4-4, 1-4: A novel architecture for data center networks and its performance study

Thesis submitted in partial fulfilment of the requirements for the award of the degree of

Doctor of Philosophy

in

Computer Science and Engineering

by

Ashok Kumar A. R.

Under the supervision of

Prof. S. V. Rao
Prof. Diganta Goswami

Department of Computer Science and Engineering
Indian Institute of Technology Guwahati
Guwahati - 781039 Assam India

July 2016

Declaration of Authorship

I, **Ashok Kumar A R**, declare that this thesis titled, '*4-4,1-4: A novel architecture for data center networks and its performance study*' and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this Institute.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this Institute or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: _____

Date: _____

Department of Computer Science and Engineering
Indian Institute of Technology Guwahati
Guwahati - 781039 Assam India

Prof. Diganta Goswami
Professor
0361-2582355
dgoswami@iitg.ernet.in

Prof. S. V. Rao
Professor
0361-2582358
svrao@iitg.ernet.in

Certificate

This is to certify that the thesis entitled **4-4,1-4: A novel architecture for data center networks and its performance study** being submitted by **Ashok Kumar A. R.** to the Department of Computer Science and Engineering, Indian Institute of Technology Guwahati, is a record of bona fide research work under my supervision and is worthy of consideration for the award of the degree of Doctor of Philosophy of the Institute.

Date: 8^{th} July 2016
Place: Guwahati

Prof. Diganta Goswami

Prof. S. V. Rao

Acknowledgements

I would like to thank everyone who supported and assisted me during my doctoral program at IIT Guwahati. First and foremost, I would like to thank my research advisors, Prof. S. V. Rao and Prof. Dignata Goswami for their advice and support during the Ph.D. program. An innumerable discussions and their valuable advices/suggestions throughout the research helped me in many fronts of the research. Their in depth knowledge in research were the last resort for solving many of the problems I faced during research. I learned a lot from them on both technical and social aspects of life.

I would like to thank Prof. Dignata Goswami, Head of the Department, Computer Science and Engineering and also my supervisor, for providing necessary facilities in the department. I would also like to thank other members of the department, Bhriguraj Borah, Nanu Alan Kachari, Nava Kumar Boro and Raktajit Pathak who helped me in utilizing the resources of the department. The warmth with which they serve the needs of researchers is unforgettable.

I wish to place on record my sincere thanks to my doctoral committee members, Dr. T. Venkatesh, Dr. Sanasam Ranbir Singh and Dr. Partha Sarathi Mandal for constantly monitoring my progress and giving their valuable guidance. Each one of them, expert in their own domain, provided a proper direction for my research.

I am also grateful to all my friends in the lab Niladri, Shounak, Nilkanta, Shirshendu, Suddhasil, Mayank and many others who made this journey of research a memorable. I would also like to thank HAB for providing Quarter in Married Scholar hostel. The stay in Married Scholar is one the best moment in our life. A special thanks to all our neighbors there.

Last, but not the least, I would like to thank my parents, my in-laws, my brother in-law, my brother and all my relatives for their constant support and encouragement during the course of study. My wife Usha has been of great support during my research. She relinquished me from all the house hold responsibilities and allowed me to concentrate on my research. Thank you very much Usha. How I can end this list without mentioning special thanks to my sweet daughter, Adithi, who is a source of joy in our life.

Abstract

The advancements in the Internet technologies, changed the service delivery model from in house delivery to delivery through Cloud. This cost effective service delivery created a greater demand for cloud and thus the cloud infrastructure. The major cloud service providers such as Google, IBM, Microsoft use data centers as central resource for their cloud computing. Therefore, data center is defined as a central repository of computing, storage and networking for storing and processing large data and information that can be accessed globally. With data center hosting millions of servers, it faces challenges such as reliability, availability, maintainability and safety. One of the major challenges in the design of data center is to combine very large number of servers into a single network fabric called data center network (DCN) and design protocols that addresses the growing needs of data centers.

The major advantages in the design of protocols for DCNs are due the proprietorship of data centers. Since, data centers are private, they can have more controlled structure for the DCNs and often does not face interoperability problem. This leads to many designs proposed for DCNs addressing various aspects of data centers.

In our first contribution, we propose a new architecture for DCN called 4-4, 1-4, based on IP address hierarchy. The design 4-4, 1-4 follows hierarchy in both topology design and address assignment. Thus, by correlating topology design and address assignment, the location of the end host is directly determined from its address. This feature helps to devise a location based routing for 4-4, 1-4 architecture which covers entire path of a route using only the location information.

Energy conservation is an important aspect of DCN. In our second contribution, we show competence of 4-4, 1-4 in conserving energy. For this, we propose a distributed greedy algorithm that controls the resource usage based on current traffic. Here, the architecture features of two different designs, 4-4, 1-4 and fat-tree are considered for the study of energy.

The design principle of DCN is to provide a greater bandwidth through a larger number of interconnections. These large interconnections provide multiple paths between the pair of servers. In our third contribution, we propose a packet scheduler for 4-4, 1-4 that does the load balancing by prioritizing and scheduling the packets of a flow at the network layer. The proposed scheduler uses the path

diversity provided by 4-4, 1-4 to schedule the flows that increases the chances for more flows to meet their deadlines.

The existing designs for DCNs follow subnetting and supernetting principle for their topology design. In our last contribution, we use the hierarchical feature of 4-4, 1-4 to propose a simple encoding scheme that uses only the IP addresses for both identifying end hosts and routing. This overcomes the limitations of the earlier proposed designs for source routing which use two different address spaces for the same.

Contents

List of Figures

List of Tables

Abbreviations

DCN	Data Center Networks.
OSPF	Open Shortest Path First.
DHCP	Dynamic Host Configuration Protocol.
RIP	Routing Information Protocol.
IGP	Interior Gateway Protocol.
BGP	Border Gateway Protocol.
MTU	Maximum Transmission Unit.
MIN	Multistage Interconnection Networks.
TCP	Transmission Control Protocol.
MSC	Modular Server Centric designs.
LDP	Location Discovery Protocol.
PMAC	Pseudo Medium Access Control.
AMAC	Actual Medium Access Control.
TVM	Topology View Message.
DCA	Decider/Chooser abstraction.
VCI	Vitrual Circuit Interconnection.
LDP	Lightweight Directory Access Protocol.
SNMP	Simple Network Management Protocols.
SMB	Server Message Block.
LDAP	Light Weight Directory Access Protocol.
AFS	Apple filing protocol.
NCP	Netware Core Protocol.
VLB	Valiant Load Balancing.
ECMP	Equal Cost Multi Path.

AIMD	Additive Increase Multiplicative Decrease.
ECN	Explicit Congestion Notification.
CE	Congestion Experienced.
TLV	Type Length Value.
ARP	Address Resolution Protocol.
LISP	Locator/ID separation protocols.
EID	Endpoint Identifier.
RLOC	Routing Locators.
CIDR	Classless Inter Domain Routing.
VLSM	Variable Length Subnet Mask.
DW	Data Warehouse.
ETLMR	Extract, Transform and Load using MapReduce.
DFS	Distributed File System.
SLA	Service Level Agreement.
HOL	Head Of Line.
VOQ	Virtual Output Queue.

Dedicated to the Almighty, my parents, my dear wife Usha and my sweet daughter Adithi.

Chapter 1

Introduction

Public and private enterprises are looking for an alternative to reduce Information Technology (IT) budget and to provide agile IT services to organizations, citizens and constituents. New paradigm, cloud computing, offers means to address issues of budget constraints and agility of services. Cloud computing is a model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction [1]. Thus, cloud computing can be understood as - the resources and services that are abstracted from the underlying infrastructure and provided on-demand, at scale and in a multitenant environment [2, 3].

Data centers are the infrastructure for cloud computing. The original concept of cloud is to the sell excess computing capacities in a distributed system as virtualized resources to anyone else. But, the cost effectiveness of cloud attracted a large number of applications such as MapReduce [4], Dryad [5], cluster based file systems, query to web search engine, parallel scientific applications and many other data and compute intensive jobs. This resulted in the higher volume of cloud traffic [6–9]. In this manner, growth in both the number of applications deployed on cloud and number of people using these applications resulted in changing infrastructure for cloud from small server farms to central factory hosting a large

collection of servers called data centers. Companies like Amazon, Rackspace, Microsoft, Google and others provide cloud services using their large data centers setup across the globe. Therefore, a data center is defined as a central repository of computing, storage and networking for storing and processing large data and information that can be accessed globally. Thus, one of the major challenge in the design of data center is ability to combine all these servers into a single manageable network fabric called data center networks (DCNs).

1.1 DCN architectures

The applications deployed on data centers often process a large volume of data typically in terabytes or petabytes size and have quick target response time to meet the service level agreements (SLA). Thus, these applications require more computing power than available with the conventional servers. The two choices of providing larger computing power are, either to use a single, super high performance and highly reliable supercomputer or a cluster of servers built out of thousands of cheap and unreliable personal commodity-off-the-shelf (COTS) hardware components. In a present scenario, with the advancement in microprocessor technology and speed of communication, clusters are considered to be a viable option compared to a single super computer.

Present data center networks relay upon two important mechanisms of interconnections. First, using specialized hardware and communication protocols such as InfiniBand or Myrinet and the second, using commodity Ethernet switches and routers. The InfiniBand architecture (IBA) is proposed to overcome the problem of using busses for interconnections [10]. In IBA, problems such as sharing of busses and failure to provide a high level of reliability and availability are overcomed by using a point to point interconnections. The point to point interconnections helps to overcome issues with busses such as arbitration and scaling. Myrinet on the other hand, is a low cost switching technology that is widely used to interconnect cluster of servers [11]. Myrinet provides a reliable communication by providing low latency and high throughput switches and links that are continuously monitored

for flow control and error control. Further, Myrinet to achieve low latency, provides a firmware in network interface cards that enable processes running on hosts to communicate directly with the network by passing the host operating system.

The InfiniBand and Myrinet are the scalable designs which can support clusters of thousands of node with high bandwidth and low latency. But, on the down side, these technologies require specialized hardware that are expensive and new protocols that are incompatible with TCP/IP applications. Thus, the cost effective alternative solution of building a DCN with the COTS hardware component is considered as a viable option. Here, we consider an evolution of DCN architectures starting from an earlier Cisco recommended a basic layered design for DCN [12].

1.1.1 A basic layered design

A basic layer design for data center network supports thousands of servers connected through two or three levels of switches [12]. At the base level, the servers are divided into different groups and are connected to the switches called top of the rock (TOR) switches. This base level is often called edge level in DCN literature. The TOR switches are then connected to the switches at a next level called aggregation layer. The aggregation layer is often made up of one or two rows of switches that are connected through a large number of interconnections. These interconnections provide a very large bandwidth that is sufficient enough to handle the traffic that is generated by the underlying hosts at any instance of time. Mostly, when all the end hosts communicate with their line speed of interconnections. Finally, the aggregate level switches are connected to the core switches at root level. The switches at the core level act as a gateway for data centers for the outside world. The schematic diagram for the basic layered design for DCN is shown in Fig.1.1.

The major limitations with the basic layered design for DCN is use of minimum number of switches at the higher level compared to the number of switches used at the lower levels. A disproportionate number of switches used at the higher

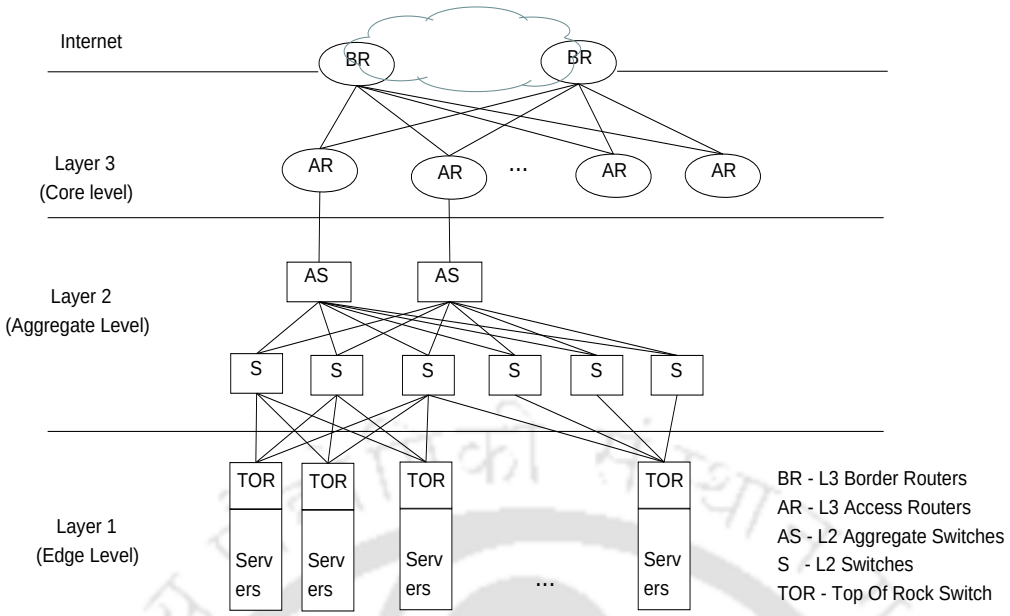


FIGURE 1.1: A basic layered design for DCN as proposed by Cisco data center infrastructure 2.5 design guide

level increases the congestion of the traffic at that level and results in reducing the overall performance of the system. Moreover, a failure of one or two switches at the higher level oversubscribe the remaining switches and prevent large number of clients from accessing the data centers. The oversubscription is defined as a ratio of worst case achievable aggregate bandwidth among end hosts to the total bisection bandwidth of the network topology. In a general a sense, the oversubscription measures the ability of hosts to fully utilize their uplink capacity [13]. Here, the oversubscription of a general tree is in the range from 2.5:1 to 8:1 [12]. Thus, the main motive behind designing architectures for DCN is to provide an oversubscription 1:1.

1.1.2 The evaluation of DCN architectures

Fat-tree [13] is one of the initial design to provide 1:1 oversubscription. An *s*-ary fat-tree is a three layer topology - an edge, an aggregation and a core level. Fat-tree, by expanding a core level into a multi-rooted tree, provides a bandwidth sufficient to handle the traffic generated when all the end hosts communicate with their line speed of interconnection. For providing this larger bandwidth, a large

number of interconnections are used between the successive levels of fat-tree. In fat-tree, there are $\frac{s}{2}$ parallel paths between the switches at successive levels of fat-tree. Further, the use of COTS switches at all the levels inspired other designs to provide the required bandwidth through a large number of interconnections [14–17].

The majority of the applications hosted on data centers demand better performance from the underlying architectures. These designs to process the requests in a timely manner, use MapReduce [4] programming model wherein the overall task is divided into a series of sub-tasks and are assigned to the nodes called worker nodes. Therefore, MapReduce nature of the deployed applications often required many server tiers for their operation and quality of their final result depend on the timely execution of constituent operations. A group of servers hosting an inverted index used for web search is one such application. As been reported, a sub-half-second response to an ordinary Google search query involves 700 to 1,000 servers [18]. Thus, efficiency in routing greatly influences the overall performance of the data center.

Data centers can either use layer 2 or layer 3 network protocols defined for Internet. The layer 2 protocol is a data link layer protocol that works on a single physical subnet such as local area networks (LANs). It consists of two layers, logical link control (LLC) layer and the media access control (MAC) layer. The MAC layer defines how the computers on the network gains control to access the data and to transmit the data over the medium. Whereas, the LLC layer provides additional mechanism for data transfer such as frame synchronization, flow control and error checking. On the other hand, layer 3 protocols are used to transfer the data across the LANs, where forwarding is done based on IP address of the end hosts.

There is a significant difference between the characteristics of Internet and DCN topologies. First, since DCNs are private networks and are often owned by a single entity, they can have more controlled structure. Compare to this, Internet is public network and the protocols proposed for Internet must address scalability and interoperability problems. The scalability in Internet is achieved by using

the hierarchical addressing and routing protocols [19]. The hierarchical addressing helps to define the subnets of different sizes using subnetting and supernetting and helps in solving the address assignments for millions of hosts an Internet supports. Next, to make the routing protocols scalable, Internet defines routing areas where routes between the routing areas are exchanged using hierarchical protocols such as interior gateway protocol (IGP) and border gateway protocol (BGP). Further, do deal with the interoperability problem such as variation in underlying links maximum transmission unit (MTU), the Internet protocols defines fragmentation and reassembly of the packets.

All the routing protocols proposed for Internet are not conducive for DCNs because constraints the underlying network offers are significantly different from DCNs. First, the DCNs do not have stringent interoperability problems. Second, the Layer 2 MAC protocols and Layer 3 routing protocols such as Open Shortest Path First (OSPF) and Routing Information Protocol (RIP) do not scale for a large number of servers that present DCNs support [20]. Further, it is not endorsed to complicate the designs by deploying hierarchical protocols such as interior gateway protocol (IGP) and border gateway protocol (BGP) which introduce large control overhead in terms of route exchanges and additional topology management by defining routing areas.

On the other hand, certain differences turn out to be advantageous for DCN's. Specially, the proprietorship of the underlying topology. Since, the DCN's are private and owned by a single administrative domain, they can have very controlled structure compared to Internet and do not have any interoperability problem. There are many designs proposed for DCNs on this basis [21–23] and majority of them are derived from graph-theoretical properties and Multistage Interconnection Networks (MINs) [24–27].

1.2 Routing protocols for DCN

The topology designs for data centers are defined recursively through baseline topology and growth model. The baseline topology (often termed $level_0$ of the

design) defines the design of a basic building block. Whereas, the growth model defines design of higher levels from the lower levels. A majority of the routing protocols for DCN designed their routing protocols by appropriately designing the baseline topology and growth model for their topology that enabled them to define the routing based on the underlying properties of topology. In this section, we present such efforts in designing routing protocols for DCNs.

1.2.1 Source routing

The designs which use source routing for data transfer are also called modular server centric (MSC) designs. They are called so because in these designs, responsibility of finding the routes exist with the source rather than intermediate switches. These designs are derived from the graph theoretical approaches which help the source to directly compute a path to the destination. Few example designs under this category are DCell [28], BCube [29], MCube [30] and MDCube [31]. Here, we briefly explain the working of source routing for DCell and BCube. We defer explaining other MSC designs as their routing methods follow similar approaches of DCell and BCube.

Designs DCell [28] and BCube [29] are example designs for weakly orthogonal and strictly orthogonal designs respectively. A network topology is orthogonal if and only if nodes can be arranged in an orthogonal n-dimensional space [24]. Routing in these designs is much easier because, separate indices are used for encoding each dimension and path to destination can be easily determined by finding an offset of a index in a particular dimension. Finding a path in strictly orthogonal design is easier because each point has at least one connection to each of the remaining dimensions. But, routing in weakly orthogonal design is involved because some nodes may not have connections to some dimensions. This is evident from the routing algorithms of DCell and BCube. The routing is by just finding hamming distance between successive levels in BCube, whereas in DCell, the routing is recursive and more involved. Further, routing in partial BCube and DCell is more involved because both the designs become weakly orthogonal in their partial

structure.

To understand the working of source routing, we briefly illustrate the working of BCube. BCube is a recursively defined structure. $BCube_k$ is formed from m $BCube_{k-1}$ and m^k *m-port* switches. The connections are established by connecting each switch among the set of m^k switches to a server from each $BCube_{k-1}$. Next, in order to correlate the address assignment with topology design, a vector of indices is used to address the servers and the intermediate switches wherein i^{th} index records the relative ordering of $BCube_{i-1}$ in $BCube_i$. Thus, a hierarchical assignment of addresses help the source to compute the path to the destination by correcting the hamming distance between the source and destination addresses. Hence, the source routing successfully addresses the scalability problem of earlier routing schemes for DCN as it does not involves any prior computations of routes.

Routing in DCell is similar to BCube. In DCell, the concept of complete graphs is used for defining source routing. In DCell, if $level_i$ subnet is abstracted for a vertex then $level_{i+1}$ subnet is formed from connecting $i+1$ $level_i$ subnets in the form of complete graph of $i+1$ vertices. Based on the logic used for forming the complete connected graph, the routing algorithm is derived for DCell.

1.2.2 Location based routing

In location based routing, the correlation between the topology design and address assignment is used to identify the relative location of the end hosts. The prominent designs using the location based routing are PortLand [20], Virtual layer 2 (VL2) [32] and Alias [33]. These designs make use underlying topologies characteristics to define their routing. The designs PortLand and VL2 use fat-tree as their underlying topology and Alias uses a general tree topology for its routing algorithm.

In PortLand, two address modes are used for addressing the end hosts and intermediate switches. An actual MAC(AMAC) address is used to identify the network elements based on their network address such as MAC address and a pseudo MAC address (PMAC) is used to address the network elements based on

their relative location in the topology. Since the underlying topology is fat-tree, the PMAC address is defined as *pod* : *position* : *port* : *vmid*. Here, *pod* (16 bits) represents the pod number of the edge switch, *position* (8 bits) represents position of the edge switch in the pod, *port* (8 bits) represents the port of the edge switch to which the end host is connected and finally, *vmid* (16 bits) represents the virtual machine id on a same physical machine. When an ARP is used to obtain mapping for IP address to MAC address, a dedicated switch called ingress switch in PortLand intercepts ARP packet and returns PMAC address of the host. Since PMAC gives the location information of the end hosts, routing is done based on this address. Finally, while delivering the packet to the end host, the PMAC address is replaced with the AMAC address at the egress switch and the packet is delivered to the end host. Here too, the location based routing successfully addresses the scalability problem of the routing protocols since it does not involves any prior computation of the routes.

Alias is an extension of PortLand wherein the underlying topology is a general tree. This results in additional complexities because, for a general tree, it is hard to determine the number of levels and structure of the tree. Therefore, Alias, first determines the levels at which each of the intermediate switches and end hosts present in a general tree and assigns unique coordinates to switches at each level. Later, unique labels are assigned to the end hosts based on the coordinates assigned to the end hosts and intermediate switches. Finally, the location based routing is done based on the labels assigned to the end hosts.

Similar is the approach in VL2. But, the main aim of the VL2 was to provide agility, an ability to allocate application to any server and still keeping the operations intact. For this, VL2 defines addresses - an application-specific address (AA) to identify the application and location-specific addresses (LA) to identify the location of the server on which the application is hosted. To map between the addresses, VL2 maintains a directory system that maps AAs to LAs and the routing is done based on LA. When the VM hosting an application has to move to other server (to avoid congestion or to perform load balancing), the AA address of application is mapped to LA address of the new server and the packets destined

for AA are then moved to a new server. This enables assigning an application to any server in the DCN and provides an impression that all the servers on which the application is hosted are on a single noninterfering Ethernet switch, named a *virtual layer 2*.

1.2.3 Hybrid methods

There are another class of routing proposed for DCN wherein advantages of different technologies are hand picked and combined to define hybrid architectures. For example, the hybrid electrical/optical structures recommends using both the packet switched Internet and circuit switched optical communication to efficiently handle the inter pod traffic. Here pods are defined as self contained shipping containers that consists of all the necessary components such as servers, networks and a cooling system. The designs, hybrid electrical/optical switch (Helios) architecture [34] and c-Through [35] are the example designs for hybrid architectures.

The Helios is a two level multi-rooted tree with the pod switches and the core switches. Unlike traditional systems, the core switches consists of both the traditional electrical packet switched networks and a Micro electro mechanical systems (MEMS) based optical circuit switch interconnections to handle multiple traffic patterns. In Helios, optical communication handles the many slow changing inter pod traffic and packet switched network handles bursty inter pod traffic.

Similarly, Guohui Wang, et al., defines a hybrid packet and circuit (HyPaC) network that reinforce traditional hierarchical packet switched networks for DCNs with the simple, low cost and a higher bandwidth optical network. They demonstrated use of this hybrid channels by designing a prototype called c-Through. To multiplex the traffic among the circuit switch and packet switch buffers, c-Through architecture increases the per-connection socket buffer limit and observes the per-connection buffer occupancy at the run time. Finally, to de-multiplex flows, VLAN are defined to handle the packets destined for packet switched network or the circuit switched networks.

1.3 Enhancement based on traffic patterns

There are some enhancements proposed for the existing data center architectures to improve their performance. These enhancements are the result of some important revelation from the study of traffic in data centers [6, 7]. The initial basis for the design of data center networks is to provide large bisection bandwidth to support what supposedly a peak internal traffic that all the servers may collectively generate. But, the study of traffic pattern for different categories of data centers have shown that this possible peak traffic rarely happens and most of the time resource of the data centers are underutilized. The other important revelation from the study of traffic pattern in data centers is volatility of data center traffic. Studies have shown that the traffic pattern inside a data center is very divergent. There are about 50 different representative traffic needed to define the overall traffic based on clustering techniques using various network parameters. Further, the traffic does not stay in a particular traffic pattern for more than 100 seconds. These two important revelations, the data centers are over-provisioned and volatility of data center traffic, motivated to propose mechanisms to improve the efficiency of the data centers.

