TEWARI, GUOHUI WANG, GEORGE PORTER, T. S. EUGENE NG, DAVID G. ANDERSEN, MICHAEL KAMINSKY, MICHAEL A. KOZUCH, AND AMIN VAHDAT. **Switching the Optical Divide: Fundamental Challenges for Hybrid Electrical/Optical Datacenter Networks.** In *Proc. of the 2Nd ACM Symposium on Cloud Computing*. ACM, 2011. 28, 55
 - [94] O. BIRAN, A. CORRADI, M. FANELLI, L. FOSCHINI, A. NUS, D. RAZ, AND E. SILVERA. **A stable network-aware VM placement for cloud systems.** In *Proc. of IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid’12)*, 2012. 32, 99, 110
 - [95] VIJAY MANN, AKANKSHA GUPTA, PARTHA DUTTA, ANILKUMAR VISHNOI, PARANTAPA BHATTACHARYA, RISHABH PODDAR, AND AAKASH IYER. **Remedy: Network-Aware Steady State VM Management for Data Centers.** In *NETWORKING 2012 - 11th International IFIP TC 6 Networking Conference, Prague, Czech Republic, May 21-25, 2012, Proceedings, Part I*, 2012. 32
 - [96] DAVID MEISNER, BRIAN T. GOLD, AND THOMAS F. WENISCH. **PowerNap: Eliminating Server Idle Power.** *SIGARCH Comput. Archit. News*, **37**(1):205–216, 2009. 33
 - [97] C.E. BASH, C.D. PATEL, AND R.K. SHARMA. **Dynamic thermal management of air cooled data centers.** In *Proc. of Thermal and Thermomechanical Phenomena in Electronics Systems*. ITherm, 2006. 33
 - [98] D. KLIASOVICH, P. BOUVRY, Y. AUDZEVICH, AND S. KHAN. **Greencloud: a packet-level simulator of energy-aware cloud computing data centers.** In *Global Telecommunications Conference*. IEEE, 2010. 33
 - [99] A. P. BIANZINO, F. CHAUDET, C. AND LARROCA, D. ROSSI, AND J. ROUGIER. **Energy-aware routing: A reality check.** In *Proc. of Globecom workshops*, 2010. 33

REFERENCES

- [100] RAFFAELE BOLLA, ROBERTO BRUSCHI, FRANCO DAVOLI, AND FLAVIO CUCCHIETTI. **Energy Efficiency in the Future Internet: A Survey of Existing Approaches and Trends in Energy-Aware Fixed Network Infrastructures.** *IEEE Communications Surveys and Tutorials*, **13**(2):223–244, 2011. 33
- [101] Y. SHIRAISHI, M. MOHRI, , AND Y. FUKUTA. **A server-aided computation protocol revisited for confidentiality of cloud service.** *Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications*, **2**(2):8394, 2011. 33
- [102] YUNFEI SHANG, DAN LI, AND MINGWEI XU. **Energy-aware Routing in Data Center Network.** In *Proc. of the First ACM SIGCOMM Workshop on Green Networking*, 2010. 33
- [103] M. SEAMAN. **IEEE 802.1aq shortest path bridging.** Available at <http://www.ieee802.org/1/pages/802.1aq.html>, 2006. 37
- [104] ANDRÁS VARGA AND RUDOLF HORNIG. **OMNeT++.** <http://www.omnetpp.org>. 47, 66
- [105] 3COM CORPORATION. **Switch 5500G 10/100/1000 family data sheet.** <http://www.3com.com/other/pdfs/products/enUS/400908.pdf>, 2009. 63
- [106] EMC. **Zero data loss solutions for extended distance replication.** *EMC Tech paper*, 2010. 71
- [107] SYMANTEC. **VERITAS Volume Replicator.** *Symantec Technical paper*, 2009. 71
- [108] IBM. **Enhanced remote mirroring.** *IBM Tech paper*, 2009. 71
- [109] M. HERMAN. **Multi-node mirrored NVRAM in a virtualized environment.** *DSpace@MIT*, 2011. 71
- [110] Z. MING, Y. LIU, AND Q. K. YANG. **Cost-Effective Remote Mirroring Using the iSCSI Protocol.** In *Proc. of IEEE Mass Storage Systems*. IEEE, 2004. 71
- [111] MANISH MARWAH, SHIVAKANT MISHRA, AND CHRISTOF FETZER. **TCP Server Fault Tolerance Using Connection Migration to a Backup Server.** In *International Conference on Dependable Systems and Networks (DSN 2003)*. IEEE, 2003. 71
- [112] EMMANUEL CECCHET, GEORGE CANDEA, AND ANASTASIA AILAMAKI. **Middleware-based Database Replication: The Gaps Between Theory and Practice.** In *Proc. of the 2008 ACM SIGMOD International Conference on Management of Data*, SIGMOD '08. ACM, 2008. 71
- [113] INTEL. **IOMeter Users Guide. Version 2003.12.16.** <Http://www.iometer.org>, 2003. 73, 77