1.3.1 Traffic patterns

Study of data traffic shown that the data centers are overprovisioned and their resource usage varies over time [6, 7, 32]. For example, a data center hosting a social networking may receive greater traffic during weekends compared to the traffic it receives on other days. However, the data center must be designed to handle peak traffic. Due to this volatility of the traffic matrix, applying the traffic engineering (TE) for DCN is a challenging job. Traffic engineering is a method of optimizing the performance of the network by dynamically analyzing, predicting and regulating the characteristics of the data transmitted over the network traffic [36, 37]. Traffic engineering applied for the Internet may not be directly extended to the DCNs because of diversity in their traffic matrix. Moreover, volatility of

traffic pattern in DCN make it hard to estimate and control the network parameters to get the desired performance.

There are some simple solutions defined to cope up with the volatility in traffic pattern, workload and failure patterns. In VL2, the valiant load balancing (VLB), a simple mechanism of distributing the traffic at each switch achieves the load balancing of the entire system. This is done without any centralized server performing any coordination and/or traffic engineering. Here, the switch need not to know how traffic was evolved over the time or anything about the future traffic pattern. Rather their experiments have shown that by knowing only the present incoming traffic at the switch, their design achieved uniform high capacity and performance isolation. This example hints at using features of DCN to solve some critical problems with DCNs .

1.3.2 Energy conservation

Study of traffic for DCNs shown that the data centers are over provisioned and the resources of DCN are often under utilized. Further, the amount of resources of data centers used is a time variant. However, data centers must be designed to handle a peak traffic and all the resources must be kept *on* all the time as it was very hard to predict when the requirement for peak traffic arises. Thus, there is scope to develop an energy efficient hardware and software along with developing an energy efficient routing algorithm. In this regard, ElasticTree proposed an energy aware routing that turns the resource of DCN *on/off* according to traffic condition [8, 38]. For this, an elastic tree intelligently routes the traffic only through the selected number of switches and switch *off* the rest of the network to conserve energy. Heller, et al., shown that their proposed elastic tree can save about 50% of the energy consumed by their underlying topology. There are many proposals that considered the properties of the underlying architectures to define an energy efficient routing algorithm. For example, Dennis, et al., proposed energy proportional DCN [38] and shown that the flattened butterfly topology consumes

less power than folded-Clos networks. Similarly, in [39], DCN design BCube is proved to be efficient in conserving energy compare to fat-tree.

1.3.3 Load balancing

Majority of the applications deployed on data center are distributed and they adopt a partition/aggregate computing model to process their assigned task in a timely manner. Here, the overall computation is divided among the nodes called worker nodes and results from the worker nodes are combined by the aggregator to produce the final result. Although, partition/aggregate scales well for solving problems with a large volume of data, they often associated with the problem such as incast congestion, queue build up and memory pressure. The incast congestion results when all the worker nodes simultaneously respond to the aggregator resulting congestion at the switch connecting worker nodes with the aggregator. The other problems queue build up and memory pressure are the result of dominance of large flows over the short flows. One way to overcome these problems is to priorities and schedule flows according to their merits.

The solutions proposed to overcome these problems are broadly classified as network layer solutions and transport layer solutions. The network layer solution may employ quality of service (QoS) to differentiate different flows [40, 41].

Similarly, transport layer solutions proposed modulating the TCP congestion window based on the extent of congestion and the associated deadlines for the flow. Earlier transmission control protocols (TCP) variants proposed for Internet such as TCP Tahoe, Reno, New-Reno, SACK and Vegas controlled the congestion in the network by adjusting the TCP congestion window of flows [19]. But, the problem with these approaches is, when the congestion occurs all the flows back *off* the window size equally. Thus, there are many TCP variants proposed for DCN which modulate the window size based on both the extent of congestion and the associated deadlines for flows. Few such proposals are, Data Center TCP

(DCTCP) [42], Deadline Driven Delivery control protocol (D^3) [43], Deadline-aware Data Center TCP (D^2TCP) [44] and Preemptive Distributed Quick flow scheduling (PDQ) [45].

1.4 Motivation

Earlier solutions proposed for the scalability problem of routing protocols for DCN introduced additional complexities to quick fix the scalability problem. The proposed solutions required additional control plane and management plane to solve the scalability problem. The key question that this dissertation aims to answer is - "can we provide the same solution with out introducing any additional control and management planes".

Earlier proposals for the scalability problem of routing protocols used two address spaces, one to identify the end hosts (AMAC, IP address) and another for routing (PMAC, labels). Both the designs define a separate control plane to identify the location of end hosts in their topologies and then assign addresses representing location information to the end hosts. Additionally, they require management planes to monitor the liveness of the end hosts based location information. Their approaches are similar to the approach of Locator/ID separation protocol (LISP) [46, 47] proposed to solve the scalability problem of routing protocols for Internet.

LISP separates *where* and *who* in networking and use a mapping system to couple location (*where*) and identifier (*who*). These addresses are respectively known as endpoint identifiers (EIDs) and the routing locators (RLOCs). The EID for the end host is the same physical address the service provider assigns. Whereas, the RLOC is the new address assigned by the Internet based on topology hierarchy and represents the relative location of the end host in the overall topology. During routing, EID is mapped to RLOC and routing is done based on RLOC. LISP was forced to do this separation because re-assigning IP addresses for an existing Internet is tenacious. But, as data centers are private, this separation can be avoided.

The above observation is the main motivation for our proposed novel architecture 4-4, 1-4 which evades use of any additional control or management planes and our proposed new architecture 4-4, 1-4.

The energy saving solutions for network topology are broadly classified as *traffic oriented* and *topology oriented* solutions based on the input parameters they use for energy saving. In traffic oriented solutions, the energy saving issue is typically formulated as a traffic engineering problem. Since this problem is NP-complete [8], the topology oriented solutions based on knowledge of the network topology are considered to be the best options [8, 38, 39]. Since the *topology oriented* approach is a localized optimization algorithm, the necessary support from the underlying topology and the corresponding routing protocol used are very significant. In our second contribution, we study our proposed 4-4, 1-4 architecture an ability to conserve energy based on its topology characteristics and the routing algorithm.

The two approaches prevalent in solving the problems due to congestion and meeting deadlines for flows are congestion control at the transport layer and resource allocator at the network layer. The solutions proposed for the transport layer modulated the congestion window for flows based on the extent of congestion and the deadline associated with flows. Since these methods modulate the congestion window based on the extent of congestion the flow experience on a particular path, they expect the flow to follow a same path. Thus, in order to change the behavior of network from connectionless to connection oriented, an intermediate state called *soft state* is defined which maintains information about all the flows going through a switch. Since data center traffic is very huge and volatile, maintaining information about all the flows at the intermediate switches effects the performance of the network. Our third contribution is to attain the same objectives through resource allocation at the network layer and preserve the network as a connectionless datagram delivery service.

The majority of the designs proposed for source routing are hierarchical wherein both their topology design and address assignment follows the hierarchical design principles. They all aggregated their lower level subnets to design a next level and assigned hierarchical addresses that certainly an index relative ordering of

$level_i$ subnet at the $level_{i+1}$ subnet. This essentially analogous to the supernetting feature of IP address hierarchy. Thus, in our final contribution, we availed similarities between the mechanism for assigning special addresses to the end hosts and IP address hierarchy to propose modifications to the source routing designs. The proposed modifications helped to define the earlier MSC designs using a single address space for both identifying end hosts and routing and evaded all the complexities of using two address spaces by the earlier designs.

1.5 Contributions

In this work, we propose scalable routing protocols for DCNs based on IP address hierarchy. The proposed routing algorithms are simple and efficient compared to their earlier routing schemes. This work consists of four parts: Propose a new location based routing which is simple and efficient compared to the earlier location based routing. As a part of this work, we propose a new architecture 4-4, 1-4. Apart from scalable routing, the other design consideration for DCN design is the optimization of resource usage. In our second work, we prove energy efficiency of 4-4, 1-4 architecture. The earlier proposals for meeting the deadlines for the flow, proposed solutions at the transport layer which required the network layer to define a *soft state*. In our third work, we propose an equivalent solution at the network layer which accepts the network as connectionless datagram delivery service. Finally, using IP address hierarchy, we propose a modification to the existing designs proposed for source routing to make them use a single address space for both identifying end hosts and routing. This evades all the complexities that resulted from use of two address spaces by the earlier designs.

1.5.1 Location based routing for data center network using IP address hierarchy

Location based routing is an alternative routing protocol proposed to solve the scalability problem of conventional routing protocols used for data centers. Location based routing determines relative location of the servers in a given topology and uses the location information for routing. PortLand and Alias are the two prominent designs for location based routing. Both PortLand and Alias bestowed on a significant exchange of messages between end hosts and intermediate switches to determine relative location of the servers in a given topology. Following are the limitations with the earlier proposed location based routing algorithms for DCN.

Large control overhead: Use of a separate network control plane and management plane for topology discovery and maintenance introduces a large control overhead. The location discovery protocol (LDP) messages exchanged between the servers and intermediate switches to determine their level and relative position in the level result in significant internal traffic. Data centers typically host millions of servers. Thus, huge control information generated by these large number of servers and intermediate switches may overlap with the original traffic and may accounts for the performance degradation of the system.

Half path: Location information, both in PortLand and Alias, helps routing in a single direction, a 'downward path' from core routers to edge servers. Both the designs use mechanism other than location based to carry packets from edge servers to core routers. PortLand uses diffusion and Alias uses virtual circuit interconnection (VCI) for data transfer in 'forward path'.

Re-Programming: Since, in these designs, routing is done using newly introduced addresses such as labels and PMAC addresses, re-programming intermediate switches is required to change lookup operation from IP address to newly introduced addresses. Further, since the packets are to be augmented with labels/PMAC addresses, the Ingress switches are to be re-programmed to respond with labels/PMAC addresses when they receive ARP requests and the Egress switches must remove the newly introduced address to ensure proper working of layers

above and below the network layer.

In this work, we use correlation between topology design and address assignment to derive the location of end hosts from their addresses. Using this location information, we devise a location based routing algorithm that determines the path to the destination using the location information derived in the previous step. To demonstrate our proposed location based routing, we designed a new architecture called 4-4, 1-4 that correlates topology design with the address assignment. Lastly, we study the performance benefits of our proposed location based routing such as reduced routing delay, better bisection bandwidth and better route aggregation by conducting simulations using NS-3 simulator.

1.5.2 Greening 4-4,1-4: A distributed greedy approach for finding an energy efficient sub-network

Data centers are over-provisioned with high network bandwidth through a large number of interconnections to handle traffic surges. However, various studies for data center traffic regarded this upsurge as a rare event and shown that the resources of data center are underutilized. There are many components involved in data centers - servers, switches, routers and cooling systems - whose usage can be optimized according to the traffic condition. Since network contributes 20% to 30% of energy usage of DCN, in this work we consider the problem of optimum usage of network in 4-4, 1-4 [48, 49]. Since finding an optimal sub-network to accommodate current set of flows is NP-hard [8, 39], we propose a distributed greedy algorithm to find an energy efficient sub-network to accommodate the current set of flows. Our proposed greedy method consists of four steps, (1) *Finding a sub-network:* Here, we determine the sub-network sufficient enough to cover all the flows while maintaining the predefined threshold. (2) *Switching off the un-used switches and links:* In this phase the switches and links of original network which are not part of the sub-network are switched *off* to save energy. (3)*Routing flows:* Here, all the flows are made to select the paths only from the sub-network computed in step (1). For some DCN designs, this may results in selecting a non-optimal path, (4)

Updating the flow matrix : When a new flow arrives or an existing flow finishes the flow matrix is updated accordingly.

To study the performance of our proposed method, we simulate our proposed algorithm in network-simulator (NS3) and compare the results with ElasticTree.

1.5.3 Packet scheduler for meeting deadlines in 4-4, 1-4 architecture

The distributed computing model such as MapReduce, Dryad, cluster based file systems, query to web search engine, parallel scientific applications and many other are associated with problems such as incast congestion, queue buildup and memory pressure. These adversities hinders flows to meet their associated deadlines and ultimately the SLA. There are two classes of solutions proposed for the problems of congestion and meeting deadlines for flows - a congestion control at the transport layer and resource allocation at the network layer. The proposed solutions at the transport layer modulated the TCP window size according to the extent of congestion and the deadline value associated with flows [42–45]. These solutions are proposed based on the assumption that the intermediate switches maintain *soft-state* about the flows. Thus, the entire flow has to follow a static path from beginning to end.

Contrast to this, the location based routing proposed for DCN tries to utilize the path diversity provided by the underlying architectures by distributing flows among all the available outgoing paths using the Equal Cost Multiple Paths (ECMP). Since our proposed 4-4, 1-4 architecture provides multiple paths between the source and destinations, in our third work, we propose a packet scheduler for 4-4, 1-4 that helps flows to meet their deadline by arranging the packets at intermediate switches according to their associated deadlines. With prioritizing and scheduling the packets at the network layer, the flows are not constrained to assign to a static path and the intermediate switches need not to maintain any flow state information.

1.5.4 Modular data centers simplified using IP address hierarchy

Designs proposed for source routing such as DCell [28], BCube [29], MCube [30] and MDCube [31] use the structure-properties of graphs [24, 26, 27] to determine path to the destination. One of the major limitation with these deigns is use of two different address spaces, one to identify end hosts and another for routing. The use of two address spaces require mapping between the address spaces and modifying packets to replace special addresses with IP addresses. In our last contribution, we propose a simple mechanism that overcomes these limitations by redesigning the MSC designs to use a single address space for both identifying end hosts and routing. The main idea of using a single address space is to encode the special addresses in the IP addresses that evade the mapping during data transfer. Apart from encoding, we propose a small change to the original topology design that helps to change the designs into a true networking topologies.

1.6 Organization of the thesis

The rest of the thesis is organized as follows. Following this introduction chapter, we provide background literature survey about evolution of data center centers along with the background for various alternative routing protocols proposed for DCN. Further in the chapter, we briefly describe other research challenges in the area of DCN. The Chapter **3** to Chapter **6** covers individual contributions of our work. In Chapter **3**, we present our proposed new architecture 4-4, 1-4 and location based routing that overcome the complexities of the earlier location based routing algorithms. In Chapter **4**, we show the amenability of 4-4, 1-4 for energy efficiency. Next, in Chapter **5**, we propose a novel network scheduler for meeting deadlines for flows in DCN. The Chapter **6** is our last contribution where we simplified the source routing to obtain a better routing performance. Finally, we conclude this thesis with Chapter **7**.

Chapter 2

Literature Survey

Research in the field of Data Center Networking(DCN) taken a greater momentum with data centers becoming increasingly essential part in the present Internet technology. The increasing complexity and sophistication of the applications deployed on DCN demanded new features and greater performance from the DCN. This resulted in many designs proposed for DCN addressing the growing needs of DCN such as cost, performance, reliability, scalability, security and energy. Here, we present a brief survey of data center networks designs that are proposed recently. First, we present various representative topologies proposed for DCN, Next, we discuss various routing protocols designed for these topologies and compare their merits and demerits. Finally, we present some advances proposed for DCNs to improve their performance. An insight into state-of-art of the design principles in these proposals provide a necessary background for our contributions.

2.1 Data center network architectures

Initial designs proposed for DCN used specialized hardware and communication protocols such as InfiniBand [50] or Myrinet [51] that offer very high throughput and very low latency. The limitations with InfiniBand or Myrinet is the use of specialized hardware and protocols that are not compatible with TCP/IP.

The proposed solution is to use the commodity switches and provide greater bandwidth through a large number of interconnections. The conventional network architectures for DCN are build from commodity switch application specific integrated circuits (ASICs) arranged in a Clos topology. But, these designs to reduce overall cost, used less number of high end switches at the higher level compared to a large number of low cost switches used at the middle and lower levels. This imbalance between the number of switches used at different levels resulted in oversubscription of switches at the core level and partitioning of underlying topology in the event of switch failures at the core level. Thus, to overcome the use of high end switches at the core level, many designs such as fat-tree, DCell, BCube, PortLand, Alias and VL2 are proposed that use only the commodity switches at all the levels.

2.1.1 Fat-tree

The primary aim of the fat-tree is to provide 1:1 oversubscription using only the commodity switches [13]. An *s-ary* fat-tree is a three layer design: an edge level, an aggregate level and a core level. An *s-ary* fat-tree uses *s-port* switches for its design. An *s-ary* fat-tree is made up of s pods. Each pod contains two layers of $\frac{s}{2}$ switches. Each *s-port* switch of the lower layer directly connects to $\frac{s}{2}$ hosts. The remaining $\frac{s}{2}$ ports of the switch are connected to all the $\frac{s}{2}$ switches at the upper layer. This makes a complete bipartite connection between a lower layer and an upper layer. These two layers in combine form aggregation level of fat-tree. Finally, at the core level, there are $(\frac{s}{2})^2$ *s-port* switches. To explain the interconnection between core level and aggregate level, the core level switches are indexed as $[i, j]$ where $i, j \in [0, s-1]$. The l^{th} ($l \in [0, s-1]$ port of $[i, j]^{th}$ switch at a core level is connected to i^{th} aggregate switch of l^{th} pod.

Address assignment for fat-tree is done using a private address block 10.0.0.0/8. Each pod switch address is of the form 10.*pod*.*switch*.1, where *pod* and *switch* in $[0, s-1]$, are their position in topology. Next, hosts are assigned the addresses 10.*pod*.*switch*.ID, where ID is the position of the host connected to the edge

switch. Since $\frac{s}{2}$ ports of the edge switch are connected to $\frac{s}{2}$ hosts, the range of values ID can take is $[2, (\frac{s}{2}) + 1]$ and their given address block is $10.i.j.0/24$. Finally, core switches are given an address block $10.k.i.j0$, where j and i denotes the switch coordinates in $(\frac{k}{2})^2$ core switches grid. Fig.2.1 shows the addressing scheme for fat-tree with $s = 4$.

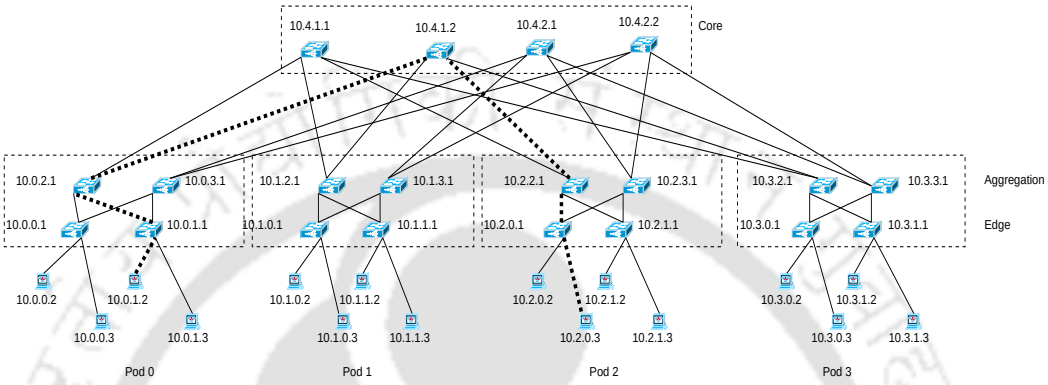


FIGURE 2.1: Fat-tree architecture for Data Center

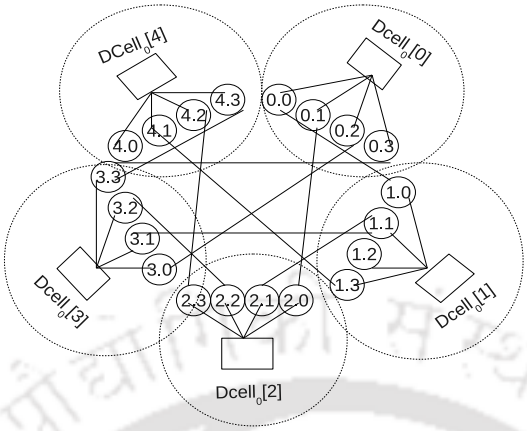
2.1.2 DCell

DCell is a recursively defined structure [28]. Baseline topology of DCell, $DCell_0$, consists of n servers connected to a mini-switch. $DCell_1$ is defined using $n + 1$ $DCell_0$s. In $DCell_1$, each $DCell_0$ is connected to all other $DCell_0$s with one link for each $DCell_0$. Thus, if we abstract a $DCell_0$ for a virtual node, then $DCell_1$ is a complete connected graph of $n + 1$ virtual nodes. The $DCell_1$ from $n + 1$ $DCell_0$s is constructed as follows. Each server in $DCell_1$ is assigned an address $[a_1,a_0]$, where $a_1 \in [0, 5]$ and $a_0 \in [0, 4]$ are the IDs for $level_1$ and $level_0$ subnets respectively. Two servers with addresses $[i, j - 1]$ and $[j, i]$ are connected with a link for every i and every $j > i$. $DCell_1$ for $n = 4$ is shown in Fig.2.2(a).

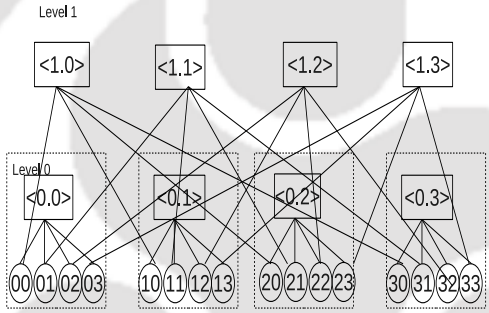
Similarly. a higher level $DCell_k$ is constructed using $t_{k-1}+1$ $DCell_{k-1}s$, where t_{k-1} is the total number of servers in each $DCell_{k-1}$ [28]. Thus, the growth model of DCell is defined in terms of g_k, number of $DCell_{k-1}s$ in $DCell_k$ and t_k, number of servers in $DCell_k$, where, $g_k{=}t_{k-1} + 1$, $t_k{=}g_k \times t_{k-1}$ with $t_0 = n$.

Each server in $DCell_k$ is addressed as $\langle a_k a_{k-1}...a_0 \rangle$, where $a_i, i \in [1, k]$ and

$a_i < g_i$ is the index of $DCell_{i-1}$ in $DCell_i$. While $a_0 < n$ is the position of the server in $DCell_0$.



(a) $DCell_1$ of DCell



(b) $BCube_1$ of BCube

FIGURE 2.2: Example designs for Modular Server Centric (MSC) Design.

2.1.3 BCube

BCube is another recursively defined structure which is simple compared to DCell due to its regular growth model [29]. Basic block $BCube_0$ consists of m servers connected to an *m-port* switch. $BCube_1$ is constructed from m *m-port* switches and m $BCube_0$s. Construction of $BCube_1$ from $BCube_0$ for $m = 4$ is shown in Fig.2.2(b). In general, $BCube_k$ is constructed from m $BCube_{k-1}$s and m^k *m-port* switches. Each server in $BCube_k$ has $k+1$ ports, which are numbered from $level_0$ to $level_k$. Thus, $BCube_k$ has $M = m^{k+1}$ servers and $k+1$ levels of switches, with each level having m^k *m-port* switches. Construction of $BCube_k$ from m $BCube_{k-1}$

is done as follows. The m^k switches are divided into k set of switches, where each set consists of m *m-port* switches. The l^{th} port of $(i,j)^{th}$ switch, where both $j,l \in [0,m]$ and $i \in [0,k-1]$ connects to j^{th} server from each l^{th} $BCube_{k-1}$.

Similar to DCell, BCube addresses its network elements as $\langle a_k, a_{k-1}, a_{k-2}, ..., a_1, a_0 \rangle$, where each a_i is the index of i^{th} level subnet in $(i+1)^{th}$ level subnet.

2.1.4 Other architectures

There are many derivatives of the above proposed architectures are proposed for DCN. The design MCube [30] is a new type of physical network infrastructure defined for small and middle size data centers. MCube uses servers with two ports instead of servers with $K+1$ ports for its k^{th} level design. The topology design of MCube places the switches at the end points of hypercubes and servers are placed between two such end points. The baseline topology of MCube is a cube containing 8 mini switches and 12 servers. The higher levels in MCube are formed by interconnecting the lower level MCube in the form of hypercubes. In general, an $MCube_r$ with $r \times r \times r$ basic blocks can support upto $3r(r+1)^2$ servers, where r is the level of MCube. The main advantage of MCube is the presence of multiple paths for each server. This forms multiple spanning tree's and helps to speed up the data transfer.

In another variant, MDCube [31] defines shipping container based mega data centers [52–54] by modifying the existing BCube to use high-speed uplink interfaces of the commodity switches. Present COTS switches provide many Gigabit ports (about 48) and several high-speed 10 Gigabit ports (about 4). The earlier BCube used only the Gigabit links for interconnection. The idea behind MDCube is to use Gigabit links for intra-containers connections and to use high-speed links for inter-container connection. This helps to achieve higher performance-to-price ratio.

There are many variants proposed for DCell. FiConn proposes to build a data centers using only the servers with two ports and low cost commodity switches, instead of using multi-port servers by the earlier designs. Finally, Markus et al. [55] proposed a generalized DCell framework for constructing different DCell

variants based on different interconnections patterns. This enabled to test various properties of DCell under different interconnection patterns.

Next, the design PCube [56] proposed a method to conserve energy for cube based designs such as BCube and MDCube. Finally, the designs dBCube [57] and KCube [58] combine the features of hypercubes with de Bruijn digraph and Kautz digraph respectively to define hybrid graphs. All these designs variations are based on BCube and DCell.

2.2 Routing in data center networks

The routing protocols proposed for DCNs use the topology characteristics of the underlying topology to derive the path(s) from source to destination. In this sections, we discuss certain approaches that take advantage of the properties the underlying topology possess to derive their routing algorithm. Here, we classify the routing algorithms based on (1) Whether routing responsibility lies with the central server or distributed among many network components, (2) Whether the routes must be pre computed before data transfer or derived instantaneously, and (3) Whether forwarding is responsibility of intermediate switches or the servers.

2.2.1 Routing in fat-tree

Routing in fat-tree uses two-way lookup with prefixes are used for forwarding intra-pod traffic and suffixes are used for inter-pod traffic. The routing table for the switch with the IP address 10.2.2.1 is given in Fig.2.3. The primary level routing table is used for routing the packet in the downward direction, that is from core routers to the edge switches and the secondary level routing table is used for moving the traffic in the forward direction. The routing based on the suffix is basically used to diffuse the traffic. For example, from the node with IP address 10.2.2.1, all the pod switches can be reached through a single port connecting to a core router. Thus, if all the hosts use the same port for inter pod communications

then the corresponding link may become congested. Instead, if the traffic is distributed based on node ID's, then the traffic is distributed among all the uplinks of the switch. The path established to transfer a packet from source (10.0.1.2) to destination (10.2.0.3) is shown with a dash line in Fig.2.1.

Thus, using *s-port* switches, fat-tree supports $\frac{s^3}{4}$ servers and the maximum path length in fat-tree is six.

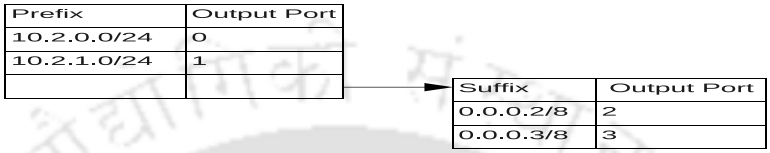


FIGURE 2.3: A table showing the two way lookup for the routing algorithm of fat-tree

One of the limitation of fat-tree is finding the routes and generating the routing table entries for the core and aggregate switches. For this, the global knowledge of a topology is required by the central router. Thus, the limitations of OSPF such as supporting larger topology are also applicable for the fat-tree [20]. Thus, to overcome the scalability problem of conventional protocols, the following alternative routing protocols are proposed for DCN.

2.2.2 Source Routing

In source routing, topology constructed based on graph theoretic properties enable a source to directly compute the path to a destination. There are many designs proposed based on this property and the list includes - DCell [28, 55], FiConn [59], BCube [29], MCube [30] and MDCube [31]. All these designs are derived from structure-property relations of graphs [24] [26] [27]. For instance, topology design of MCube, BCube and MDCube are derived from hypercube. These designs use vectors to record indices for each dimension and as in hypercube, the difference bits (hamming distance) are corrected to find the path from source to destination. In another variant, designs DCell and FiConn can be abstracted to complete graph of n vertices. In DCell, if $level_i$ subnet is abstracted for a vertex

then $level_{i+1}$ subnet is formed from connecting $i+1$ $level_i$ subnets in the form of complete graph of $i+1$ vertices.

To summarize, in these designs, graph theoretical properties enable a source to directly compute the path to the destination. The proposed routing algorithms scale for a very large number of server as they do not involve any prior computations of routes. These designs are also termed as modular server centric (MSC) designs since they resemble orthogonal design in *n-dimension* space and routing is the responsibility of the servers rather than intermediate switches. In this subsection, we briefly describe two such designs - DCell and BCube.

2.2.2.1 Routing in DCell

DCell proposed an efficient single-path routing algorithm by using the recursive nature of the topology design. If two nodes *src* and *dst* are in same $DCell_k$ but in different $DCell_{k-1}$, then intermediate link (n_1, n_2) that connects two $DCell_{k-1}$ is determined. Thus the path is from *src* to n_1 and n_2 to *dst*. Similarly paths from *src* to n_1 and n_2 to *dst* are calculated recursively. Thus, the final path is combination of paths *src* to n_1, n_1 to n_2 and n_2 to *dst*. The routing algorithm for DCell is given in Algorithm 2.1. In the algorithm, GetCommonPrefix returns the common prefix of *src* and *dst*. For example, to find links connecting two $DCell_i$s, the indices used represent $DCell_{i+1}$ to $DCell_k$ are masked and the indices for $DCell_i$, that is src_i and dst_i are extracted. Then, by using the DCell building procedure, the src_i and dst_i are connected through $\langle src_i, dst_i - 1\rangle$ and $\langle dst_i, src_i\rangle$ are determined. This procedure continues recursively until links connecting all the levels of $DCell_i$s, where $i \in [0, k-1]$ are resolved.

DCellRouting (*src*, *dst*)

Input :

$src = [s_k, s_{k-1}, ..., s_{k-m+1}, s_{k-m}, ...s_0]$

$dst = [d_k, d_{k-1}, ..., d_{k-m+1}, d_{k-m}, ...d_0]$

pref = GetCommonPrefix(*src*,*dst*);
m = len(pref);
if m == k **then**
return (*src*, *dst*);
end if
(n1,n2) = GetLink(pref,s_{k-m}, d_{k-m})
path1 = DCellRouting(*src*,n1);
path2 = DCellRouting(n2,*dst*);
return path1 + (n1,n2) + path2

Algorithm 2.1: **Algorithm for finding a path in DCell**

2.2.2.2 Routing in BCube

The BCube structure is closely related to generalized Hypercube. BCube uses a vector to record indices for each dimension and as in hypercube, the difference bits (hamming distance) are corrected to find path from source to destination. Using this principle, for given two servers with address arrays $A(a_k a_{k-1}...a_0)$ and $B(b_k b_{k-1}...b_0)$, multiple paths are determined by finding $h(A, B)$. Where h is the hamming distance of two servers addresses A and B, which is the number of digits disagreed in their address array. The Algorithm 2.2 gives the routing for BCube.

BCubeRouting(A, B, Π)

Input :

$A = [a_k, a_{k-1}, ..., a_0]$

$B = [b_k, b_{k-1}, ..., b_0]$

$\Pi = [\Pi_k \Pi_{k-1}, ...\Pi_0]$ is a permutation of $[k, k-1, ...0]$

path(A,B) = {A};
I_Node = A;
for $i = k$ to 0 **do**
  if $A[\Pi_i] \neq B[\Pi_i]$ **then**
    I_Node$[\Pi_i] = B[\Pi_i]$;
    append I_Node to path(A,B);
  end if
end for
return path(A,B);

Algorithm 2.2: **Algorithm for finding a path in BCube.**

In BCube routing, $A = [a_k a_{k-1}...a_0]$ and $B = [b_k b_{k-1}...b_0]$ are the addresses of the source node and destination node respectively. A simple algorithm for finding a path in BCube determines Π, a permutations of k digits $[k, k-1, ..., 0]$ in the source and destination addresses. Then, the path is determined by systematically correcting the digits that differ in source address and destination address. For example path from A(0001) to B(1011) is determined as $\{0001, \langle 3, 001\rangle, 1001, \langle 1, 101\rangle, 1011\}$. It can be easily shown that if all the digits of A and B are different, then there are maximum $k + 1$ paths from A to B.

2.2.3 Location based routing

The other category of the routing algorithm proposed to solve the scalability problem is the location based routing. In this subsection we briefly discuss the location

based routing algorithms designed for DCN. At the end, we discuss the similar approach adopted by the Internet to solve the scalability problem.

The location based routing algorithms for DCNs are derived from Multistage Interconnection Networks (MINs). In MIN, input devices are connected to output devices through a number of stages. Depending on the interconnection scheme used between adjacent stages and number of stages, various MINs, Banayn, Omega, Butterfly, *Beneš* and many other, are proposed in the literature [60]. The different interconnection patterns are done based on different permutations, a perfect shuffle, that determines a path from source to destination. For example, design fat-tree [13] is derived from bidirectional MIN with turnaround routing [61]. In conventional MINs, the outputs from each stage are connected to the inputs of the next stage using a perfect shuffle connection system. Location based routing are expander MINs with n inputs and m outputs, where $n < m$. Two prominent location based routing protocols proposed for DCN are PortLand [20] and Alias [33]. Unlike in conventional MINs, where the connection between successive stages is determined through the permutations, location based routing for DCN uses the known baseline topology and the growth model of the underlying topology. For finding a relative location of the end hosts, PortLand uses the topological characteristics of fat-tree and the Alias uses the properties of general tree. A MIN representation of PortLand is shown in Fig.2.4.

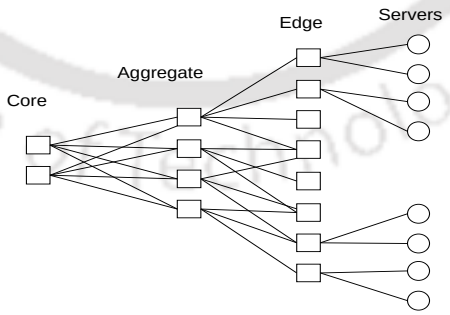


FIGURE 2.4: A multi stage interconnection (MIN) representation of PortLand

2.2.3.1 PortLand

PortLand uses the DCN topology similar to multi-rooted tree, fat-tree [13]. PortLand defines a new address called Pseudo MAC address (PMAC) to encode the relative location of the each host. For this, it uses the symmetry in interconnections of fat-tree. PortLand proposes a scalable layer 2 routing, forwarding and addressing for DCN. For this, The task of finding a route in PortLand is broadly divided into two steps, first, determining the relative location of the network elements in a given topology and the second, to find the route based on location information determined in the first step. For this PortLand uses two addressing modes, an actual MAC address (AMAC), the real MAC address for the network elements and a Pseudo MAC address (PMAC), the new address defined for embedding the location information.

Determining Location information: In this step, PortLand determines the positions of switches and end hosts in the topology using light weight protocols such as Location Discovery Protocol (LDP). The network elements in order to set their location, periodically send Location Discovery Messages (LDMs) to their neighboring switches to establish their position.

First, in order to set their levels, network elements are configured such that only the switches are to respond to the LDM messages and the hosts are refrain from doing so. By this, the edge switches receiving the messages from aggregate switches first confirms their level. This information is further propagated to the higher layers, that help them to determine their levels in the topology. After determining the levels of the network elements, each switch determines their position in their level. For example, the edge switches and aggregation switches determine about their pod number and their relative position within the pod and the core switches determine their ordering among $(\frac{s}{2})^2$ set of core switches.

Once the relative position of the switches is determined, the edge switches assigns unique identifier (PMAC) to its directly connected hosts by aggregating the coordinates from the core switches. Precisely, the edge switch assigns a 48-bit PMAC of the form $pod : position : port : vmid$ to the end host, where pod (16 bits) reflects a pod number of the edge switch to which the host is connected,

position (8 bits) is its position in the pod, and *port* (8 bits) is the switch port number through which the host is connected to. Last 16 bits for *vmid* multiplex multiple virtual machines on the same physical machine.

After switches establish their relative position in the topology, they use updates

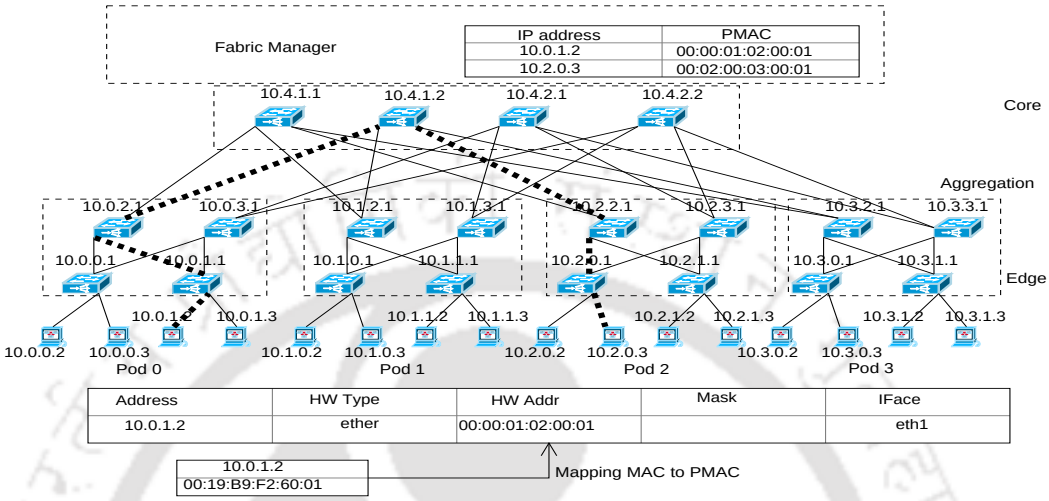


FIGURE 2.5: Location identification and routing in PortLand using PMAC Address.

from the neighboring switches to populate their forwarding table. For example, the core switches learn the pod number of the directly connected aggregation switches, the aggregation switches learn the position number of all directly connected edge switches and the edge switches learn port through which they are connected to the end hosts. Thus, the forwarding tables in these switches maps the coordinates of their neighboring connected switches to the link address of the port through which the corresponding neighboring switch is connected.

Once the forwarding tables are established for each switch, the forwarding of packets is done based based on the PMAC address. For this. PortLand maintains a mapping between the IP address to PMAC address in the special designated switches called ingress switches. To resolve, IP address to PMAC address, the intermediate switches are configured to send the ARP request to the ingress servers which in turn returns the PMAC address of the end host to the switch. Then, all packet forwarding proceeds based on the PMAC address and finally, egress switches perform PMAC to Actual MAC (AMAC) header rewriting to maintain the illusion of the unmodified MAC address at the destination host. In PortLand,

constructed PMAC address only help core switches to determine the servers position in the topology. Thus, only downward movement of the packet, that is from core switches to edge switches use location information in the PMAC for data transfer. Upward movement of the packet, that is from the edge switch to core switch still uses diffusion method as proposed in fat-tree. The assignments of PMAC address to servers and the routing mechanism in PortLand is shown in Fig.2.5.

2.2.3.2 Alias

Location based routing in Alias [33] is more complex compared to the PortLand as the underlying topology used by the Alias is a general tree. Due to this, Alias involves additional complexities such as determining levels of each switch and relative position of the switches in each level. The location based routing in Alias is broadly divided into three steps.

Level Assignment: Since the number of levels in a general tree is not fixed, the first task in Alias is to define an L_i switch to be the switch with i hops to the nearest host. The process of determining the levels is initiated with the switches at level L_1. For this, Alias periodically exchanges Topology View Messages (TVMs) between switches at the neighboring levels. In addition to sending TVMs, each switch also periodically sends IP pings to its neighbors which are end hosts. The end hosts are configured to respond to the ping messages but not to the TVMs. Thus, switches at L_1 can easily determine their levels as they are the only switches which are connected to the end hosts. Later, the level information of L_1 switch is propagated to the higher levels which enables switches at other levels to determine their level.

Label assignment: During label assignment, the switches at the same level are assigned a non overlapping coordinates based on their relative ordering of connectivity to their parent node. Since there is no centralized routine to perform this task, they resolve their coordinate values using decider/chooser abstraction (DCA). In DCA, two choosers resolve their coordinate values by selecting a suitable coordinates for them and communicating their chosen value to the chooser.

The chooser responds with an acknowledgment if no other switch has used that coordinate. If in case the coordinate already in use, then decider compiles a list of hints of already selected values and sends this list with its rejection to the chooser. The chooser then selects an unique coordinate based on this hint.

Once the coordinates are assigned to every switch, the labels are assigned to the end hosts by concatenating the coordinates of the switches along a path from the core switches to the labeled switch. Alias defines the concept of hypernode to limit the range of coordinates assigned to the end hosts. A Hypernode is defined as set of L_i switches that are strongly connected to sets of L_{i-1} switches. The concept of Hypernode is to assign a unique coordinate to the set of nodes in that corresponding hypernode, instead of assigning separate coordinates to individual nodes in it. By the use of single coordinate among the members of L_i hypernodes, ALIAS enables hosts below the hypernode to share a common label prefix, thus reducing forwarding table entries. In Fig.2.6(a), grouping of nodes into hypernode is shown by encircling them. Since S5 and S6 switches at L_2 connect to same set of L_1 switches, namely {S1,S2,S3}, they are grouped together as L_2 hypernode. Next, since switch S4 at L_2 is connected to {S1,S2} at L_1, it forms independent L_2 hypernode. Similarly, as switches S7 and S8 at L_3 are connected to both the L_2 hypernodes (although through different constituent members), they form one L_3 hypernode and as S9 connects only to one L_2 hypernode below, it forms a second L_3 hypernode. An example of label assignment in Alias is given in Fig.2.6(b).

Routing in Alias: In Alias, to forward the packets based on labels, every switch maintains the forwarding table by mapping incoming port and coordinates to the appropriate output ports. A L_i switch compares destination label with any one of its matching label prefix to forward the packet to the L_{i-1} level switch. If none of their label prefixes matches with the destination label, then the packet is forwarded towards the core switch.

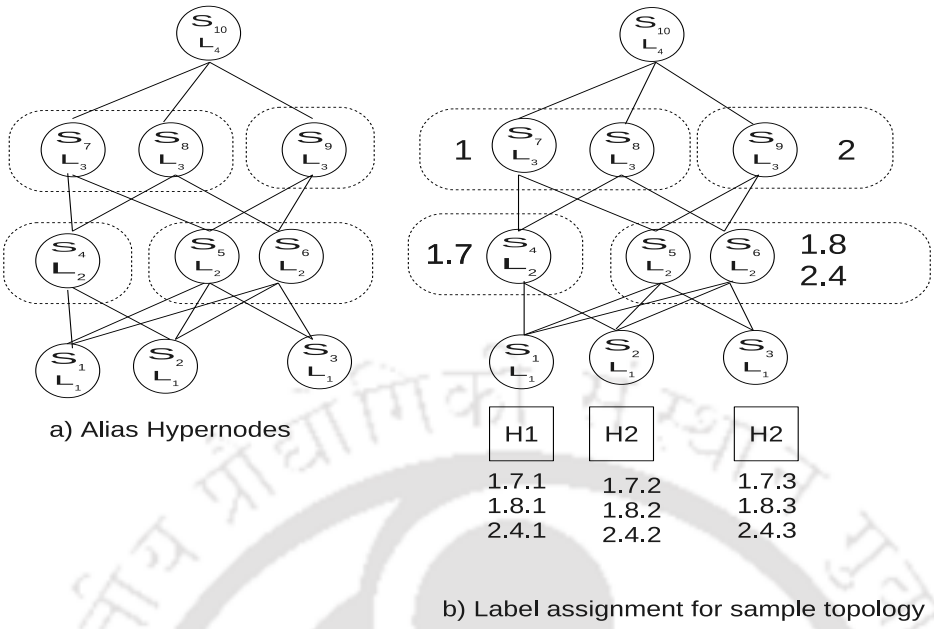


FIGURE 2.6: Concept of Hypernodes and label assignments in Alias

2.3 Enhancement based on study of traffic pattern in DCN

The initial design motive for DCNs is to provide high bandwidth through a large number of interconnections. But, study of data center traffic [6, 7] has revealed some important facts about network level traffic characteristics that helped the researchers to reassess design considerations. In this section, we briefly explain such a studies conducted for data centers and their implications for future researches in DCN.

2.3.1 Traffic characteristics:

There are two prominent studies for measuring and analyzing the network traffic for data centers. Benson et. al., presented their initial empirical study of end-to-end traffic patterns in data centers in [7]. Later, the same group reported the detail study of traffic pattern for different variety of data centers in [6]. The first study

analyzed traffic pattern collected for 19 corporate and enterprise data centers and the second study considered a proper mix of data centers such as three university based data centers, two private enterprise data centers and five cloud data centers. Both studied the traffic patterns at macroscopic and microscopic scales by collecting coarse-grained data from simple network management protocols (SNMP) and fine-grained data from the packet traces for the switches, respectively. Following are the some important findings from the studies.

- ***Applications in data centers:*** There were variety of applications hosted on data centers such as regular and secure HTTP transactions, file sharing applications such as server message block (SMB) protocol, authentication services such as light weight directory access protocol (LDAP) and file sharing protocols such as Apple filing protocol (AFS) and netware core protocol (NCP).

- ***Application communication patterns:*** In this analysis, they studied aggregate network transmission characteristics of the applications wherein they analyzed aggregate characteristics such as arrival rates, sizes and durations for the traffic at both the flow level and the packet level. Regarding number of flows at any given instance of time, it is varied across the data centers. However, in all the data centers number of flows are found to be less than 10,000 and majority of these flows (about 80%) are less than 10 KB in size. Regarding packet level communication, the packet arrival exhibited *on/off* pattern wherein durations of the *on* period, duration of the *off* period and packet inter-arrival time during the *on* period fitted lognormal distribution.

- ***Network communication patterns:*** In this analysis, they studied the utilization of the network resources of data centers by the deployed applications. Specifically, percentage of utilization of network resources, the variation in resource usage at different levels, time variant utilization of network resources and finally the effect of resource utilization on the packet loss rate. The key finding from this analysis are - (a) The ratio of intra-rack and

inter-track in data centers is depend upon the nature of data center. For example, in cloud data centers, significant amount of traffic stayed in side the rack while in enterprise and university data centers, the inter-rack traffic was predominant, (b) Among the switches at different levels, the core level switches are found to be heavily utilized and the edge switches are found to be lesser utilized, (c) To analyze the loss rate, they defined 'hot-spot' links, the links that are utilized more than 70%. Although, there were many hot-spots links at the core level, the number of core links that are hot-spots at any given time is less than 25%, and (d) Losses at the links are not proportional to persistently higher utilization, rather they are attributed for the momentary bursts in the traffic.

2.3.2 Implications for data center designs

The above findings instigate to reconsider data center network architectures attempt to maximize the network bisection bandwidth. A data center was said to exhibits full bisection capacity if it can support all the servers communicating with their full link speed capacity. Following are the two implications made out of the study of traffic pattern for DCNs.

(1) The first implication is due to the finding that percentage of bisection bandwidth provided by the data centers is sufficient enough to support any kind of traffic and percentage of bandwidth utilization across all the data centers has never exceeded 25% of the total available bandwidth. This means that the data centers equipped with a network that provides full bisection bandwidth, at least 95% of its capacity would go unused for current traffic patterns. This implies that, rather than increasing the capacity of data centers, what is needed is improving routing, topology and better load distribution for the data centers.

(2) Few of the architectures proposed [8, 13, 20, 62, 63] for DCN relay on centralized controller for finding a route and dissemination of routing information to the end hosts. However, efficiency of the solution depends upon the flow inter arrival time and scalability of the controller for a large volume of data. For instance,

for a data center with 100 end nodes and with an interarrival time for the flows is 10 μs, one can see 10 million flows per second. Thus, efficiency of the centralized controller greatly influences the overall working of the system.

2.4 Optimizations proposed for DCNs

In the remaining part of the section, we discuss three research dimensions proposed to optimize the performance of DCN designs. Two proposals, dynamic virtual machine migration and congestion control are proposed for load balancing and other proposal is to optimize the energy usage of DCN. These proposals are based on the study of data center traffic for DCNs. The proposals, dynamic virtual machine migration and congestion control are derived from the first implication. The other proposal, optimization of energy usage of DCN is derived from the second implication.

2.4.1 Dynamic virtual machine migration

Virtual Layer 2 (VL2), to achieve high capacity between two end servers [32], provides a mechanism to assign service to any set of servers that are by no means inter-related to each other according to the Networking properties. VL2 uses two separate classes of IP-addresses, Application Addresses (AAs) used for application servers and Location Addresses (LAs) used for location. The routing mechanism is unaltered, and uses the proven network routing protocol OSPF that operate based on the LAs assigned to the end hosts and intermediate switches. On the other hand, applications use permanent AAs, which remain same irrespective of change in server location due to VM migration or re-provisioning. VL2 uses association between AA and LA, the IP address of the ToR switch to which the application server is connected. The ToR switch need not be physical hardware, it could be a virtual switch or hypervisor implemented in software on the server itself. Thus, VL2 maintains a directory system that stores the mapping of AAs to LAs, and

this mapping is done when the application servers are provisioned to a service and are assigned AA addresses.