-
- [114] R. FAGIN, J. NIEVERGELT, N. PIPPENGER, AND R. STRONG. **Extendible hashing: a fast access method for dynamic files.** *ACM Trans. Database Syst.*, 1979. 85
- [115] **Secure hash standard.** <http://csrc.nist.gov/publications/fips/fips180-2/fips180-2.pdf>, 1995. 88, 89
- [116] GOOGLE. **Ostinato Packet/traffic generator and analyzer.** <http://code.google.com/p/ostinato/>. 92
- [117] D. ERSOZ, M. S. YOUSIF, AND C. R. DAS. **Characterizing network traffic in a cluster-based multi-tier Data Center.** In *Proc. of ICDCS'07*, 2007. 103
- [118] Y. CHEN, A. S. GANAPATHI, R. GRIFFITH, AND R. H. KATZ. **Analysis and lessons from a publicly available google cluster trace.** *Technical Report*, 2010. 103
- [119] MINGHUA CHEN, SOUNG CHANG LIEW, ZIYU SHAO, AND CAIHONG KAI. **Markov Approximation for Combinatorial Network Optimization.** *IEEE Transactions on Information Theory*, **59**(10):6301–6327, 2013. 110
- [120] T. R. HENDERSON, M. LACAGE, AND G. F. RILEY. **Network simulations with the ns-3 simulator.** In *Proc. of ACM SIGCOMM*, 2008. 115
- [121] **The ns-3 network simulator.** <http://www.nsnam.org/>. 115
- [122] T. BABASAKI, T. TANAKA, Y. NOZAKI, T. AOKI, AND F. KUROKAWA. **Developing of higher voltage direct-current power-feeding prototype system.** In *Proc. of IEEE Telecommunications Energy Conference*, 2009. 123
- [123] J. RAJI. **Evolving towards the GREEN Data Center.** 2009. 123
- [124] J. KOOMEY. **Estimating total power consumption by servers in the us and the world.** *Final Report*, 2007. 123
- [125] R.B. JOHN AND DELL SERVER PERFORMANCE ANALYSIS TEAM. **Power Efficiency Comparison of Enterprise-Class Blade Servers and Enclosures.** *Technical White Paper*, 2007. 123
- [126] EDISON GROUP. **Blade Server Power Study.** *Report*, 2007. 123
- [127] S. FANG, H. LI, C. FOH, Y. WEN, AND K. AUNG. **Energy Optimizations for Data Center Network: Formulation and its Solution.** In *Proc. of IEEE Globecom'12*, 2012. 123
- [128] BINZHOU XIA AND ZHIYI TAN. **Tighter Bounds of the First Fit Algorithm for the Bin-packing Problem.** *Discrete Appl. Math.*, **158**(15):1668–1675, 2010. 124, 127

REFERENCES

- [129] E. G. COFFMAN, JR., M. R. GAREY, AND D. S. JOHNSON. **Approximation Algorithms for Bin Packing: A Survey**. pages 46–93, 1997. 124
- [130] C. CLARK, S. FRASER, J. HANSEN, C. JUL, E. LIMPACH, I. PRATT, AND A. WARFIELD. **Live migration of virtual machines**. In *Proc. of USENIX Symposium on Networked Systems Design and Implementation*, 2005. 129
- [131] THOMAS LIN, JOON-MYUNG KANG, HADI BANNAZADEH, AND ALBERTO LEON GARCIA. **Enabling SDN Applications on Software-Defined Infrastructure**. In *IEEE/IFIP Network Operations and Management Symposium*, 2014. 138
- [132] H. BALLANI, P. COSTA, T. KARAGIANNIS, AND A. ROWSTRON. **Towards predictable datacenter networks**. In *Proc. of ACM SIGCOMM*, pages 242–253, 2011. 139
- [133] J. LEE, M. LEE, L. POPA, Y. TURNER, S. BANERJEE, P. SHARMA, AND B. STEPHENSON. **CloudMirror: Application-Aware Bandwidth Reservations in the Cloud**. In *Proc. of USENIX HotCloud*, 2013. 139
- [134] K. CHRISTENSEN, P. REVIRIEGO, B. NORDMAN, M. BENNETT, M. MOSTOWFI, AND J.A. MAESTRO. **IEEE 802.3az: the Road to Energy Efficient Ethernet**. *IEEE Communications Magazine*, 48(11):50–56, 2010. 140