Since AA addresses are not associated with the routing protocols of the network, they must be mapped to LA. This mapping from AA to LA is done as follows. The servers of each service are configured as if they all part of a single network fabric and when an application sends a packet to an AA for the first time, it is resolved to LA. For this, VL2 agent running in the sources networking stack intercepts the ARP request and converts it to a unicast query to the VL2 directory system. The directory system answers the query with the LA of the ToR to which packets should be tunneled. During this resolution process, the directory server can additionally evaluate the access-control policy between the source and destination and selectively reply to the resolution query, enforcing necessary isolation policies between applications. This process is explained in Fig.2.7.

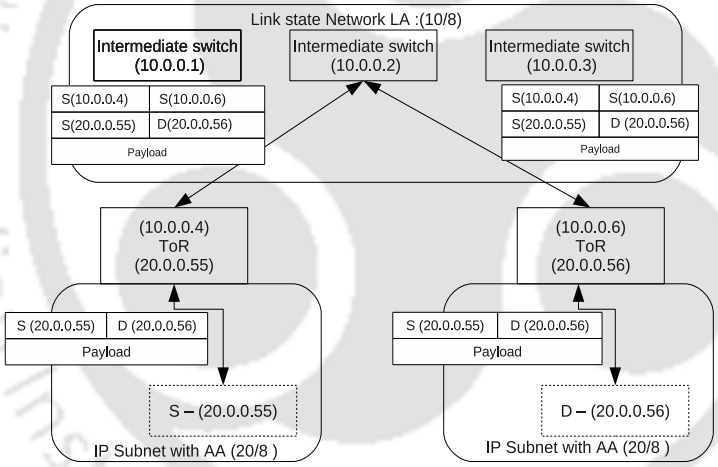


FIGURE 2.7: Routing - Mapping of AAs to LAs in VL2

Thus, VL2 provides load balancing by moving application servers anywhere in the topology according to the traffic condition. But, with respect to a service, it gives an impression that set of servers (either 1 or 10,000) hosting that service are in a single broadcast domain.

2.4.2 Conserving energy in data center networks

There are two important motives for the study of energy for DCNs. First, an increasing energy costs and towards that, DCN contributes 15% of the overall energy consumed by the data centers [6, 7] and the second, time variant resource usage. Data centers are overprovisioned. But, the study of data center traffic shown that keeping all the resources always*on* is needless. Thus, data centers must be build to provide bandwidth for what perhaps a peak traffic that all the servers may collectively generate, but there must be control knobs that turn resources *on/off* according to the traffic condition. ElasticTree [8] provides such mechanism that provides a network wide manager which dynamically turn the resources *on/off* according to the traffic condition.

ElasticTree uses three logical modules for conserving energy: (i) an optimizer accepts a topology, a traffic matrix, a power model for each switch and a desired fault tolerance parameters (spare switches and spare capacities) and returns a minimum power network subset that satisfies the current traffic. An optimizer find a subset of components that satisfy the present traffic and outputs a network subset (ii) routing module then routes the flows to use only the network subset returned from the optimizer. This re-routing may constraint a flow to use a non-optimal path that is very much different from the original path, finally (iii) power control module turn *off* the remaining components that are not part of the network subgraph returned by the optimizer.

ElasticTree used three optimizers - formal model, greedy bin packing and topology aware heuristics and shown that their proposal can save up to 50% of the energy.

The other proposals for conserving energy in DCNs used the architecture features to measure the performance of their proposed methods. Dennis, et al., proposed energy proportional DCN [38] and shown the flattened butterfly topology consumes less power than a folded-Clos networks. Further, they proposed a mechanism to adjust the link rate according to the present traffic. Similarly, in [39], DCN design BCube is proved to be efficient in conserving energy compared to

fat-tree. A general framework for assigning VM based on the hierarchy and symmetry of the architectures is proposed in [64, 65] and their selected sub-network is sufficiently large to accommodate the current set of flows. Since performance of certain distributed applications depends upon the topology characteristics, it is desired to have a framework that helps to realize different DCN topologies with an ease. Towards this, there are proposal to define software-defined networking [66–68] that enable creating different topologies through reconfigurable testbed. This test bed is used to realize various designs for DCN and to study their support for energy optimization.

2.4.3 Load balancing in data center networks

A uniform way of adjusting the window size - additive increase multiplicative decrease (AIMD), slow start, etc that candidly avoided congestion in Internet may not have similar consequences for DCN [69, 70]. Majority of the applications deployed on data centers use workflow pattern similar to partition/aggregate. The real time deadlines for the end results are constraints by the latency targets for the individual tasks. The latency targets will be in the range 10 ms to 100 ms. These latencies are soft deadlines and any task that fail to meets its deadline is simply ignored while computing the final result. This affects the quality of end result. Thus, application requirement for low latency influences the quality of the result and ultimately the revenue. For example, in order to give fast response, the web portal for flight reservation may exempt certain tasks that fail to meet the deadline while getting flight details. The second reason being the varying demands for different flows. As stated earlier, the short flows demand low latency to meet their deadline, the long flow demands may be altogether different. The long flows need much of the bandwidth as they continuously need to update internal data structures of applications as the freshness of the data also influences the quality of the end result. Due to these and many other reasons such as incast problem (fan in burst at the parent port) [71, 72] and use of shallow buffers [73], modifications to the TCP congestion window are proposed.

Data Center TCP (DCTCP): The primary aim of the Data Center TCP (DCTCP) [42] is to make the source to react to congestion in proportion to extent of congestion. For this, DCTCP uses the Explicit Congestion Notification (ECN) mechanism available with the modern switches. DCTCP sets some marking threshold, T. When a particular packet reaches that switch and that result in queue occupancy greater than the threshold, then the switch sets Congestion Experienced (CE) code point of the packet. Next, the DCTCP receiver while acknowledging the packet, sets ECN-echo flag for every marked CE codepoint. Finally, the DCTCP source reacts by reducing the window size by a factor that depends upon the fraction of the marked packets, greater the fraction, larger the reduction.

In order to react to extent of congestion, source estimates the fraction of packets that are marked, α, which then updated for every window of data as follows:

$$\alpha \leftarrow (1 - g) \times \alpha + g \times F$$

Where F is the fraction of packets marked in the last window of data, and g, $0 < g < 1$, is the weight given to present samples against previous estimate for α.

Deadline Driven Delivery (D^3) control protocol: One of the limitations of DCTCP is its deadline agnostic nature. Whenever the flows in the current path experience a congestion, they all backoff their window size according to an extent of congestion. Because of this, all of them might miss their deadlines and produce zero result. Thus, in order to prioritize the flows based on their deadline, Deadline Driven Delivery control protocol (D^3) [43] proposes allocation of buffer sizes in the intermediate switches according to their deadline information.

D^3 assumes for its operation, the availability of flow size (s) and deadline information (d) before hand. The source endhost initiates desired rate $r = \frac{s}{d}$ and the desired rate is carried in packet. The intermediate switches in the path of the destination observes the desired rate and sets the vector indicating amount of buffer that can be allocated to that flow. Next, the receiver endhost, copies this vector in the acknowledgment packet. The sender then adjusts its window size to the minimum value in the vector and sets that as next desired rate and piggyback the desired rate in one of the data packet of next RTT.

Deadline aware Data Center TCP (D^2TCP): Deadline aware Data Center TCP (D^2TCP) [44] proposed to address the limitations of both, DCTCP - a deadline-agnostic that equally throttle all the flows and D^3 - a deadline-aware congestion schemes. The major limitations of D^3 are race condition and backward compatibility. As D^3 tries to assign switch buffers to the flows in a greedy way, the earlier flows with late deadlines may prohibit the allocation of bandwidth for later flows with early deadlines. Due to this race condition, D^3 causes frequent priority inversion which lead to many missed deadlines. Regarding backward compatibility, the switch software has to be changed to do the buffer allocation according to the deadline information carried in each packet. Further, D^3 cannot coexist with the traditional TCP. TCP flows flowing through the same switch as D^3 flows would not recognize D^3's honor-based allocation. Thus, separate Quality of Server (QoS) classes may be created to handle different versions of TCP.

D^2TCP combines the aspects of both DCTCP and D^3 to modulate the congestion window size based on both the deadline information and extent of congestion. It borrows congestion control algorithm from DCTCP and build deadline awareness on top of it. The extent of congestion window size, α, is defined similar to DCTCP:

$$\alpha \leftarrow (1-g) \times \alpha + g \times F$$

Then it defines, d, deadline immense factor, similar to gamma correction function defined for color correction in graphics, where d is inversely proportional to deadline. Minimum the value of d, later the deadline and greater the value of d, closer the deadline. Then, based on the value of α and d, the penalty function, p, applied for the window size is calculated as:

$$p = \alpha^d$$

Finally, the congestion window size, W, is set as follows:

$$\begin{aligned} W &= W \times (1 - \tfrac{p}{2}), && if\ p > 0 \\ &= W + 1, && if\ p = 0 \end{aligned}$$

When α is zero (that is, absence of congestion marked by no marked CE packets), p is zero and the window size is incremented similar to additive increase of TCP. On the other hand, when all the packets are CE-marked, then α=1 and

p=1. This indicates the congestion and the window size is halved. Leaving these two end points, for $0 < \alpha < 1$, window size is modulated by p. Finally, how the priority inversion takes place is explained in Fig.2.8.

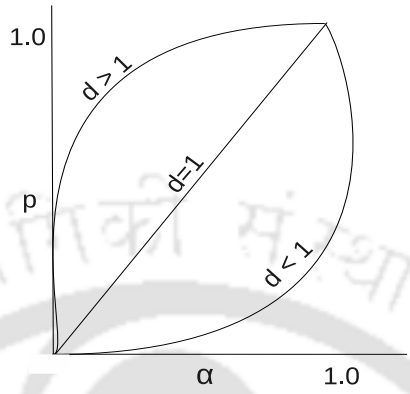


FIGURE 2.8: Gamma correction function of D^2TCP

The penalty function p for different values of d is shown in Fig.2.8. For $d = 1$ (in the absence of deadline information), $p = \alpha$, penalty function depends only on extent of congestion. When $d < 1$, the near deadline flows have lower penalty, shown as line below the straight line. Finally, when $d > 1$, the far deadline flows will have greater penalty, shown as line above the straight line. Thus, the combination of $d < 1$ and $d > 1$ conditions complement each other during congestion. The far-deadline flows make a way for near-deadline flows by backoffing more compared to the near-deadline flows.

There are other proposals that work considering the intermediate switch status for controlling congestion. Detour Induced Buffer Sharing (DIBS) [74] overcomes the congestion at the switch by detouring the packet to other switch. Similarly, Allocation Switch Buffer [75] sets the congestion window based on the available buffer sizes along the path of the flow. Lastly, there are proposals to handle the delay caused by virtualization. Since the flows goes through the hypervisor introduce additional delay, Vsnoop [76] and Vflood [77] are proposed to offload the TCP functionality from the servers to hypervisor.

2.5 Conclusion

In this chapter, we provided survey about various architectures proposed for DCN. Research in the field of data centers attracted large number of researchers and this lead to many designs proposed for DCN addressing different aspects such as VM migration, hybrid technologies, QoS and many others. Hence, in this chapter we provided details about only those architectures which addresses the issues we discussed in this work. The main four aspects we covered in the chapter are the location based routing, source routing, energy conservation and load balancing.

Chapter 3

4-4, 1-4: An Architecture for Simple, Efficient Location Based Routing for Data Center Network using IP address Hierarchy

The survey in the previous chapter showcases the significance of the topology designs in addressing various issues of the data center networks. For example, topology designs DCell and BCube help to devise a source routing that successfully addresses the scalability problem of routing protocols. Similarly, PortLand and Alias derive their location based routing from their underlying topologies. But, there are other proposals for DCNs such as ElasticTree, DCTCP, PDQ and many others that are generalized for any topologies. However, several studies are done to compare their performance on different topologies. For example, Dennis, et al., proposed energy proportional DCN [38] and shown that the flattened butterfly topology consumes less power than a folded-Clos networks. Similarly, in [39], DCN design BCube is proved to be efficient in conserving energy compared to fat-tree. To summarize - design of an efficient architecture holds a great promise in the area of data center networks. In this work, we propose a new network topology that addresses the scalability problem of routing protocols.

The routing protocols proposed for Internet such as OSPF and RIP do not scale for large number of servers that a typical data center hosts. As a result, many alternative routing protocols are proposed to overcome the scalability problem of the routing protocols proposed for Internet [78]. These designs encode locality and topology information into their server and switch addresses for performance and routing purpose. These alternative routing protocols are broadly classified as source routing and location based routing. As illustrated earlier, location based routing determines relative location of the servers in a given topology and uses the location information for routing. PortLand and Alias are the two prominent designs for location based routing. Both PortLand and Alias bestowed on significant exchange of messages between end hosts and intermediate switches to determine relative location of the servers in a given topology. The obtained location information is then encoded in separate addresses called Pseudo MAC and label in PortLand and Alias, respectively. After encoding, location information helps routing from the top level to the bottom level of their respective topologies. We identified following limitations with the existing location based routing.

Large control overhead: Use of separate network control plane and management plane for topology discovery and maintenance in PortLand and Alias introduces large control overhead. The location discovery protocol (LDP) messages exchanged between the servers and intermediate switches to determine their level and their relative position in that level result in significant internal traffic. A typical LDP header is 10 bytes and if an average size of 64 bytes is assumed for LDP message, where data is represented in the form of type length value (TLV), then significant amount of the bandwidth of data center is used by these messages. Data centers typically host millions of servers. Thus, huge control information generated by these large number of servers and intermediate switches may overlap with the original traffic and accounts for performance degradation of the system. The problem is much elevated because, switches periodically send a location discovery message (LDM) out of their all connected ports to set their positions and to monitor liveness.

Half path: Location information, both in PortLand and Alias, helps routing

in a single direction, a 'downward path' from core routers to edge servers. This is because, in both the designs the underlying topologies are tree variants and tree can be best viewed from top to bottom. Both the designs use mechanism other than location based to carry the packets from edge servers to core routers. PortLand uses diffusion and Alias uses virtual circuit interconnection (VCI) for data transfer in 'forward path'. Thus, the designs are not entirely location based because the location information helps to cover only the half path of the route.

Re-Programming: Introduction of new addresses such as labels and PMAC addresses required re-programming of network elements. Since routing is done using newly introduced addresses such as labels and PMAC addresses, re-programming intermediate switches is required to change lookup operation from IP address to newly introduced addresses. Next, the network elements are to be re-programmed to respond to labels/PMAC when they receive ARP request. Further, since the packets are to be augmented with labels/PMAC addresses, the Ingress switches are to be re-programmed to respond with labels/PMAC addresses when they receive ARP requests and the Egress switches must remove the newly introduced address to ensure proper working of layers above and below the network layer.

All these limitations resulted from the underlying architectures the both designs use. The first and third limitations are due to the absence of correlation between the topology design and address assignment. While, the second limitations is due to the use of tree variant designs. The label based routing in these designs represents top-down parsing and it works well because the assigned coordinates are non overlapping and define unambiguous grammar. But, since the same top-down parsing cannot be used for bottom-up, they use other approaches for the first half.

In this work, we use correlation between topology design and address assignment to define our new architecture 4-4, 1-4, wherein location of the network elements is determined directly from their assigned addresses. Thereafter, a routing algorithm is designed to use the location information that covers entire path of the route. Since, the location information is determined directly from the addresses of end hosts, our proposed routing algorithm eliminates complexities of the earlier

routing algorithms in determining location information.

This work is organized as follows. In the next section, we propose our new architecture 4-4, 1-4 and the location based routing which uses 4-4, 1-4 to define its routing. In Section 3.2, we discuss various properties of 4-4, 1-4. The simulations conducted and the results obtained are given in Section 3.3. Finally, we conclude with Section 3.4.

3.1 4-4, 1-4 architecture

Our idea of deriving location information from the address is inspired from Locator/ID separation protocols (LISP) [46, 47, 79] which correlate the hierarchy of topology design and address assignment. LISP separates *where* and *who* in networking and uses a mapping system to couple location (*where*) and identifier (*who*). LISP addresses are respectively known as endpoint identifiers (EIDs) and the routing locators (RLOCs). RLOCs are assigned according to the topology design that enable identifying the end host through its address. During the routing, identifier (*who*) is mapped to location (*where*) and routing is done based on location information. We briefly describe working of LISP, proposed to solve the scalability problem of IP.

The current analysis have shown there are presently 2.92 billion internet users world wide [80] and their number is growing per year. This rapid growth in the Internet challenges the scalability problem of IPv4 protocol and Locator/ID separation protocol (LISP) is one such solution which improves the scalability of Internet routing tables. For this, LISP creates two addresses for each network node, one for its identity and another for its location in the network. While the traditional IPv4 uses a single address space, an IP address, for the both. These addresses are respectively known as Endpoint Identifiers (EIDs) and the Routing Locators (RLOCs).

Next, to speed up the routing, LISP constructs a separate hierarchical topology and assigns RLOCs based on this hierarchy. For example, higher prefixes are used for addressing the Continents, the next level prefixes are used to address the

Countries within the Continents, and so on. Thus, when the ARP request tries to resolve EIDs, the mapping system maintained at some designated switches (ingress switches) intercept these ARP requests and return the corresponding RLOC. Since RLOCs are assigned based on location information, the packets are tunneled to reach the required destination by using a separate interconnections system established for faster routing. Finally, at the destination (Egress routers), the RLOCs are mapped back to EIDs. Working of LISP based on two address spaces is shown in Fig.3.1.

Thus, LISP provides two benefits - routing will be faster because of the hierarchical topology and the number of entries in a routing table are considerably reduced because the hierarchical address assignment enables a high degree of address aggregation.

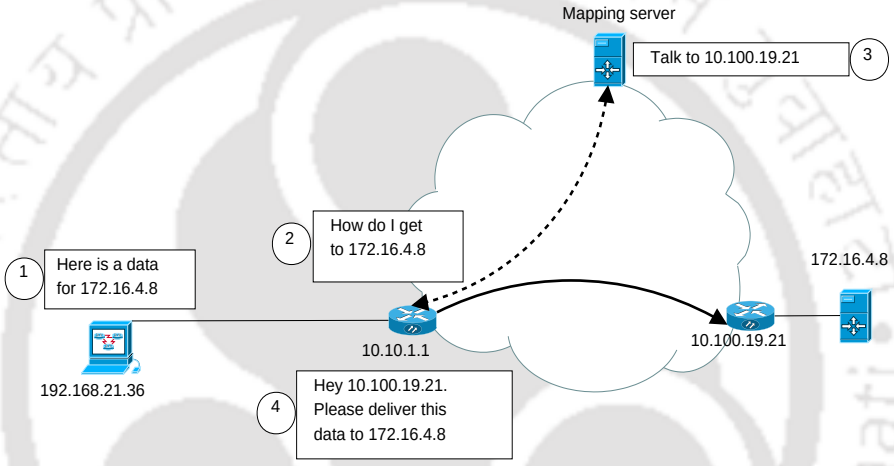


FIGURE 3.1: LISP first resolves EID of the destination host to RLOC and then uses the RLOC for routing

LISP is forced to do this separation because re-assigning IP addresses for an existing system is tenacious. But, as data centers are private, this separation can be avoided. Thus, we used only that part of the LISP, assign addresses to end hosts such that their location can be determined through their addresses. This is possible because both the topology design and address assignments follow the hierarchical design. Based on this principle, we propose our new architecture 4-4, 1-4.

Further, following observations about the topology designs for DCN and IP address hierarchy consolidated the design approach for 4-4, 1-4 architecture and our proposed routing algorithm.

- The growth model of DCN designs resembles sub-netting and super-netting of classless inter domain routing (CIDR) and variable length subnet mask (VLSM) [81].
- In IP address hierarchy, when smaller subnets aggregate to form a larger subnet, they all share a common prefix among them.

The first observation is made from the study of topology characteristics of fat-tree, DCell, BCube and many other designs for DCN. All these designs present their design by giving baseline topology and growth model. Essentially, the hierarchical growth model defines expansion of the respective design in an orthogonal space and resembles super-netting feature of IP.

Next, we devise a routing algorithm from the second observation. Since all the $level_i$ subnets of $level_{i+1}$ share a common prefix, they all have the same prefix encoded upto $level_{i-1}$. Thus, the routing decision is made based on the bits used for encoding $level_i$ subnet. For this, we assign the addresses for switches connecting $level_i$ subnets at $level_{i+1}$ from the address range of $level_i$ subnet. This helps switches at $level_i$ to determine next hop for a packet by comparing their address with the destination address.

Thus, using a single address space for both identifying the end hosts and routing we achieved the following advantages.

- No extra control overhead for determining location information of the end hosts.
- Covering entire path of the route by only the location information.
- Easy deployable on proven network technologies with a very minimum re-programming.

3.1.1 Topology design

The design of 4-4, 1-4 is explained through baseline topology and growth model. The baseline topology, $level_0$ subnet is formed from 1024 server. The 1024 servers

are arranged in the form of a pod of fat-tree. The 1024 servers are divided into 32 groups and each group of 32 servers are connected to a switch called pod switch. As there are 32 groups of switches, a row of 32 pod switches connected to 32 different group of servers forms a first row of aggregation layer. As in fat-tree, these 32 switches are connected to a set of 32 switches at the second row of aggregation layer. This entire interconnection scheme provides 1:1 oversubscription at $level_0$ subnet. We use 64-port switches for our design. With respect to switches at the first row of aggregation layer, among 64 ports, 32 ports are connected to 32 servers and the rest, 32 ports, are connected to 32 switches at the second row of aggregation layer. The design of $level_0$ subnet of 4-4, 1-4 architecture is shown in Fig.3.2.

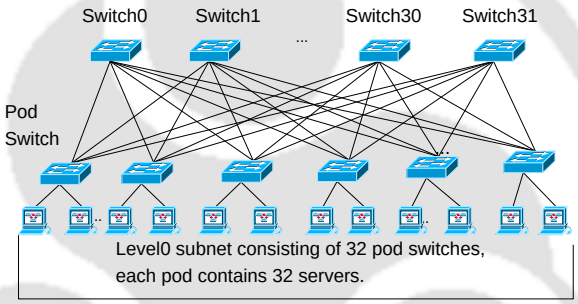


FIGURE 3.2: $Level_0$ subnet designed as a pod of fat-tree

Next levels, $level_i$, $i \in [1, l]$ are designed by combining four $level_{i-1}$ subnets in the form of complete connected graph. The $level_1$ subnet is formed from four $level_0$ subnets. As there are 32 TOR switches at the aggregation level of each $level_0$, four j^{th} switches, $j \in [0, 31]$, one from each $level_0$ subnets, are combined in the form of fully connected graph called cycle. Design of $level_1$ subnet from four $level_0$ subnets is given in Fig.3.3. In the figure, only four out of 32 cycles are shown. Further, each cycle of $level_1$ subnet is shown with different color.

Since we used 64 port switches for our design, 35 ports of the switches at the second row of $level_0$ are used for interconnection. Among 35 ports, 32 ports are connected to 32 switches at the lower layer of $level_0$ subnet and remaining three ports are used for forming a cycle. Therefore, we are left with 29 ports unused for each switch of a cycle. We use these ports for defining the next level, $level_2$. Four unused ports, one from each switch of a cycle are connected to a switch called

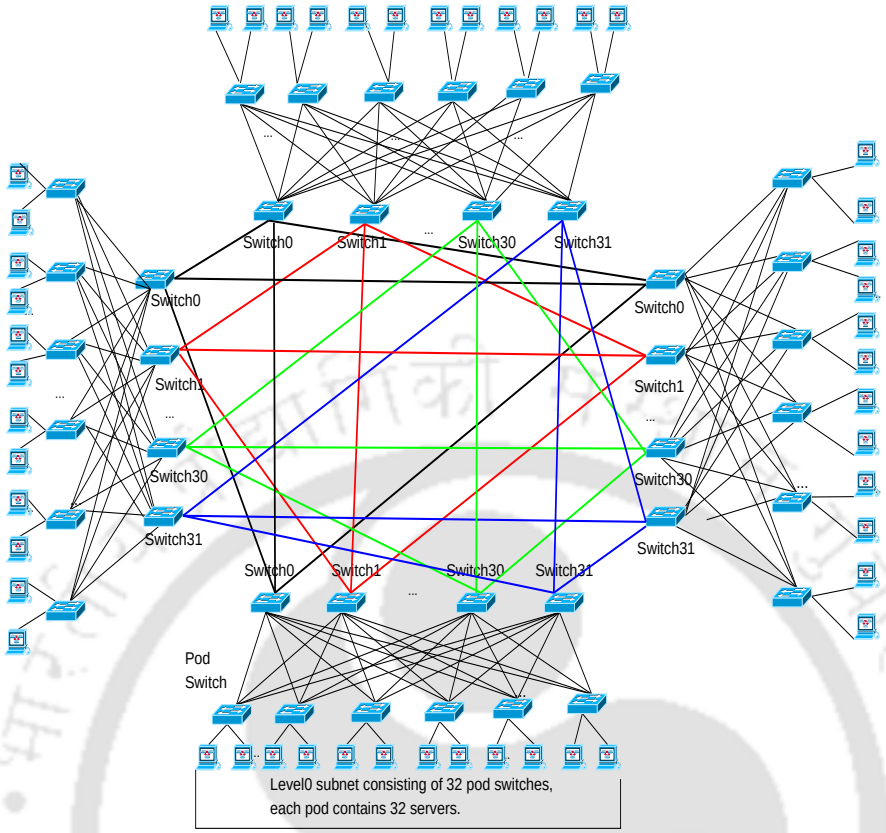


FIGURE 3.3: $Level_1$ subnet formed by combining four $level_0$ subnets.

fusion switch. As there are 29 unused ports, each cycle from a $level_1$ subnet are connected to 29 different fusion switches. Fig.3.4 shows interconnection of four switches of a cycle to 29 different fusion switches. The idea behind a cycle with a fusion switch is as follows. If a switch at $level_i$ receives a packet belongs to another $level_i$ subnet, then it forwards the packet to corresponding $level_i$ subnet in that cycle. In case the packet belongs to higher subnets, $level_{i+1}$, $level_{i+2}$,...,$level_l$, then it is forwarded to the fusion switch. Finally, as there are 32 cycles at $level_1$ subnet and each of the cycle is connected to 29 different fusion switches, there are $32 \times 29 = 928$ fusion switches from each $level_1$ subnet. Interconnection of these many fusion switches provides better oversubscription ratio that helps to distribute the incoming traffic evenly among the outgoing links. Since the traffic coming from 32 different switches is distributed among 29 fusion switches, this provides an oversubscription of 1:1.1034 at the fusion switch.

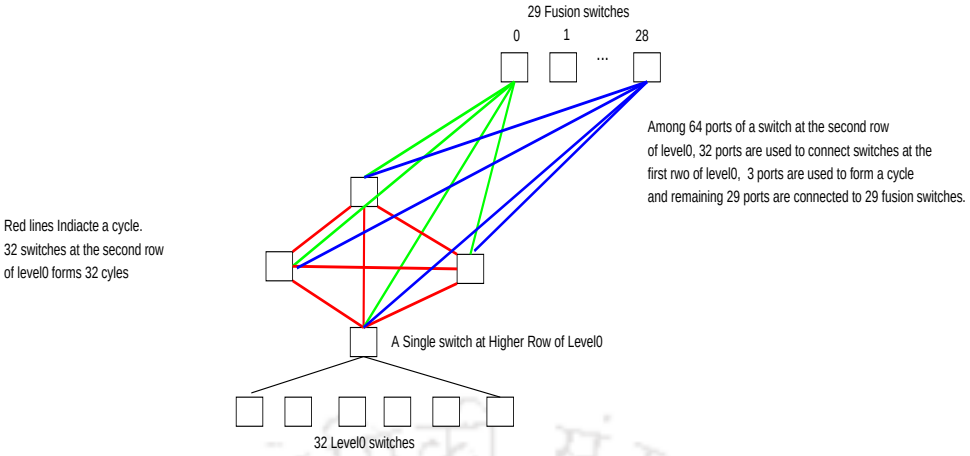


FIGURE 3.4: A cycle from $level_1$ subnet connected to 29 different fusion switches.

Next levels, $level_i$, $i \in [2, k]$, are designed by combing four $level_{i-1}$ subnets. Design of these levels follows the design of $level_1$ subnet wherein fusion switches from the previous levels are combined in the form of cycles. For example, design of $level_2$ subnet four fusion switches each from four different $level_1$ subnets ate interconnected to form a cycle. As there are 928 fusion switches in each $level_1$ subnets, there will be 928 cycles at $level_2$. A detailed version of $level_1$ subnet of 4-4, 1-4 is given in Fig.3.5.

In 4-4, 1-4, number of server supported at $level_0$, $level_1$ and $level_2$ are 1024, 4096 and 16,384 servers respectively. A growth model of this type has both advantages and disadvantages. In fat-tree, number of servers supported is directly proportional to port-density of the switches used. For example, fat-tree with k-port switches supports $\frac{k^3}{4}$ servers. Thus, fat-tree needs 102-port switches to design a data center of size 2,62,144 servers. Since port-density is one of the parameter in deciding the cost of a switch, a higher port-density switches incur more cost. Moreover, major expansion in fat-tree may result in redesigning whole fat-tree using a higher port switches. Contrast to this, port-density of the switches used in 4-4, 1-4 is always 64. Further, the hierarchical nature of 4-4, 1-4 design enable to provide additional servers by designing the next level. On the downside, hierarchical nature of 4-4, 1-4 results in more levels than fat-tree. To support 2,62,144 servers, it needs five levels, two levels more than fat-tree. However, the increase in path length affects only the servers which are far apart. For example, when

a source and destination belongs to two different $level_2$ subnets then resulting path spawns one level more than fat-tree. Thus, 4-4, 1-4 results in additional path length when the source and destination are separated by more than two levels. Although application placement problem, placing applications on a hosting platform is NP-hard [82], but it always make sense to place servers hosting related components of an application together.

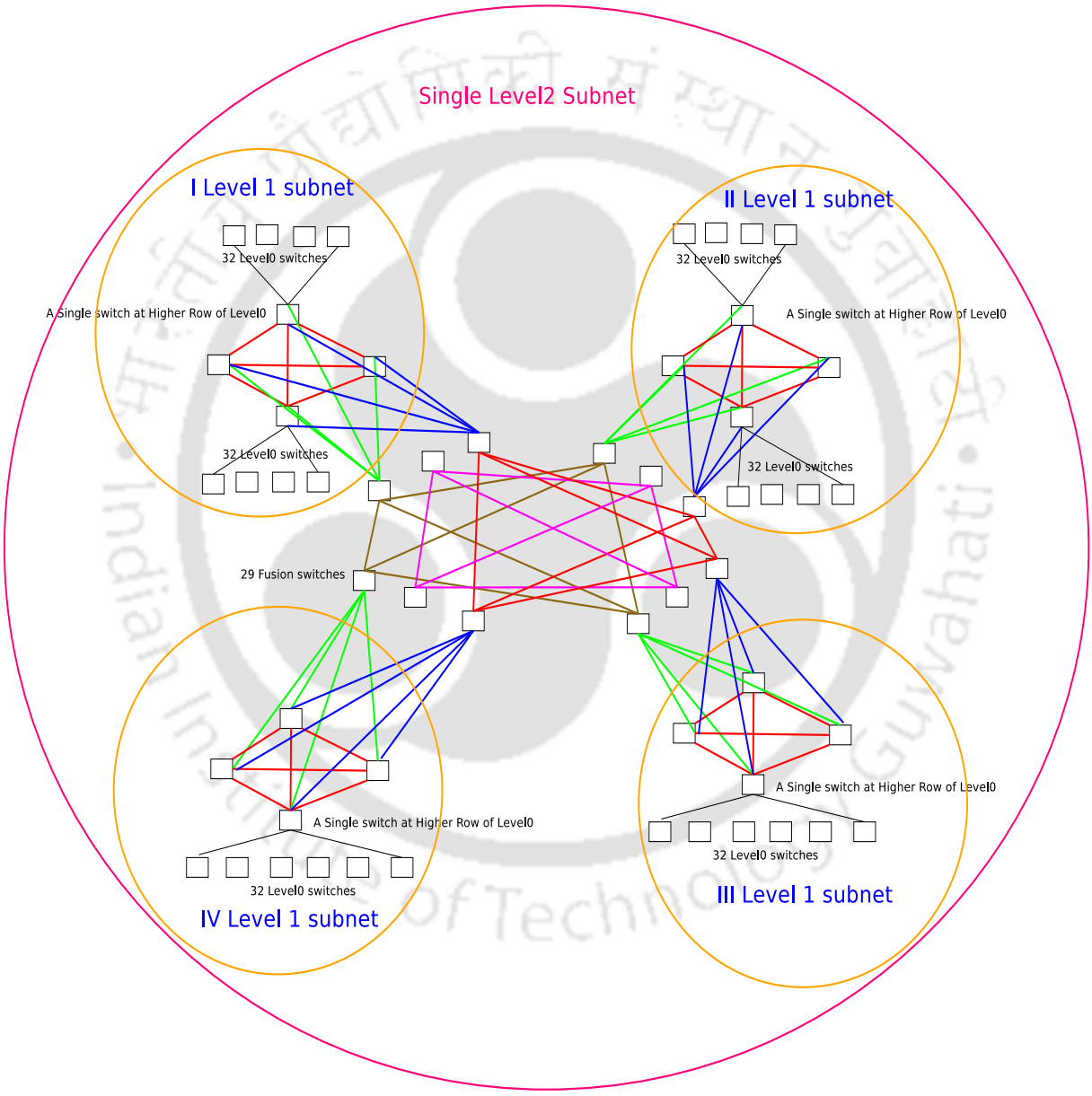


FIGURE 3.5: Providing oversubscription in 4-4, 1-4 architecture using a large number of fusion switches.

3.1.1.1 Addressing

To explain addressing, we use X to denote an octet and x to denote a bit of an octet. As there are 1024 servers at $level_0$ subnet, last 10 bits (bits 9-0) of IP address are used to address the servers and intermediate switches of $level_0$. Next, since $level_1$ subnet is formed by combining four $level_0$ subnets, the next two bits ($2^2 = 4$) (bits 11-10) are used to address the four $level_0$ subnets of a $level_1$ subnet. Similarly, further levels $level_{2,3,4...l}$ are addresses using bits (13-12, 15-14, 17-16, ...$l2$-$l1$) of IP address. Here $l1 = 10 + (l-1) \times 2 + 1$ and $l2 = l1 + 1$. Fig.3.6 shows hierarchical aggregation of IP addresses to define different level subnets.

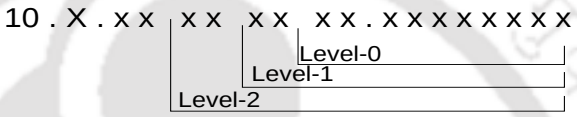


FIGURE 3.6: Addressing servers at different levels using the recursive format of IP address.

The arrangement of switches upto $level_2$ is shown in Fig.3.7. S1, S2, S3 and S4 are the switches at the lower row of $level_0$ subnet. Along with the switches, we denoted a range of $level_0$ addresses corresponding to that switch. These addresses can be easily derived from the topology. We indicated along the intermediate switches, values the bit positions of IP address takes in that direction. For example, switch S4 is connected to a switch labeled as 10 at $level_2$ and to a switch labeled as 10 at $level_1$. Thus bits 5,4 takes value 1,0 and bits 3,2 takes values 1,0. By this, a range of IP addresses assigned to the machines connected to S4 is determined as [10.X.00 10 10 00.00000000 to 10.X.00 10 10 11.11111111] and its decimal equivalent is [10.X.40.0 to 10.X.43.255].

Addresses in this hierarchical manner are much easier to assign using dynamic host configuration protocol (DHCP). By specifying the range of address along with the subnet mask, every servers and switches at a particular subnet are assigned addresses. For example, the four $level_2$ subnets are assigned addresses in the range 10.201.0.0/18, 10.201.64.0/18, 10.201.128.0/18 and 10.201.192.0/18. Further, four $level_1$ subnets in 10.201.0.0/18 are assigned in the range 10.201.0.0/20,

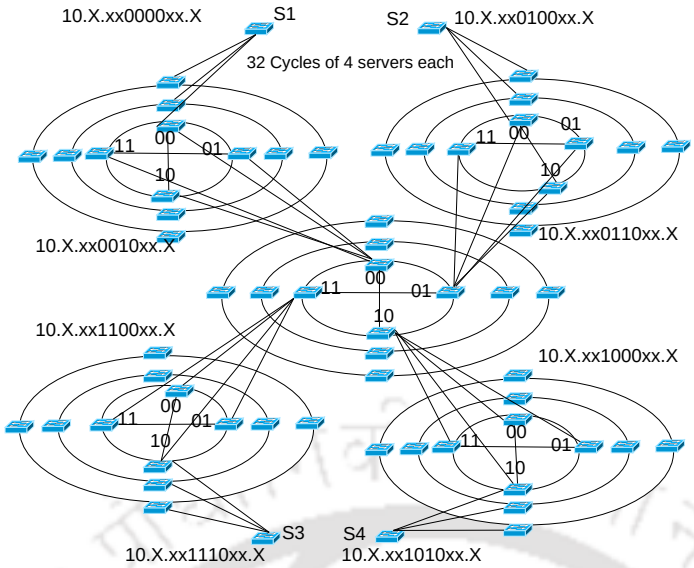


FIGURE 3.7: A simple layout of 4-4, 1-4 architecture. In the figure, the address assigned to the end hosts can be derived by hierarchically aggregating the values the bits used for encoding different levels.

10.201.16.0/20, 10.201.32.0/20 and 10.201.48.0/20. Similarly, four $level_1$ subnets in 10.201.64.0/18 are assigned in the range 10.201.64.0/20, 10.201.80.0/20, 10.201.96.0/20 and 10.201.112.0/20. Since the addresses assigned are hierarchical, the network addresses and subnet masks are easily computed. Further, as only requirement is to assign the address to the network elements from a specific address range irrespective of any ordering in address assignment, DHCP can be effectively used to assign addresses for network elements of 4-4, 1-4.

Assigning addresses using DHCP is much easier compared to other assignment proposed in a dynamic address configuration (DAC)[78]. DAC derives the logical ID's such as IP addresses for the network elements using the blue print provided for the design and the physical topology using graph isomorphism. For this DAC first learns the physical topology labeled by device ID's such as MAC addresses. Then it maps the physical topology to logical topology derived from the blue print to find the logical ID *e.g.,* IP address.

3.1.2 Routing

Large traffic in data centers and very fast links demand greater performance from the intermediate switches. Performance of IP address lookup is thus critical for faster packet forwarding [81]. The main idea behind our proposed location based routing is to take packet to the larger subnet of the destination host and from there descend to exact smaller subnet where the destination host exists. This is done by assigning address to each switch from the address range corresponding to the largest subnet to which it belongs. Thus, the address assigned to the switches act as a location indicator during routing and the forwarding decisions are made by comparing certain bits of switch address with the destination address.

Since the address assignment in 4-4, 1-4 follows IP supernetting, the address assigned to servers at $level_i$, $i \in [1, l]$, have the same prefix encoded upto $level_{i+1}$. Thus, at every intermediate level switch, the decision about the next hop is done by comparing the intermediate switch address with destination address. For comparison, starting from the rightmost bit of IP address, the bits used to encode upto $level_i$ subnets are masked and compared.

Let X_i and Y_i be the remaining bits of destination and intermediate switch addresses after masking. If both X_i and Y_i are not equal then the destination belongs to other $level_{i+1}$ subnet and the packet is forwarded to a fusion switch. Instead, if both X_i and Y_i are equal then the destination belongs to same $level_{i+1}$ subnet. Now, to locate destination host within that subnet, bits used to encode only the $level_i$ subnets of intermediate switch and destination addresses are compared.

Let P_j and Q_j be the bits used for encoding only the $level_i$ subnet. If P_j and Q_j are equal then the destination belongs to same $level_i$ subnet and is forwarded to the $level_{i-1}$ subnet of $level_i$. If P_j and Q_j are not equal then the destination belongs to other $level_i$ subnet and thus forwarded to the corresponding $level_i$ subnet through the cycle. This process is continued at each level until the packet reaches a TOR switch. The TOR switch then forwards the packet to correct destination. The pseudocode for forwarding logic at a $level_1$ switch is given in Algorithm 3.1.

Algorithm *Next-Hop-At-Level1 (Swaddr, Daddr)*

Input: ***Swaddr***(Switch Address), ***Daddr***(Destination Address)

Output: *Next hop in forwarding path*

Let X_i = Swaddr & 255.255.240.0

{***255.255.240.0 is netmask used at*** $level_i$}

Let Y_i = Daddr & 255.255.240.0

if $X_i \neq Y_i$ **then**

Forward the packet to the fusion switch

{***Packet belongs to other*** $level_2$ ***switch***}

else

P_i = bits 3,2 of Third Octet of Swaddr

Q_i = bits 3,2 of Third Octet of Daddr

{***Packet belongs to same*** $level_1$ ***subnet***}

if $P_i == Q_i$ **then**

Forward packet to a downward switch

{***Packet belongs to same*** $level_0$ ***subnet***}

else

Forward packet to other $level_0$ subnet of a *cycle* based on value of P_i

{***Packet belongs to other*** $level_0$ ***subnet within the cycle***}

end if

end if

Algorithm 3.1: **Algorithm for forwarding packet at** $level_1$ **subnet switch**

Fig.3.8 shows routing from source S to destination D. Path taken by a packet is highlighted in the figure. We indicated along the path, the intermediate switch and destination addresses compared at each level. Further, bits compared at each level are also highlighted. As shown in figure, the source and destination address are 10.2.19.9 and 10.2.59.9 and their binary equivalents are 10.2.00010000.9 and 10.2.00111011.9 respectively. The following are the steps in forwarding the packet to the destination.

1) The servers at $level_0$, after determining the destination does not belongs to its broadcast domain, forwards the packet to TOR switch. Let the address of TOR switch be 10.2.16.3. The TOR switch determines the next hop for a packet by comparing its address with the destination address. For this, last 12 bits of both the addresses are bitmasked with 255.255.11110000.0 and compared. After the masking, the bits of TOR switch and destination compared are 10.2.0001.0 and 10.2.0011.0 As both the addresses are different, TOR switch simply forward the packet to the TOR switch at the second row of the pod.
2) Let the switch address at the top row of the pod be 10.2.16.2. Logic similar to the step (1) is used at this switch and the packet is forwarded to a fusion switch.
3) Let fusion switch address be 10.2.16.1. At $level_1$, netmask used is 255.255.11000000.0 and after masking, the switch and destination address are 10.2.0.0 and 10.2.0.0 respectively. Since both the addresses are equal, the packet must be descended to reach the destination host. This is done by using the bits used for encoding $level_1$ subnets of both the switches. Since their values are 01 and 11 respectively, the packet is forwarded to $level_1$ subnet with the value 11.
4) At step four, as the bits used for encoding $level_1$ subnet of both the switches are 11, the packet descended to reach the TOR switch at the top row of the pod and from there to the destination host in the steps (5) and (6).

3.1.2.1 Loop free path

It is easy to show that the entire path is loop free. In forward direction, packet moves from $level_0$ subnet to higher levels searching for a largest subnet to which destination belongs. For this, in forward direction, bits from last octet to first octets are masked and compared. That is, masking is done from right to left. This stops at some level because hierarchical aggregation is done by combining smaller subnets to form larger subnet. Once the packet reaches to the largest subnet to which the destination server belongs, packet proceeds in a reverse direction. In the backward direction, we mask bits from left to right. This ensures, the packet ultimately reach to the smallest subnet at which the destination resides. Loop

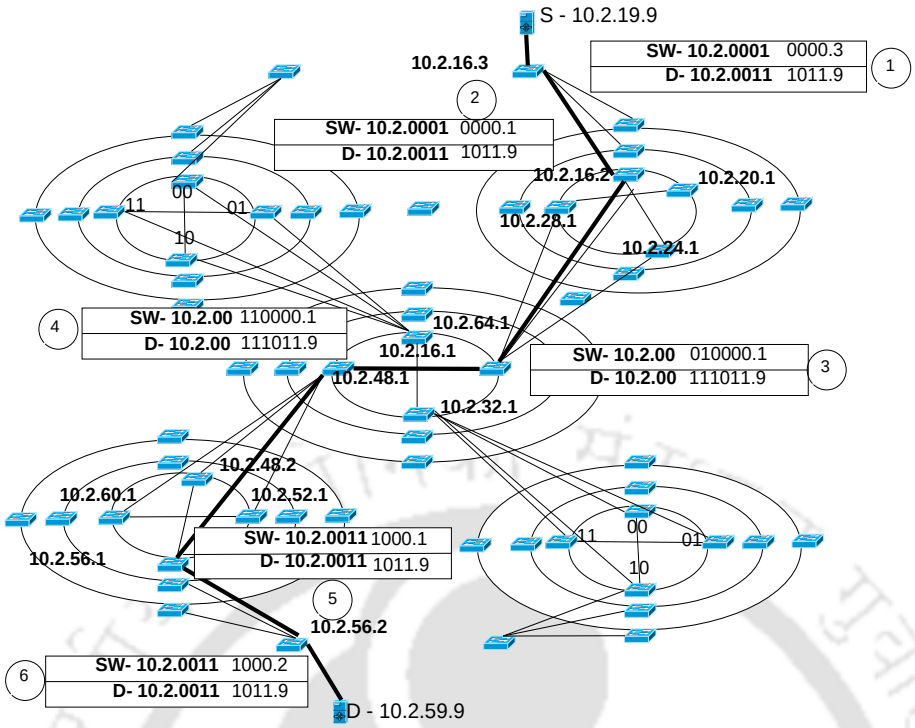


FIGURE 3.8: Routing in 4-4, 1-4 architecture

free forwarding is ensured because we can only aggregate smaller subnets into larger subnet provided they all have a common prefix. Since we follow the similar convention in the design, we can ensure loop free forwarding.

3.1.2.2 Covering entire path

Unlike existing location based routing schemes, our proposed routing uses only the location information for entire routing. A uniform way of routing is used in both the direction - the forward direction, from the edge switch to larger subnet, and the backward direction, from the larger subnet to ultimate destination. At every hop, routing decision is taken by comparing the switch address with the destination address. Compared to our proposed scheme, existing location based routing schemes cover only the later half of the route and for covering the first half they use mechanism other than location based routing such as diffusion and VCI. Thus, a uniform way of routing in our proposed routing overcomes difficulty of

distinguishing two directions for the route and adopting suitable routing for each direction.

3.1.2.3 Deployable on existing technology

Earlier proposals required major re-programming of network elements as routing decision are made based on newly introduced address schemes such as labels and PMAC. But, in our proposal, the network elements can use their supporting communication protocols, as routing is done based on IP address. However, the minimum re-programming effort is required to change forwarding logic from IP look up for the destination address to comparing the destination address with the switch address. This can be easily implemented with click modular router [83], which allows re-programming only the required aspect of a switch. Further, as IP addresses are used for identifying the end hosts, other functions such as ARP query/response need no modifications.

3.1.2.4 Fault tolerant routing

Fault Tolerant routing provides high levels of availability and reliability in network connections during hardware failure or a routing outage or change. In a large systems such as data centers, failures of network components is a commonplace and thus the performance of routing algorithms depends upon how fast these algorithms recover from failures. The fault tolerant routing for our proposed routing algorithms is implemented through a centralized fabric manager that monitors the network element (link/node) failures and informs the affected switches to restrain from using that path for the routing. The three steps implementation of our fault tolerant routing are - in step (1), when a network component goes down, one of its neighboring switch informs the fabric manager about the failure. In step (2), the fabric manager updates its fault matrix and informs the set of switches which are affected by this failure to restrain from using that path and finally, in step (3), the switches reconfigure their routing tables to eliminate the routing entry

suggested by the fabric manager. As our network elements are implemented using NS-3-click modular router, routing tables can be easily updated by adding/removing the routing entries.

Similar operations are performed when the network elements comes up. The routes removed when that elements gone down are updated again when they come up. Thus, handling the network failures is much easier compared to other routing schemes as it does not involves any re computation of routes.

3.2 Discussion

Design 4-4, 1-4 apart from being a better architecture for location based routing, exhibits other features that are equally essential for today data centers. In this section, we qualitatively state those feature. Our future direction for research is quantitatively proving these. Before giving those details, we give insight for proposing 4-4, 1-4 architecture.

3.2.1 Insight for proposing 4-4, 1-4 architecture

In the existing designs for DCN, we examined the designs exhibiting the correlation between topology design and address assignment and found BCube and fat-tree closely exhibiting this correlation due to their regular growth model. But, as we found certain limitations with both the designs, we derived 4-4, 1-4 by combing the positive features of BCube and fat-tree.

3.2.1.1 Number of parallel paths

For large systems like data centers, applying traffic engineering for load balancing is a more involved and hence localized solutions such as valiant load balancing (VLB) [84] are considered better due to their simplicity (more about VLB is explained in following sub section). However, VLB needs support from the underlying architecture for effective load balancing. One of the desired feature is the architecture

providing 1:1 oversubscription so that total incoming traffic is distributed over outgoing links.

Location based routing in PortLand and Alias perform better over fat-tree as fat-tree ensures $\frac{k}{2}$ parallel paths between any two switches of successive levels. Factor k is large in fat-tree and is usually in the range 48 to 64. Contrast to this, the maximum number of multiple paths between two end hosts in BCube is $k+1$, where k is a number of levels. The typical value for k will be in the range 4 to 6. Thus, to effectively use VLB for load balancing, fat-tree is a preferred choice over BCube.

3.2.1.2 Growth model

As mentioned earlier, PortLand uses the routing based on location information for the later half of the route only. PortLand uses the location information that gives the relative position of a child node with respect to its parent for the later half of the route. Doing the same for the earlier half requires an entire different address space where the tree is traversed from bottom to top. To avoid this complication, PortLand uses the location information for the later half but for the earlier half, it uses diffusion as proposed in fat-tree. On the other hand, in hierarchical designs like BCube a uniform routing is used. The primary reason is, the hierarchical growth model of BCube resembles IP supernetting and our proposed location based routing is best suited for designs of this kind.

3.2.1.3 Approach taken by 4-4, 1-4

The $level_0$ subnet of 4-4, 1-4 follows the design principles of a pod in fat-tree. The pod of fat-tree is a preferred choice for designing $level_0$ of 4-4, 1-4. This is because, a pod of fat-tree provides 1:1 oversubscription irrespective of the number of servers it supports. Since there are 1024 servers at the $level_0$ of 4-4, 1-4, pod of a fat-tree is preferred choice. But, for designing next levels, BCube like design is a preferred choice. However, since connecting subnets based on design principles of BCube

does not provide enough parallel paths, we use cycle/fusion switch concept which provides better oversubscription through a large number of interconnections at the subnets above the $level_0$ subnet.

3.2.2 Comparison

Comparison of data center architectures is done based on cost and performance [16]. We compare our proposed design 4-4, 1-4 against fat-tree, BCube and DCell for cost and latency. We consider data centers of different sizes for comparison. For designing fat-tree, we use 48-port switches. For designing DCell and BCube, we use 8 NIC cards with each server. For our design, we use 64-port switches. Further, we compare our design against Portland and Alias for the complexity involved in determining the location information.

3.2.2.1 Cost

The cost of the data center is estimated based on setup cost that involves equipment cost and operational cost. Here, we compare the cost of different designs based on the equipment cost. In cost comparison, the number of switches needed is estimated based on the number of servers and total cost is estimated based on the cost of total switch ports. Popa et.al [16] estimated cost of 10 Gbps Ethernet switch as $450, a 10 G Ethernet port as $150 and a server core cost is estimated as $200. Based on this cost estimate, the cost comparison for different designs is given in Fig.3.9. The cost estimation of our design is comparable with that of a fat-tree.

3.2.2.2 Latency

Latency is measured in terms of average and the maximum path length of the data center with varying number of servers. For 4-4, 1-4, we kept 1024 servers at $level_0$ and number of servers supported at $level_i$ are four times the number of servers

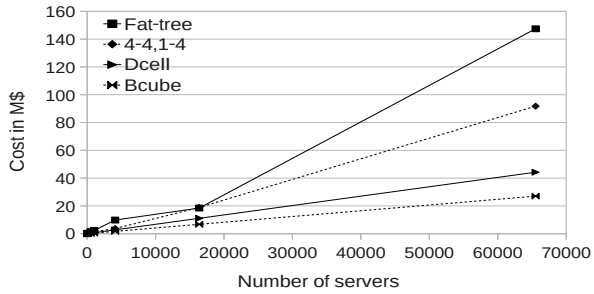


FIGURE 3.9: Total cost of interconnection

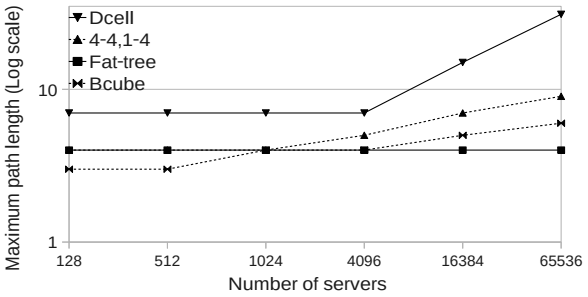


FIGURE 3.10: Path length against the number of servers

supported at $level_{i-1}$ subnet. For fat-tree, the number of levels is always three. For the BCube and DCell, we used their respective growth models to decide their height. The graph in Fig.3.10 shows maximum path lengths against the number of servers for each design. The maximum path length in 4-4, 1-4 architecture is comparable with fat-tree upto the $level_1$ subnet supporting 4096 servers. But, due the hierarchical aggregation of subnets, maximum path length increases by two for each extra level.

3.2.2.3 System comparison

We compare our proposed location based routing with other location based routing designs such as PortLand and Alias. The comparison is done based on additional complexities required in PortLand and Alias for performing location based routing. This mainly includes the number of control frames exchanged between the network elements to establish their level and their relative position in the topology. We estimated the total control frames required to determine the location information based on the number of intermediate switches present at each level and the

Design	Topology	Address constructed for Location information	Complexity of determining Level	Complexity of Constructing the Address	Mapping	Routing algorithms
PortLand (k-array fat-tree)	Mutli-rooted tree	Psuedo MAC	O(3 * k^3/4)	O(k^3/4)	IP address to PMAC Address	Diffusion (Forward) Location based (Backward)
Alias (General Tree)	General Tree.	Labels	O(Number of levels * Nodes at each level)	O(Number of nodes in a tree)	IP Address to Labels	VCI (Forward) Location based (Backward)
4-4,1-4 BCube-IP	IP address hierarchy	IP addresses	None	None	None	Location based (Forward) Location based (Backward)

TABLE 3.1: System Comparison

Number of machines			Number of switches											
4-4,1-4	Fat-tree	Dcell (n_{Switch}=8)	Leve1 Switches			Level2 Switches			Level3 Switches			Total Switches		
			4-4,1-4	fat-tree	DCell	4-4,1-4	fat-tree	DCell	4-4,1-4	fat-tree	DCell	4-4,1-4	fat-tree	DCell
1024	1024(k=16)	1000	64	256	125	--	64	--	--	--	--	64	320	125
4096	5488(k=28)	4000	256	784	500	128	196	--	--	--	--	256	980	500
16384	16000(k=40)	16000	1024	1600	2000	1536	400	--	128	--	--	2560	2000	2000
Number of machines			Number of Connections											
4-4,1-4	Fat-tree	Dcell (n_{Switch}=8)	Level1 Connections			Level2 Connections			Level3 Connections			Total Connections		
			4-4,1-4	fat-tree	DCell	4-4,1-4	fat-tree	DCell	4-4,1-4	fat-tree	DCell	4-4,1-4	fat-tree	DCell
1024	1024(k=16)	1000	1024	1024	496	--	1024	91	--	--	--	1024	2048	587
4096	5488(k=28)	4000	4096	5488	1990	8192	5488	1525	--	--	--	12288	10976	3515
16384	16000(k=40)	16000	16384	16000	7993	21768	16000	7890	--	--	6	38152	32000	15889

TABLE 3.2: Total switches and interconnections required to support different number of servers

number of times the control frames exchanged between the switches to establish their position. For the PortLand whose underlying architecture is fat-tree, first, nodes at $level_0$ learn their levels, then the nodes at $level_1$ and $level_2$. Thus, the complexity of learning levels is three times the number of servers and intermediate switches. Next, as every node must get address by communicating with its neighbors, the complexity of the address assignment is approximated to the number of network elements. The same idea is used for Alias by approximating the number of elements to the number of nodes in a tree. These details along with the other advantages of our location based routing are summarized in Table 3.1.

However, few of the shortcomings of the proposed architecture are, 1) The architecture is not possible to describe mathematically as fat-tree, BCube and DCell and hence there is scope for improving its design, and 2) The number of switches and interconnections needed are relatively more compared to other designs as shown in Table 3.2.

3.3 Simulations

Performance comparison is done using NS3 simulator. For the performance study, we constructed 4-4, 1-4 supporting 4096 servers, fat-tree supporting 4394 (s=26) servers, $BCube_5$ supporting 4096 (m=4) and a partial DCell supporting 4200 servers. In order to keep the levels of 4-4, 1-4 and fat-tree same, we constructed a three level 4-4, 1-4 wherein we restricted the number of servers connected at $level_0$ to 256. Lastly, we constructed a partial DCell supporting 4200 servers because a complete $DCell_3$ supports 176820 servers which is much beyond our required number of servers. This has resulted in additional performance penalty for DCell since the routing in partial DCell is more involved compared to the routing in complete DCell [28]. For the simulations, we used all the links with bandwidth 5 Mbps and delay 5 ms. About routing, for fat-tree, two way look-up routing is implemented. For BCube and DCell, their respective source routing are implemented. Finally, for 4-4, 1-4, our proposed location based routing is implemented. For this, we configured both the hosts and switches of 4-4, 1-4 with click configuration [85] which allows re-programming only the required aspect of the routing.

3.3.1 Routing time

We measured the convergence time for routing protocols to show our proposed scheme's ability to eliminate the need for any additional control overhead. For the fat-tree, we measured the time taken by simulation to converge routing table of all the nodes. For BCube, DCell and 4-4, 1-4, we made every node of an architecture to send data to every other node and we measured the time taken for computing routes to all the destinations. Total time taken for routing algorithm to converge routing tables at all the levels is given in Fig.3.11. From the figure, it is clear that 4-4, 1-4 does better compared to other architectures as it does not need any prior computation of routes. 4-4, 1-4 can start transmitting data instantaneously as the route is determined at successive hops by comparing the address of the switch at that hop with the destination address.

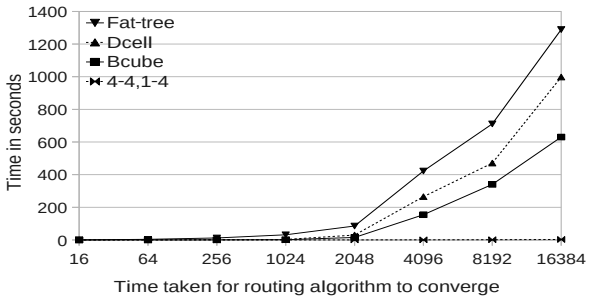


FIGURE 3.11: Time taken for routing algorithm to converge

3.3.2 Throughput and delay

Next, we compare the performance of our proposed routing algorithm against the routing algorithm for fat-tree, BCube and DCell. For the performance study, we used the realistic data center traffic provided by Theo Benson and Aditya Akella [86]. The traffic consists of packet traces for two universities data centers and a private data center. Among these traces we used seven traces of UNIV1 data center traffic for our performance study. The characteristics of each traces used for the study are given in Table 3.3. For studying the performance of routing algorithms, we considered average throughput and response time for the flows for each traffic trace. Here, throughput is measured as the number of requests processed per second and the response time as the time elapsed between the submission of request and to get the response. In order to measure the performance, five trial runs are conducted for each packet trace and an average of all the five trails are presented in the result. We used flow-monitor [87] feature of NS-3 to measure the delay and throughput. Finally, the simulations are conducted to measure the error recovery feature of each of the routing algorithms.

Both the parameters, throughput and delay, depend upon the underlying architecture and the corresponding routing algorithm used by architecture. Here, we consider, designs using three routing algorithms - modified OSPF, source routing and location based routing.

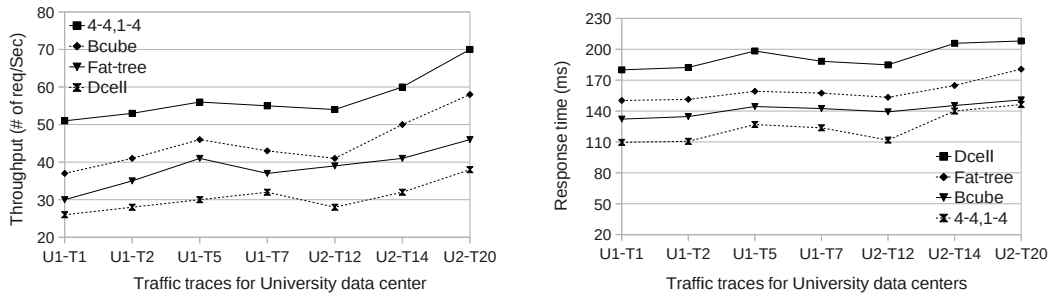
Among the routing algorithms, the performance of the location based routing is better as routing requires only a small table consisting of neighbors addresses

Trace	# of Packets	Average Pkt Size	Average Mbits/Second
UNIV1-PT1	1021724	2961.49	12.85
UNIV1-PT2	1020020	3023.40	12.60
UNIV1-PT7	986087	4925.88	24.11
UNIV1-PT12	1005632	4349.38	22.48
UNIV1-PT14	997536	4513.24	20.95
UNIV1-PT20	985312	5391.47	27.19
UNIV1-PT20	967533	18072.24	117.437

TABLE 3.3: Characteristics of different packet traces for University data center

to find the next hop. Compare to this, the modified OSPF needs to trace the table with a comparatively large number of entries compared to the location based routing. Next, the ability of source routing in using the path diversity is provided through the underlying architecture. Both the DCell and BCube provide $k + 1$ parallel paths between any two pair of nodes. Here, k is the number of levels. But, among these two deigns, performance of BCube is better compared to DCell as all the parallel paths result in the same delay (keeping congestion aside). But, in DCell, among all the parallel paths, there exists a single shortest path and all other paths have variable path length.

The other important factor that impacts throughput is the interconnection pattern. The strong mesh connectivity between the subnets in 4-4, 1-4 provides a greater bandwidth between the subnets and hence the better performance. But, in fat-tree, the switches at the core level are over congested since these switches are shared among all the pods for their interconnection. About BCube and DCell, BCube provides a greater bandwidth due to its complete design but, the partial structure of DCell makes the routing very complicated and often results in selecting non optimal paths. The Fig.3.12 shows the results for the above described experiments.



(a) Throughput measured as the number of requests processed/Second

(b) Response time measured as time elapsed between the submission of request and to get the response

FIGURE 3.12: Performance comparison of different architectures by using the packet traces collected for the University data center

3.3.3 Error recovery

Fault tolerant routing is the property of the routing to continue operating in the event of the network failures. Here, we study the convergence time, the time required by the routing protocols to recover in the event of network failures. For measuring convergence time, we considered each architecture supporting 4096 and we randomly failed switches in the range 1 to 10. Among the routing algorithms, the convergence time for the fat-tree's routing algorithm is high because it is the only routing algorithm that requires re computation of routes. But, the other routing algorithms, DCell, BCube and 4-4, 1-4, do not face this problem as they compute the path instantaneously during data transfer. Although, all the three routing protocols use central server for the recovery, 4-4, 1-4 performs better as the only action it requires is to exempt that link while forwarding the packet to the next hop. Compared to this, the DCell and BCube need to determine whether the failed link is in the any of the paths they compute and if so that path must be exempted and another path must be computed. The convergence time for each of the routing algorithms is given in Fig.3.13.

3.3.4 Performance measure based on growth model

One of the shortcoming in design of 4-4, 1-4 architecture is its vertical growth model. As shown in Fig.3.10, the vertical growth model of DCell, BCube and

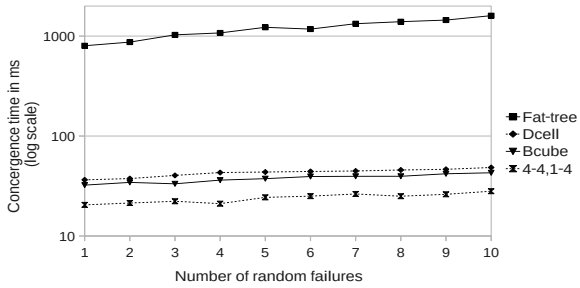
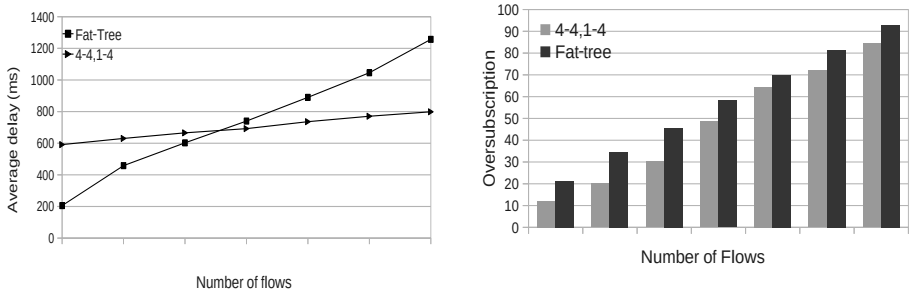


FIGURE 3.13: Time taken for routing algorithm to converge after a network failures. The results are taken by varying the number of elements random fail between 1 to 10

4-4, 1-4 introduces additional path lengths to support more servers. On the other side, fat-tree supports any number of servers with only three levels. Here, we compare architecture features of 4-4, 1-4 and fat-tree with respect to their growth model. For the study we considered MapReduce computing model such as Extract, Transform and Load using MapReduce (ETLMR). We explained more about ETLMR in section 4.2, but for the purpose of discussion we briefly explain the communication model of ETLMR. In ETLMR, the mapper functions extract the values for attributes from unstructured representation such as files and pass them to reducers for processing. The reducer then organizes the values into a structured representation called schemas. Since, mappers and reducers need to accomplish the task in combine, their placements in the system will have significance impact in overall processing. Considering this fact, the performance of both architectures, 4-4, 1-4 and fat-tree is studied. For this, skewed traffic from the nodes executing mapper functions to the nodes executing reducer is created by dividing the number of servers hosting mapper and reducer in the ratio 3 : 1. We have placed all the reducers at one part of the architecture. In 4-4, 1-4, we placed reducers in one of the $level_2$ subnet and the mappers in remaining three $level_2$ subnets. The same kind of placement is followed for fat-tree. We then compared the performance of both the architectures by varying the number of flows between the mappers and reducers. The performance in terms of throughput and delay is given in Fig.3.14. The delay for flows when they are small in number is better in fat-tree compared to 4-4, 1-4. This is because the flows in 4-4, 1-4 need to span additional hops compared to fat-tree. However, 4-4, 1-4 performs better for larger number of flows as



(a) Delay for flows (b) Oversubscription

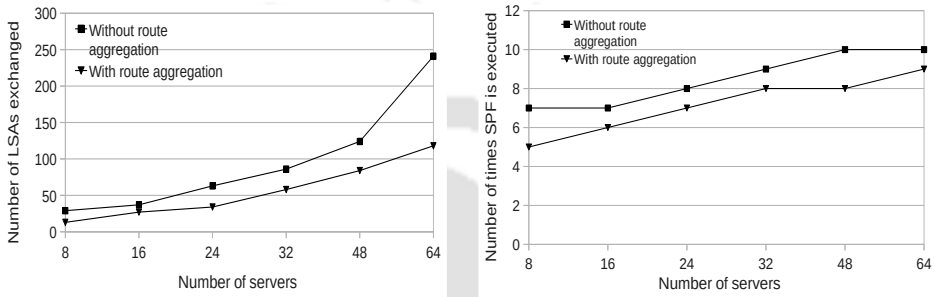
FIGURE 3.14: Performance comparison of 4-4, 1-4 and fat-tree

more number of parallel paths (majority of them are mutual exclusive) between source destination pairs helps to distribute the traffic evenly compared to fat-tree. On the other side, in fat-tree, the same set switches at the core level that connect mappers to reducers pods are faced with the congestion problem and thus incur more delay. For the same reason oversubscription which is measured as ratio of bandwidth used to available bandwidth is better in 4-4, 1-4 compared to fat-tree. To measure the overall utilization, we considered the average oversubscription of all the intermediate switches which participate in data transfer. Since there exists separate paths between each $level_i$ subnet to remaining all $level_i$ subnets in 4-4, 1-4, the utilization is better in 4-4, 1-4 and same can be observed from Fig.3.14 (B).

3.3.5 Control packet overhead

The other advantage of the hierarchical address assignment is the aggregation of routes, which is demonstrated using the emulator Quagga [88, 89]. We configured each virtual node of emulation to run the Quagga software routing suite which provides implementation of OSPFv2, OSPFv3, RIP for UNIX-platforms. Instead of emulating original designs, we used simple tree-like designs because support of Quagga by NS3 at present does not scale for a larger topology of nodes configured with Linux namespace. In one set of simulations, we assigned IP addresses in such a way that route aggregation was not possible at any level. In another set of simulations, we assigned IP addresses such that routes are aggregated at subsequent

higher levels, which is similar to route aggregation in 4-4, 1-4 architecture. We call both the setups 'without aggregation' and 'with aggregation' respectively. Fig.3.15 shows number of link state advertisements (LSA) exchanged and the number of times shortest path first (SPF) executed by an arbitrary router selected at the intermediate level before all the routing tables converge. We get better performance in 'with aggregation' as summarization of routes at successive levels results in advertising lesser routes and hence lesser control overhead.



(a) Number of LSAs exchanged (b) Number of times SPF algorithm executed

FIGURE 3.15: Performance of routing algorithm due to aggregation of routes

3.4 Conclusion

Location based routing is one of the alternative routing protocols proposed to solve the scalability problem of routing protocols for DCNs. In this work, we identified certain limitations with the existing location based routing algorithms. Primarily, large complexities involve in them to determine the location information of end hosts and their inability to cover an entire path of a route. We shown in this work that these limitations are due to their underlying architecture design principles.

In this work, we proposed a new architecture called 4-4, 1-4 based on IP address hierarchy. The correlation between topology designs and address assignment helped us to determine the location of end hosts directly from their addresses. Next, our proposed location based routing used the hierarchical design of 4-4, 1-4 to devise a routing protocol that covers the entire route. On the top of its simplicity, the simulation results confirmed its performance efficiency in terms of throughput and delay compared to the other architectures.

Finally, our proposed architecture holds great promise as an area for future research. Much work remains in optimizing 4-4, 1-4 architecture to provide better oversubscription and to reduce its height. In addition, it would be interesting to look at studying load balancing and energy conservation of 4-4, 1-4. In fact, the later is interesting because unlike fat-tree, 4-4, 1-4 is not a strict three layer design and the hierarchical subnet aggregation may help in traffic localization.

Chapter 4

Greening 4-4,1-4: A distributed greedy approach for finding an energy efficient sub-network

Data Center Networks (DCNs) are the networks that combine millions of servers into a single network fabric. The contributing factors that marked the study of energy for data center are increasing energy cost and over-provisioning nature of data centers. Various studies indicate a huge amount of energy consumed by these data centers [64, 90, 91]. A more recent report from the Natural Resource Defense Council (NRDC) estimated the total energy consumed by all the data centers of U.S. is a massive 91 billion kWh of electricity in 2013 and is expected to increase to 140 billion kWh by 2020, the equivalent of 50 large (500 megawatt) power plants [92]. With the energy becoming a very prominent component in the design of DCNs, there are efforts to propose energy aware data centers [8, 9, 38, 39].

The earlier designs [13, 28, 29, 32, 48] kept their operation simple and annihilated the need for any load balancing or traffic engineering by overprovisioning the network resources. For instance, the designs fat-tree [13], DCell [28], BCube [29] and many others provide sufficient bandwidth through a large number of interconnections to handle traffic that is generated when all their end hosts communicate

with their line speed of interconnection. The aspect of over-provisioning is continued in improved versions of data centers such as PortLand [20], Alias [33], VL2 [32] and Hedera [63].

However, the proclamation from the study of traffic pattern for data centers is that the data centers are not always used to their full capacities and their percentage of utilization varies over time [6, 7, 93]. For example, a data center hosting an important sports event receives higher traffic during the event compared to regular traffic on other times. Similarly, a data center hosting a popular social network may receives greater traffic during weekends compared to weekdays. A detail examination of traffic patterns indicate that utilization of the links that are not part of the core level is very low and utilization of core links at any instance of time is less than 25%. Even the percentage of higher utilization of data center is relatively low compared to the percentage of lower utilization, data centers must be designed to handle unpredictable momentary spikes. Thus, better approach is to make data centers over-provisioned to handle any bursty traffic and devise a method that keeps only the required components *on* as per present/near-future traffic.

There are many components involved in data centers - servers, switches, routers and cooling systems - whose usage can be optimized according to the traffic condition. Since network contributes 20% to 30% of energy usage of DCN, in this work we consider the problem of optimum usage of network in 4-4, 1-4 [48, 49]. Since finding an optimal sub-network to accommodate the current set of flows is NP-hard [8, 39], we propose a distributed greedy algorithm to find an energy efficient sub-network to accommodate the current set of flows. The proposed algorithm determines the sub-network of the entire topology to accommodate the current traffic and then switch *off* the components that are not part of the sub-network to conserve energy. Following are the four major steps of our proposed distributed greedy algorithm, (1) *Finding a sub-network:* To optimize power usage, the proposed algorithm selects a sub-network that can accommodate the current set of flows, (2) *Switching off the unused switches and links:* in this step, the switches and links that are not part of the sub-network are switched *off* to conserve energy,

(3)*Routing the flows:* All the flows are made to select the paths only from the sub-network computed in step (1). For some DCN designs, this may results in selecting a non-optimal path, and 4) *Updating the flow matrix:* When a new flow arrives or an existing flow finishes the sub-network is modified accordingly.

To study the performance of our proposed method, we simulate our proposed algorithm in network-simulator (NS3) and compared with ElasticTree.

This work is organized as follows. In the next section, we give architecture features of 4-4, 1-4 that helps in devising an algorithm to conserve energy. In Section 4.1, we give our proposed distributed greedy algorithm for conserving energy in 4-4, 1-4 architecture. Simulations conducted and the results obtained are given in Section 4.3 and finally, we conclude with Section 4.4.

4.1 Greening 4-4, 1-4 architecture

Before giving our proposed algorithm, we explain the architectural features of 4-4, 1-4 that inspired the proposed algorithm.

4.1.1 Architecture features of 4-4, 1-4 to conserve energy

In this subsection, we describe appropriateness of 4-4, 1-4 for energy conservation. The topology of 4-4, 1-4 [48] is a hierarchical design based on IP address hierarchy. The lower level subnets are combined in the form of a mesh networks that provides strong connectivity among the subnets. Further, design 4-4, 1-4 uses a single address space for both identifying end hosts and routing to overcome the complexities result from use of two different address spaces by earlier designs.

The hierarchical design principle of 4-4, 1-4 allows to create subnets of various sizes. Depending on the application requirement, we can select a best-fit subnet for deploying the application such that intratraffic generated by the application does not interfere with the traffic in the remaining part of the topology. This helps in conserving energy since intratraffic generated by the application is localized. After

completion of the application, the subnet can be switched *off* completely, if the links and switches of the subnet are not used by any other flows for data transfer and all the servers of the subnet are idle. This aspect is illustrated in Fig.4.2.

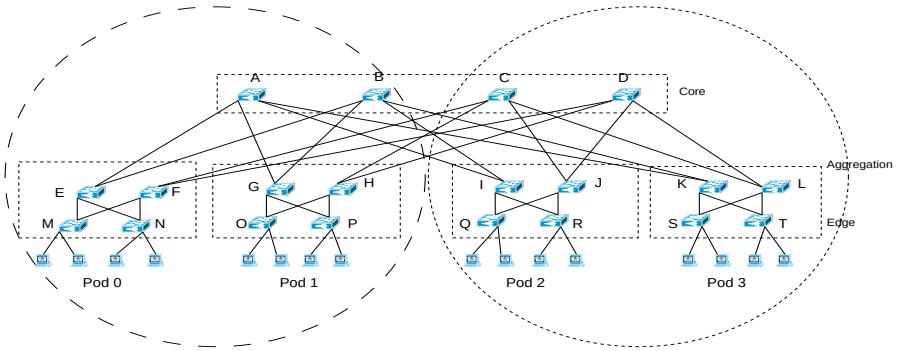
In fat-tree, dividing it into multiple levels is more involved compare to 4-4, 1-4 architecture. A simple way of dividing fat-tree as shown in Fig.4.1(a) results in imbalance. For example, when pod0 and pod1 are grouped into one sub-network through the switches A and B and pod2 and pod3 are grouped into another sub-network through the switches C and D, communication happens only through the paths $\langle E, A, I\rangle$, $\langle G, A, I\rangle$, $\langle E, B, I\rangle$ and $\langle G, B, K\rangle$. But, the switches F and H in first sub-network and J and L in the second sub-network are unused. Instead, connections done according to Fig.4.1(b) result in more bandwidth between the two sub-networks. But, what we shown is an ideal case. Often, fat-tree supports large number of server and dividing them into multiple sub-networks is more involved.

Compare to this, in 4-4, 1-4, the subnets connected through cycles can be easily divided into separate subnets. This is because, in 4-4, 1-4, connecting $level_i$ subnets in the form of cycle provides interconnection between every pair of $level_i$ subnet. The scheme of dividing 4-4, 1-4 architecture into two sub-networks is shown in Fig.4.2.

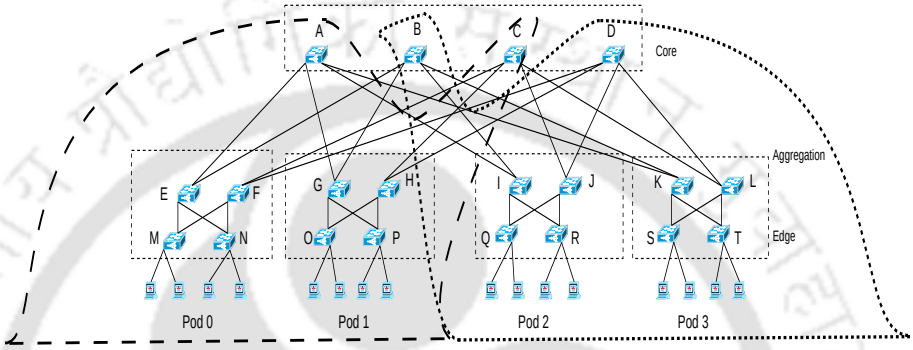
Similar to 4-4, 1-4, design BCube can also be easily divided into different sub-networks. But, routing in the partial BCube is more involved compared to BCube. Further, doing the same for DCell is a much more complex because of its non-linear growth.

The other architecture feature of 4-4, 1-4 is the rich mesh connectivity. At each level, complete connectivity between the subnets through cycles provide ability to handle any type of traffic because the cycle provides multiple paths between servers.

Finally, location based routing for 4-4, 1-4 helps in conserving energy as follows. Since location based routing of 4-4, 1-4 is a packet switched network, each packet at an intermediate switch is forwarded to next hop by comparing the destination address and its address. The presence of multiple paths between the switches at successive levels enable a switch to distribute the traffic evenly among



(a) A simple scheme of dividing fat-tree into two sub-networks. The subnets are shown by marking their boundaries with the dotted lines



(b) A complex scheme of dividing fat-tree into two sub-networks. This schemes helps in utilizing all the links connecting two sub-networks

FIGURE 4.1: In the figure, marking topology for the division into subnets is shown with dotted lines.

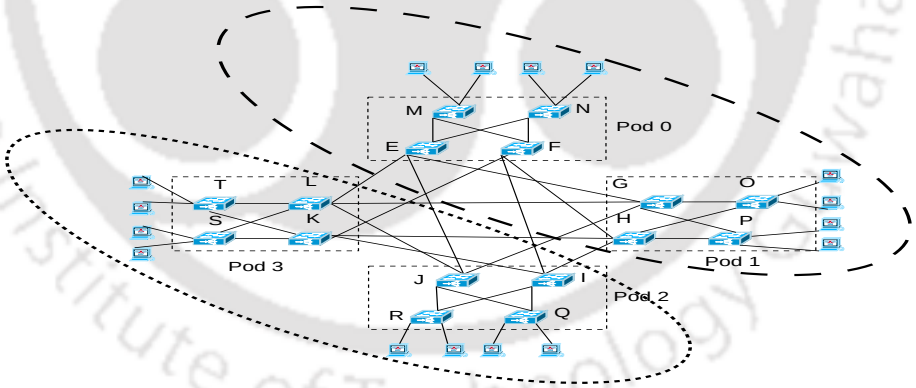


FIGURE 4.2: 4-4, 1-4 is divided into two subnets. The subnets are shown with marking their boundaries with the dotted lines

the selected links using equal cost multipath [ECMP] routing. Therefore, the routing algorithm can forward packets through any of the available links without any additional overhead.

4.1.2 Distributed greedy algorithm for finding energy efficient sub-network

Our proposed algorithm takes input **(G, FM, THS)**, where G is the graph representation of 4-4, 1-4 topology, FM is a 0-1 flow matrix and THS is threshold limit value, and returns a sub-network which can accommodate all the flows of FM while guaranteeing THS value. The THS limit value represents the percentage of additional resources needed in case of network components failure. We use distributed greedy algorithm for finding energy efficient sub-network. The four steps of our proposed method are given below.

1. Finding sub-network: For each flow f_i in FM, we find a path E_i while maintaining the threshold THS. Then, we construct a sub-network $G_0 = \bigcup_{i=1}^{n} E_i$.

2. Switching off unused network resources: In this phase the switches and links of the original network which are not part of the sub-network are switched *off*.

3. Routing the flows: The paths found for each flow are shortest in terms of the number of hops, since each switch at the i^{th} level finds a next hop switch either in the $(i+1)^{th}$ or $(i-1)^{th}$ level depending on the subnet of destination. The location based routing of 4-4, 1-4 optimizes link usage by distribuing using ECMP.

4. Flow matrix update: Whenever a new flow is added to the flow matrix, we find additional resources required to accommodate the flow and switch them on. The location based routing automatically takes care of routing the flows through the updated sub-network. For each finished flow, the links assigned for the flow are marked for deletion. However, before switching *off* these links it is ensured that these links are not assigned to any other flows.

The study for power consumption for switches of data centers shown that the power saved by switching *off* the entire switch is much higher than switching *off* its all the ports. The increase in power consumption by switch is about only 8% going from no traffic to traffic that uses the entire capacity of a switch. Compare to this, the power saved by turning *off* an unused port is just 1-2 Watts [6, 7]. Therefore, if all the ports of any switch are found to be switched *off* then we switch *off* the entire switch. The pseudocode for the algorithm is given in Algorithm 4.1.

Sub-network(G,FM,THS)

Input :

G: A graph representation of 4-4, 1-4 topology

FM: 0-1 flow matrix

THS: Threshold limit value

BEGIN

1) $G_0 = \emptyset$

for i = 1 to n **do**

2) E_i = Path for the i^{th} flow

3) $G_0 = G_0 \cup E_i$

end for

4) Switch *off* the network components which are not part of G_0

5) Use location based routing for routing the flows through G_0

while TRUE **do**

if a flow arrives or a flow finishes **then**

if F_j is a new flow **then**

6) Update G_0 to include F_j

7) Switch*on* additional resources needed for F_j

end if

if F_j is a finished flow **then**

8) Update G_0 to exclude F_j

9) Switch *off* resources of F_j if they are idle

end if

end if

if all the ports of a switch are idle **then**

10) Switch *off* the entire switch

end if

11) sleep(2 sec)

end while

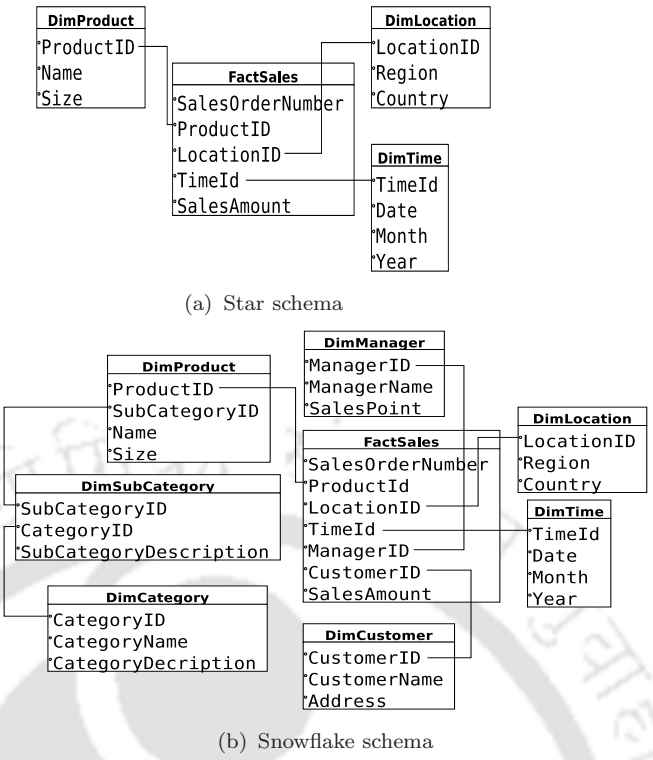
Algorithm 4.1: **Greedy algorithm to find energy efficient sub-network**

4.2 ETLMR: A case study

Applications hosted on data center such as Sort, Word Count, Hive Join, Hive Aggregation, ETLMR and many others use MapReduce programming model to efficiently solve the problem. Xie, et al. [94] profiled the network traffic patterns of these applications and observed many traffic patterns with varying traffic loads. In this section, we consider a data warehouse (DW) application extract, transform and load using MapReduce (ETLMR) to study the efficiency of our proposed greedy algorithm.

A typical data warehouse (DW) consists of facts like sales details, order processing and performance summary of employees and dimensions such as product, location and time. Further, in data warehouse the facts and dimensions are represented as schemas. A schema is a logical description of either a fact or dimension. There are two popular schemas, star schema and snowflake schema. An example of star schema representing a fact *FactSales* is given in Fig.4.3(a). This schema is used to describe the fact table *FactSales* that is connected to three dimension tables through foreign keys. Further, a fact table may have its own attributes such as *SalesAmount*. Similarly, a snowflake scheme representation of fact table *FactSales* is shown in Fig.4.3(b). In a snowflake schema, certain dimensions such *DimProduct* shown in figure is normalized by splitting the attribute into multiple attributes.

Because data in a DW is stored in the distributed file system (DFS), an application extract-transform-load (ETL) extracts necessary data from DFS and builds the data representation in the form of schemas. Since ETL processes gigabytes of data, it uses MapReduce [4] programming model where mappers do the projections and reducer does necessary transformation to represent data in DW schemas [95, 96]. A schematic depiction of working of ETLMR is shown in Fig.4.4.



(a) Star schema

(b) Snowflake schema

FIGURE 4.3: Examples of star schema and snowflake schema in data warehouse

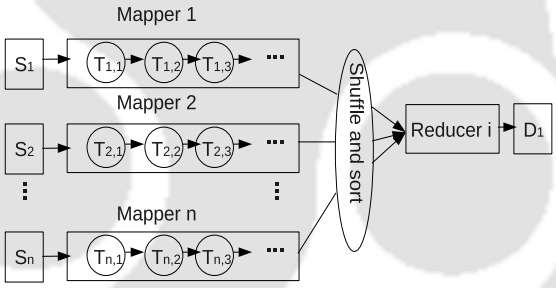


FIGURE 4.4: Schematic depiction of dimension processing using MapReduce

4.2.1 Dimension processing

Processing each dimension is considered as data intensive job where data stored in DFS is processed from the mapper and reducer transform these values into dimensions. Since processing of each dimension is treated as a separate entity, it is advantageous to deploy processing of each dimension on a separate subnet. As discussed in the previous section, the hierarchical subnet feature of 4-4, 1-4 helps to deploy the dimension processing on a separate subnet. This helps in two ways. One, processing of one dimension does not interfere with the processing of other.

Second, after processing a particular dimension, a subnet hosting that dimension can be completely switched off. As mentioned earlier, the hierarchical nature of 4-4, 1-4 helps in assigning best-fit subnet to individual dimension processing.

4.2.2 Fact processing

In processing fact table, many dimension tables are referred through foreign keys of fact table. This contributes traffic between subnets hosting dimensions and subnet hosting fact table. This skewed traffic that causes congestion at the subnet hosting fact table. This is a well known problem in TCP called incast congestion [70, 71, 97]. Incast refers to a many-to-one traffic pattern typically found in distributed processing such as MapReduce, Hadoop, and big data application networks. In an incast scenario, a parent node places a barrier synchronized request for data to a cluster of nodes. The cluster of nodes, in response, simultaneously responds to this request, resulting in a burst of traffic from many nodes to the parent node. In 4-4, 1-4, complete connectivity among the switches of a cycle helps in distributing the traffic evenly among the links connecting many subnets hosting dimensions and subnet hosting fact processing. Since each subnet hosting a dimension can use separate links to communicate with subnet hosting fact processing, the effect of incast congestion is greatly reduced.

4.3 Performance analysis

Simulation is done using network-simulator (NS3). The performance of the proposed greedy algorithm on 4-4, 1-4 supporting 4096 servers and ElasticTree method on fat-tree [8] supporting 4394 servers (k=26) are compared. The design 4-4, 1-4 is a two level design, where four $level_0$ subnets each consisting of 1024 servers are grouped to form $level_1$ subnet. For the study, traffic similar to fact processing of ETLMR (shown in Fig.4.3(b)) with one fact table and five dimension tables are considered. A highly skewed traffic is generated by hosting dimension processing

on 75% of the servers and fact processing on 25% of the servers. Even processing of certain dimensions such *DimProduct* represents an instance of skewed traffic. But, the amount of traffic generated for the fact (FactSales) processing is much higher than the amount of traffic generated for dimension (DimProduct) processing because *FactSales* interact with five different dimension tables and *DimProduct* interacts with only two other dimension tables. In a highly skewed traffic scenario, a large amount of traffic goes from the subnets hosting dimensions to the subnet hosting the fact processing creates incast scenario at the subnets hosting fact processing. Since skewed traffic represents the extreme condition, the measurement of various network parameters are done at two places: the subnet hosting the *DimProduct* and *FactSales*. The entire execution sequence represents two step execution of ETLMR where processing of the FactSales begins only after the processing of *DimProduct* is completed. Similarly, a lesser skewed traffic is generated by hosting dimension and fact processing on a equal number of servers and the performance of the proposed method is studied. Finally, to measure energy usage, the power consumed by servers, switches and links are set as 250 Watt, 97 Watts and 0.25 Watts respectively [8, 93].

4.3.1 Energy conservation

The performance of both the methods in terms of energy consumption for different number of flows, with flow sizes of 500 MB, is measured. This simulation is repeated by varying the ratio of servers allocated to fact processing and dimension processing, $1:3$, $2:3$ and $1:1$. The energy consumed is given in Fig.4.5. In these simulations, power conserved by our method is much better than ElasticTree. This is because of the rich connectivity in 4-4, 1-4. The study shows, both the methods are able to handle the traffic gracefully when the percentage of servers allocated for the dimension table is higher. However, when 25% of servers are allocated to dimension processing, 5% of the servers to each dimension and 75% of servers allocated to a single fact table results in heavy traffic on the links connecting subnets

hosting dimension tables with a fact table. Due to this, both the method suffered heavy packet drops and could not complete flows and hence dividing the servers in the ratio of 3 : 1 is exempted from performance study.

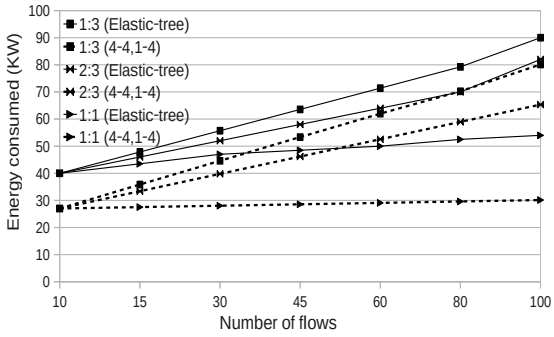


FIGURE 4.5: Total energy consumed by 4-4, 1-4 and ElasticTree to achieve 100% packet delivery

The amount of energy consumed at different time intervals is shown in Fig.4.6. During the first five minutes of the simulations, since there is only a dimension processing, the amount resources used is minimal. However, once the fact processing starts, the amount of energy used is considerably increased.

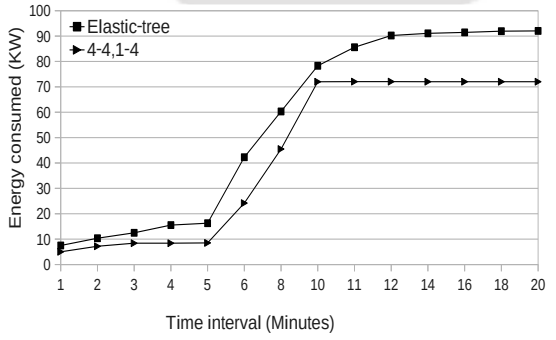


FIGURE 4.6: Energy consumed by 4-4, 1-4 and ElasticTree at different time intervals indicates two step execution of ETLMR

Finally, the performance of both the methods by varying flow sizes for 100 flows is given in Fig.4.7. The amount of network resources required for both the methods is increasing as flow sizes are increased. However, the amount of network resources required by 4-4, 1-4 are lesser compared to ElasticTree. Fat-tree connects i^{th}, $i \in [0, \frac{k}{2}]$, aggregation switch from each pod to i^{th}, $i \in [0, \frac{k}{2}]$, core switch in each set. Selecting a core switch from different pods separately may lead to congestion at the link connecting the destination pod. Thus, to get away with

this problem, the core switches are selected randomly and that results in selecting more switches and hence more energy.

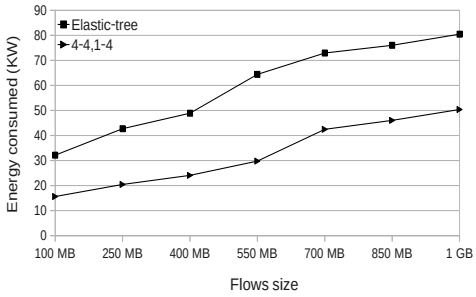
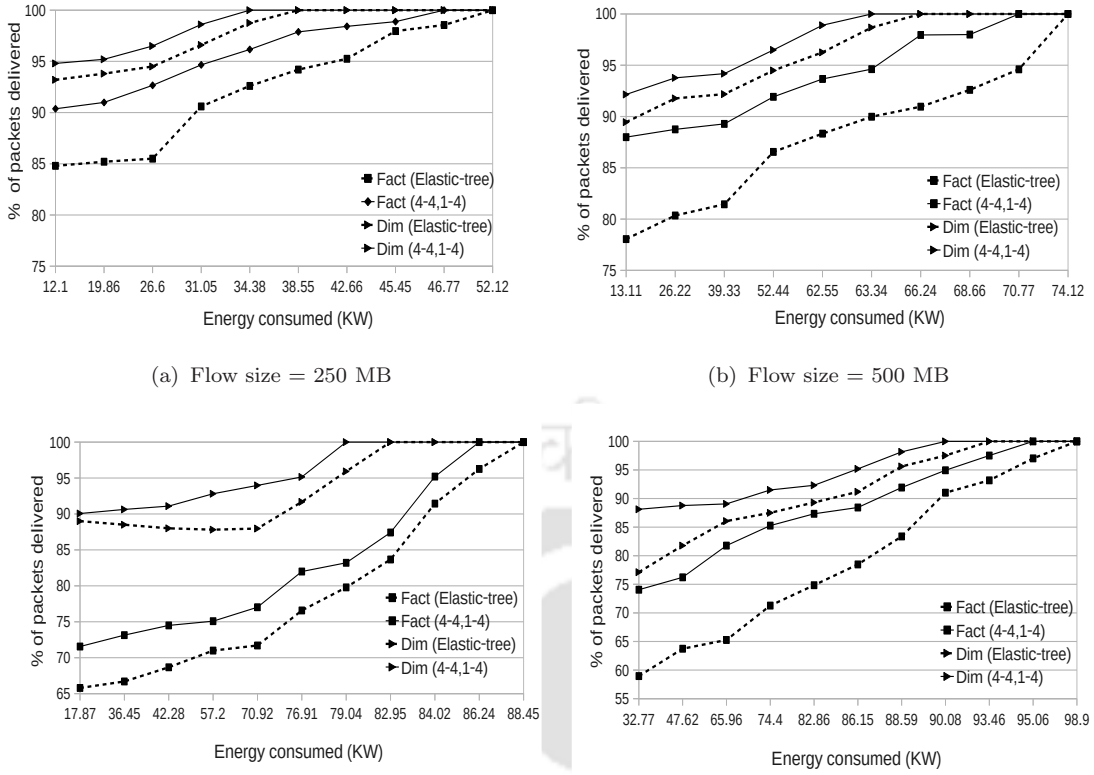


FIGURE 4.7: Energy consumed for 100 flows

4.3.2 Packet delivery ratio

The performance of both the methods in terms of energy consumed by the subnets hosting *DimProduct* and *FactSales* against the packet delivery ratio for 100 flows is studied. The energy consumed for various traffic loads is given in Fig.4.8. In both the methods, as the number of switches selected increased, the percentage of packets dropped is reduced accordingly. Simulation results show that the greedy heuristic method achieves higher level of energy conservation in 4-4, 1-4 compared to ElasticTree. Design 4-4, 1-4 provides 15% more energy conservation compared to ElasticTree when the flows sizes are 1 GB. This is because of the complete connectivity between the subnets, the design 4-4, 1-4 is able to check more packet drops compared to ElasticTree.

Oversubscription: The oversubscription of a subnet is defined as the ratio of the worst-case achievable aggregate bandwidth among the end hosts to the total bisection bandwidth of the subnet. The oversubscription of the subnet hosting *FactSales* and subnets hosting *DimProduct* and the overall network is given in Fig.4.9. The oversubscription at the subnet hosting *FactSales* is higher than the subnet hosting *DimProduct* because fact processing simultaneously interacts with five different subnets, each hosting different dimensions, causes high traffic on the links connecting subnets hosting fact table with many dimension tables. While the traffic at the subnet hosting *DimProduct* is less because it interacts with only two



(a) Flow size = 250 MB

(b) Flow size = 500 MB

(c) Flow size = 7500 MB

(d) Flow size = 1 GB

FIGURE 4.8: Percentage of packets delivered as a function of energy consumed, for a flow sizes of 250MB, 500MB, 750MB and 1GB

other dimensions. In all three cases, oversubscription in ElasticTree is higher than 4-4, 1-4. This also leads to higher packet drops in ElasticTree which is consistent with the result of Fig.4.8. Whereas in 4-4, 1-4 a complete connectivity between the switches of cycles helps in evenly distributing the traffic among the links which result in fewer packet drops.

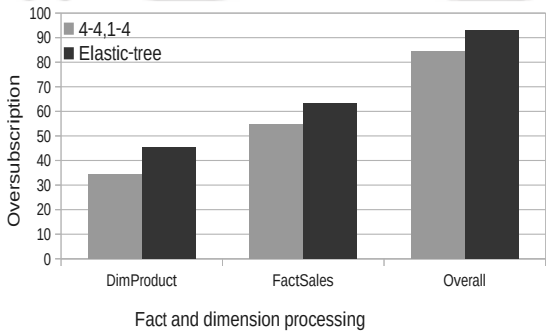


FIGURE 4.9: Oversubscription of subnets hosting dimension processing and fact processing. The results in the last column are the oversubscription for entire sub-network

Latency: The average delay of 100 flows with 100% delivery ratio is measured

and given in Fig.4.10. The average delay in the proposed method is lesser than the ElasticTree which is consistence with the earlier results for the percentage of packet drops due to the oversubscription.

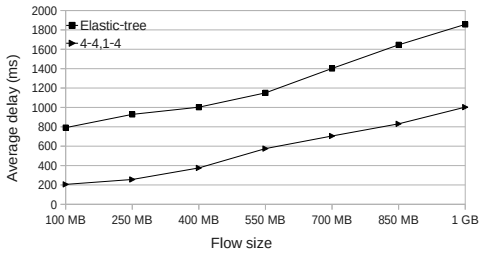


FIGURE 4.10: Average delay of 100 flows

4.3.3 Location based routing vs. source routing

In the source routing, hierarchy and symmetry in both topology construction and address assignment enables the source to compute entire path to the destination. Hence, each source node needs to know the sub-network to find a path for each of its flow through the sub-network. Therefore, the centralized server that finds the energy efficient sub-network, to accommodate the current set of flows, need to inform all the active servers the available network resources through the sub-network.

DCell [28] and BCube [29] are two popular examples for source routing. In partial BCube, in the absense of regular route it is still possible to reach the destination by correcting the hamming distance code arbitrarily. This may result in additional extra delay because of increase in number of hops. For example, for BCube shown in Fig.4.11, the direct path between (11) and (01)⟩ is $((11), \langle 1,1\rangle, (01))$. If the energy saving forces this flow to take the path through the level-1 switch $\langle 1,0\rangle$ instead of $\langle 1,1\rangle$, then the re-routed path is $((11), \langle 0,1\rangle, (10), \langle 1,0\rangle, (00), \langle 0,0\rangle, (01))$. This is an additional overhead compared to location based routing. The location based routing of 4-4, 1-4 does not experience additional delay because the number of hops in the energy efficient sub-network remain same as the original network. This is because of the symmetry in connection between the switches in successive levels which enables any switch at $level_i$ to select any arbitrary switch at the next

level.

To study the increase in delay due to re-routing of flows, we constructed four

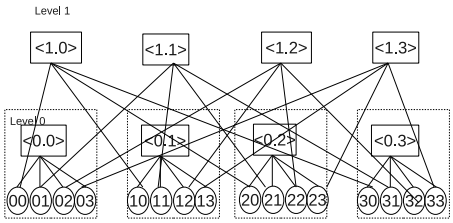


FIGURE 4.11: BCube, architecture that uses source routing

level BCube supporting 4096 servers in NS3. We powered *off* an arbitrary switch in the regular path of a flow and measured the increase in delay due to re-routing of 20 flows between two hop apart servers in complete BCube. Similarly, we measured the delay for four, six and eight hop apart servers in complete BCube by switching of two or three arbitrary intermediate switches in the regular paths of the flow. The simulation results are given in Fig.4.12. The simulation results show 162% to 292% increase in average delay due to re-routing.

Re-routing in partial DCell involves finding a proxy that can re-route the

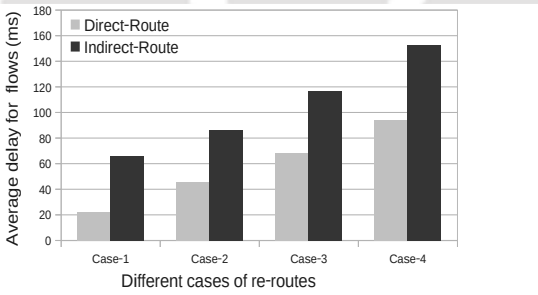


FIGURE 4.12: Increase in delay for a flow because of re-route.

flows through an alternative route. This process is more involved as claimed by the authors of DCell and increases the number of hops between the servers drastically.

4.4 Conclusion

In this work, we proposed a greedy algorithm to conserve energy for 4-4, 1-4 architecture. The proposed method conserves energy by selecting sub-network that can accommodate all the flows of current traffic and powered *off* the remaining network. We used the class of MapReduce programming model called ETLMR to study the performance and compared with ElasticTree. The simulation results shown higher ability of 4-4, 1-4 in conserving energy because of its hierarchical growth model and strong connectivity. In the next chapter, we consider the ability of 4-4, 1-4 architecture for load balancing.

Data is growing exponentially due to the data generated from many sources. Effectively storing, querying, analyzing, understanding, and utilizing these immense data sets presents one of the grand challenges to the research community. For this, the infrastructure to store and computation models to process must also be of this large scale. Accordingly, the computations models must be built adopting parallelization, distribution of computations, task management, and fault tolerance to process the task quickly and effectively. Map-Reduce is one such computation model proposed in recent years that has all these capabilities to process the data in a distributed environment. Along side the computation model, the platform on which the computations hosted are also important. Data center are the best choices in this regard with their high storage and processing capacity.

Thus, many of the data and compute intensive applications such as web search [4, 98], large scale machine learning computation [99, 100], simulation [101] and digital media processing are implemented using Map-Reduce. For data centers, these applications are the best choice to study their performance due to their varying demands. In this regard, we considered Map-Reduce programming model Extract, transform and load using Map-Reduce (ETLMR) to construct schemas in distributed environment of data warehouse and shown how 4-4, 1-4 performs better for the traffic generated by ETLMR as all the above stated requirements for Map-Reduce - parallelization, distribution of computations, task management, and fault tolerance are provided by 4-4, 1-4.

Chapter 5

Packet scheduler for meeting deadlines in 4-4, 1-4 architecture

Today, Cloud is a preferred choice for hosting the applications. The advantages of cloud, such as scalability, simplicity and economic feasibility attracted very large applications to be hosted on it. Some of the example applications hosted on cloud are online reservation system, bidding systems and spot resource markets [102, 103]. These applications are latency bound, wherein they must guarantee a response within a strict latency target called deadline. The latency in this context is expressed as end-user perceived latency and estimated through customer impact studies [104]. This forces the backend servers processing the request to obtain the result in a timely manner and deliver the result to the client within a specified latency target. Hence, while addressing the latency target, the backend server must consider both processing and delivery of the result. The latter component, delivery of the results is a network delay that is measured as elapsed time between the request for data and its delivery. This latency is measured as the sum of transmission delay, propagation delay, processing delay and queuing delay. Thus, after subtracting typical Internet and rendering delays, the backend part is left with a very tight deadline to process the request. This deadline is called all-up service level agreement (SLA) [105–107] and is in the range from ∼230ms to ∼300ms. Thus, orderly execution of the job is crucial.

With the massive demand for the applications hosted on data centers, the amount of data processed by each application is also increasing. For example, the number of users in facebook at the end of fourth quarter 2014 is about 1.32 billions [108]. Thus, facebook and other large scale applications such as web search, advertisement selection and many others need to process terabytes of data to obtain the desired result. These applications in order to process the requests in a timely manner, use partition/aggregate in which overall task is divided into sub-tasks and are assigned to different instances called the worker nodes. The worker nodes in order to speed up the computations, execute their assigned sub-tasks in parallel. The results from these nodes are collected by a node called aggregator node. The aggregator combines the outputs from worker nodes to produce a final result. Since the overall task is divided into sub-tasks, the whole latency target for the application is converted into smaller latency targets for the individual sub-tasks [109]. These smaller latencies are typically in the range from ∼10ms to ∼100ms.

For the smaller latency targets assigned for the sub-tasks, the aggregator expects the worker nodes to obtain the desired result within the specified latency target. Failing to do so, their returned output is exempted by the aggregator node which may result in reducing the quality of the end result returned by the aggregator. For example, an algorithm while selecting flights between two cities, may exempt results from worker nodes which do not fetch the required results within their assigned time limit. By doing so, it may results in exempting some of the cheaper flights operating in that segment.

Although the partition/aggregate appears promising for meeting latency targets, it often results in performance impairments. Simultaneous responses generated by workers create incast congestion at the aggregator and results in TCP timeouts and re-transmissions [71, 72]. The other associated problems are queue build up and buffer occupancy [73, 75]. Infuse of short flows and long flows in DCN traffic may cause queue buildup by the long flow and may results in starvation for the short flows. Further, since the buffer space for the switch is shared by all the ports, an over activity at a certain port may fill up the entire buffer resulting in buffer space nonexistent for other ports.

There are two main classes of solutions proposed for the problems of congestion and meeting deadlines for flows, one at the transport layer and another at the network layer. The proposed solutions at the transport layer modulates the TCP window size according to the extent of congestion and the deadline value associated with flows [42–45]. These solutions are proposed based on the assumptions that all the packets of a flow follow a static path. Based on this assumption, they modulate the congestion window.

In this work, we take a path diversity used by the location based routing in 4-4, 1-4 to overcome the limitations of using a static path. The location based routing, proposed to overcome the scalability problem of conventional routing protocols for DCN, uses location information of the end hosts for the routing. The location based routing proposed in PortLand [20], Alias [33] and 4-4, 1-4 [48] distribute flows traffic among all the available outgoing paths using the Equal Cost Multiple Paths (ECMP). Therefore, in this work, we propose a packet scheduler for 4-4, 1-4 that helps the flows to meet their deadline by arranging the packets at intermediate switches according to their associated deadlines. The packet scheduler prioritizes the packets based on the number of hops the packet has to cover and their associated deadline. Thus, with prioritizing and scheduling the packets at the network layer, flows are not constrained to assign to a static path and the intermediate switches need not maintain any flow state information. Thus, the following are the three main advantages of our proposed method.

Solutions for incast congestion: An efficient and simple way of prioritizing the packets of the flows, the problems of incast congestion, meeting deadline for the flows and the queue buildup are addressed.

Utilization of multiple parallel paths: The proposed method, unlike TCP variants, is not constrained to use static paths. Instead, it helps distributing the flows evenly among the available multiple paths using ECMP.

Eliminate the need for storing flow information: In location based routing [20, 33, 48], no prior computation of routes is required and the intermediate switches do not maintain any flow related information. Similarly, our proposed method overrides the need to maintain any flow related information.

We demonstrate the performance of our proposed packet scheduler using 4-4, 1-4 architecture. This work is organized as follows. In the next section, we explain the background for this work. In Section 5.2, we give our proposed packet scheduler for meeting the deadline. The simulation conducted and the results obtained are given in section 5.3 and finally, we conclude with Section 5.4.

5.1 Background

There are two approaches prevalent in solving the above mentioned problems due to congestion. The proposed modification to TCP behavior to handle the congestion is one approach. In Data Center TCP (DCTCP) [42], the TCP window for the flows are modulated based on the extent of congestion. Thus, whenever a congestion event occurs, the flows instead of decreasing their window size by half, reduce their window size based on the extent of congestion they experience on the path. One of the limitations of the DCTCP is its deadline agnostic nature. During congestion, all the flows backoff equally. The problem with DCTCP is overcome by Deadline Driven Delivery (D^3) control protocol [43] that considers both the congestion and flows deadline for modulating TCP window. However, a major limitation of D^3 is a race condition. As D^3 tries to assign switch buffers in a greedy way, the earlier flow with a later deadline may make the buffer space nonexistent for the flows that arrive later but with early deadline. This problem is overcome by Deadline-aware Data Center TCP (D^2TCP) [44] and Preemptive Distributed Quick flow scheduling (PDQ) [45] where the priorities for a flow are adjusted relative to the priorities of the other flows.

There are two main limitations with the proposed TCP variants. First, the computations for performance evaluation and optimization for the massive traffic of DCN are complex. The study of traffic for the data center shown the data center traffic is very divergent [6, 7]. When the traffic patterns are characterized, there are about 50 representative traffic and a majority of them lasted for less than 100 ms before switching over to other pattern. Hence, by the time the parameters

are adjusted based on present traffic, the traffic may change its pattern and the computations may become stale.

The other limitation is about the static path the flows constraint to use. These methods work on an assumption that the packets of the flows always follow a static path. Based on this assumption, they modulate the congestion window. This constraints the flows to always follow a static path and the intermediate switches to maintain path information about the flows. For example, in DCTCP, the intermediate switches of a path mark the ECN flags for a flow and congestion window for the flow is modified accordingly. Thus, the change in congestion window is effective only when the flow follows the same path.

5.1.1 Insight for proposing packet scheduler

We demonstrate our proposed packet scheduler on 4-4, 1-4 architecture. The architecture features of 4-4, 1-4 that helped in devising our proposed packet scheduler are given below.

5.1.1.1 Oversubscription

Oversubscription of 1:1 is a desired feature for DCN. With this oversubscription, DCN can handle any kind of a traffic that is generated by the DCN. Here, load balancing done by the intermediate switches achieve overall load balancing of the system. For this, the switches need not to know how the traffic has evolved over the time or any future estimation of the traffic. Instead, ability to distribute the incoming traffic among the outgoing links is sufficient to achieve the overall load balancing [32, 84]. But, to perform this localized load balancing, the underlying architecture must provide an oversubscription of 1:1. With 1:1 oversubscription, every switch can select outgoing switches sufficient enough to handle the incoming traffic and distribute the incoming traffic evenly among the outgoing links using ECMP. Since, oversubscription of 4-4, 1-4 architecture is 1:1.1034, by upgrading

the capacity of outgoing links to 1.1034 Mbps we can achieve a oversubscription of 1:1.

5.1.1.2 Location based routing

The proposed methods for handling deadline modulate the congestion window based on the traffic on a path the flow is currently assigned. This constraints the flow to follow a static path. But, location based routing favors distributing the incoming traffic evenly among the outgoing links. Thus, these two contradictory requirements, forces either one of them to relax their requirement. Hence, we propose a packet scheduler at the network that allows the packet to use dynamic paths.

5.1.2 Direction of the packet

In our proposed scheduler, the number of hops a packet has to travel to reach the destination is used as the primary priority. For this, it is necessary to know the direction in which the packet is moving. It can be easily done in 4-4, 1-4 by comparing the address of the intermediate switch with the destination address.

5.2 Packet Scheduler

A packet-switched and integrated service environment are two prevalent methods for meeting the different service requirements. In integrated service environment, a path is pinned down for a flow by reserving the resources along the path for that flows. While in packet scheduling, the packets are prioritized based on the service requirements and the intermediate switches schedule these packets based on their associated priorities. In a large system such as DCN, scheduling the packet based on simple scheduling algorithm may not be feasible for meeting the service agreements for different flows. Rather, scheduling algorithm with a proper

statistical multiplexing will provide an efficient solution in meeting the service level agreements of the flows. In this section, we propose one such statistical multiplexing of the packets at the intermediate switches for meeting the deadlines for the flows.

5.2.1 Packet scheduler for meeting the deadline

We implemented our proposed packet scheduler for the intermediate switches of 4-4, 1-4 architecture. For this, the source computes deadline for individual packets based on the deadline value associated with the corresponding flow. Since, the flow is defined as a logic unit representing a sequence of packets, every packet of a flow uses the same deadline value of the flow. However, their associated deadlines may vary a little when their time of arrival is considered. Here, we make an assumption that the information about the flow deadline is made available for the source at the beginning. The intermediate switches then rearranges the packets based on their associated deadline and schedule the packets by picking up the packets in increasing order of their deadline information. For this, the switch maintains a set of virtual output queues (VOQs) associated with each of the input port. There are many scheduling algorithms defined for packet scheduling such as first come first serve (FCFS), round robin (RR), strict policy and earliest deadline first (EDF). Among these scheduling algorithms, we used EDF as it is the best scheduling algorithm for real time applications.

5.2.2 System architecture

The system architecture for our proposed packet scheduler is given in Fig.5.1. We describe our proposed method by illustrating the working of each of its components.

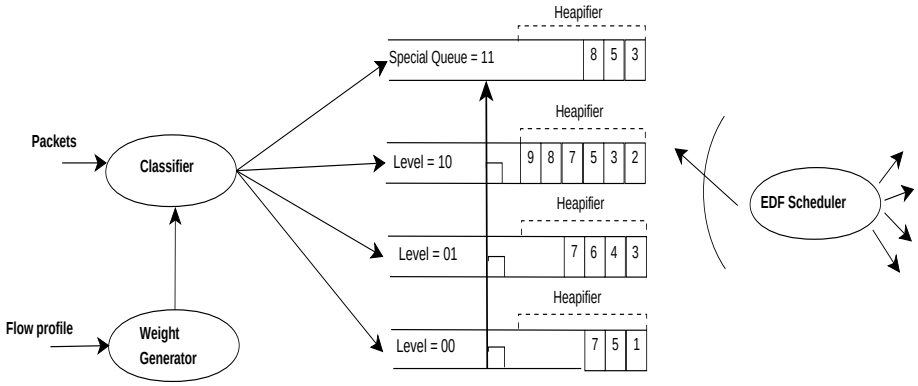


FIGURE 5.1: The system architecture for our proposed packet scheduler

5.2.2.1 Flow profile

Every flow i that is admitted to the system is associated with a latency target, D_i. The latency target for the flows is determined based on two priorities, the level information and the flow deadlines. The level information is used as a primary priority. The packets which need to cover more hops are given a highest priority. Further, the deadline information associated with the flows serve as secondary priority. Here, we assume these information about the flows are available during the flow initiation.

5.2.2.2 Packet classifier

There are two levels of priority used, primary priority and secondary priority. The number of hops a packet still needs to travel to reach the destination is used as primary priority. The intermediate switches to classify the packets and to place them in separate queues use the primary deadline. The secondary priority, the remaining deadline for the packet, is used to arrange the packets within individual queues in non-decreasing order of their weight.

We assume time synchronization of all the nodes in order to prioritize the packets. Since the worker is constrained to finish the assigned subtask within a specified deadline, it in turn imposes a deadline for each packet of the flow based on the estimated flow size and computation time. Hence, an average delay for a packet is estimated as a (Flow deadline)/ (Flow size), where flow size is

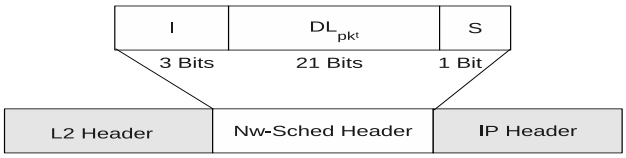


FIGURE 5.2: Additional header for prioritizing packets based on deadline

expressed as number of packets. Thus, the deadline for individual packets of a flow is calculated as

$$\mathrm{DL_{pkt}} = \left(\frac{\mathrm{pos_{pkt}}}{\mathrm{size_{fl}}}\right) \mathrm{DL_{flow}} + \mathrm{T_0}$$

where, $\mathrm{DL_{pkt}}$ is the deadline for packet specified as an absolute time within which the packet must meet the destination, $\mathrm{pos_{pkt}}$ is the relative position of the packet in the flow, $\mathrm{size_{fl}}$ is the flow size expressed as the number of packets in a flow, $\mathrm{DL_{flow}}$ is the deadline for the entire flow and $\mathrm{T_0}$ is the current time.

In order to give priorities to the packets which need to travel more number of hops, packets are prioritized based on l. Here, l is the number of hops a packet has to travel in a forward direction, half the number of hops between source and destination. While measuring l, a single direction is considered to reduce the number of level queues at the intermediate switches.

Finally, a special flag is used to give a highest priorities for the re-transmitted packets. The format of the special header consisting of this information is shown in Fig.5.2.

5.2.2.3 Scheduler

The classifier at the intermediate switches organizes the packets into separate input queues based on the primary priority, the level information. There are two reasons for using multiple input queues [110, 111]. First, it is observed that the bandwidth of the links used for interconnection is increasing in recent years. However, the size of packets used in applications deployed on DCNs (control messages) has not changed much. The second reason is head of line (HOL) blocking. If the destination output port for the packet at the head of the queue is busy, then the packets behind it in the queue are subjected to HOL blocking delays.

Each switch at $\mathrm{level_i}$ maintains $m - i + 2$ queues. All the packets going upto

$i+j$ levels are placed in Q_j, where $i < j <= m-1$. Here m is the total number of levels in 4-4, 1-4 architecture. The queues $Q_{m\text{-}i+1}$ and $Q_{m\text{-}i+2}$ are used for storing packets moving downwards and the packets with special flag set respectively. We use the special flag for the retransmitted packets. Among $m-i$ queues, queue Q_j, $0 < j < m-i$ stores packets going upto $(i+j)^{th}$ level in the forward direction. For example, for $m = 4$, a switch at $level_1$ contains total four queues. Among four queues, queues Q_0 and Q_1 are used to store packets going upto $level_2$ and $level_3$ respectively, Q_2 stores the packets going downward and Q_3 stores packets with special flag set.

After the packets are classified and placed into a separate queue, the packets within a queue are arranged in non-decreasing order of their secondary priority. This will ensure the earlier deadline packets are placed ahead of the later deadline packets in the queue. While classifying the packets, two problems need to address.

Since intermediate switches receive packets going in both directions, classifying the packets based on the number of hops still a packet has to traverse is difficult. For example, for a packet that needs to travel n hops when received at the intermediate switch in level i, the number of hops still remaining is either n-i or n-m-i depending on a direction in which the packet is traveling. If the packet is going towards the core then it is n-i and if it is from the core switches towards end hosts then it is n-m-i. Since topology design of 4-4, 1-4 follows IP supernetting, the direction in which the packet must be transmitted can be determined by comparing intermediate switch address with the destination. Let, $x_n x_{n-1}...x_0$ and $y_n y_{n-1}...y_0$ are destination switch address and the intermediate switch address at $level_i$ respectively. When the packet is moving in the forward direction, the suffix $x_i x_{i-1}...x_0$ matches $y_i y_{i-1}...y_0$ and number of hops a packet still need to travel in a forward direction is determined from finding hamming distance between $x_n x_{n-1}...x_{i+1}$ and $y_n y_{n-1}...y_{i+1}$. Similarly, when the packet is moving downward, the prefix $x_n x_{n-1}...x_{i+1}$ matches $y_n y_{n-1}...y_{i+1}$ and number of hops a packet still need to travel to reach end hosts is determined from finding hamming distance between $x_i x_{i-1}...x_0$ and $y_i y_{i-1}...y_0$.

The next challenge is to calculate the remaining deadline for the packet after

reaching a switch at $level_i$. Here, again the remaining deadline for the packets is determined based on direction the packet is moving. Thus, the remaining deadline for the packet is calculated as,

$$\mathrm{DL_{rem}} = \frac{DL_{\mathrm{pkt}} - T}{h_{\mathrm{rem}}}$$

Here $\mathrm{h_{rem}}$ is the remaining hops a packet has to travel to reach the destination and T is the current time. Again, the remaining hops $\mathrm{h_{rem}}$ is 2m-i when the packet moving forward and is i, the current level when the packet is moving towards end hosts. After the packets are classified and placed into separate queues, the individual queues are sorted according to non-decreasing order of the remaining deadlines for the packets. This entire classifying and sorting the packets can make use of external sorting [110, 111] wherein different queues are implemented as virtual output queues (VOQs).

After packets are re-arranged in multiple level queues, scheduler picks up the packet from each level queue in EDF fashion. However, before scheduling level queues, the packets in special queues are given highest and are scheduled ahead of the level queues. The level queues are scheduled in EDF fashion since it is the best possible solution to overcome the head of blocking. The scheduling of multiple level queues in a switch at $\mathrm{level_0}$ is given in Fig.5.1.

5.3 Simulations

The simulation is conducted using Network-Simulator (NS-3). For the experiment, we used 4-4, 1-4 architecture implemented upto $level_2$ supporting 16384 hosts. Since the design is of two levels, the switches at $level_0$, $level_1$ and $level_2$ contains four, three and two queues respectively. These queues and schedulers are implemented as click elements of NS3 [83, 85] in which three new elements, classifier, heapifier and scheduler are implemented as separate elements and are integrated with the click graph. For the simulations, we use 10 Mbps interfaces and a round-trip propagation delay of 10 ms. To study the performance of our proposed method, different flows with different flow sizes 250KB, 500KB, 750KB,

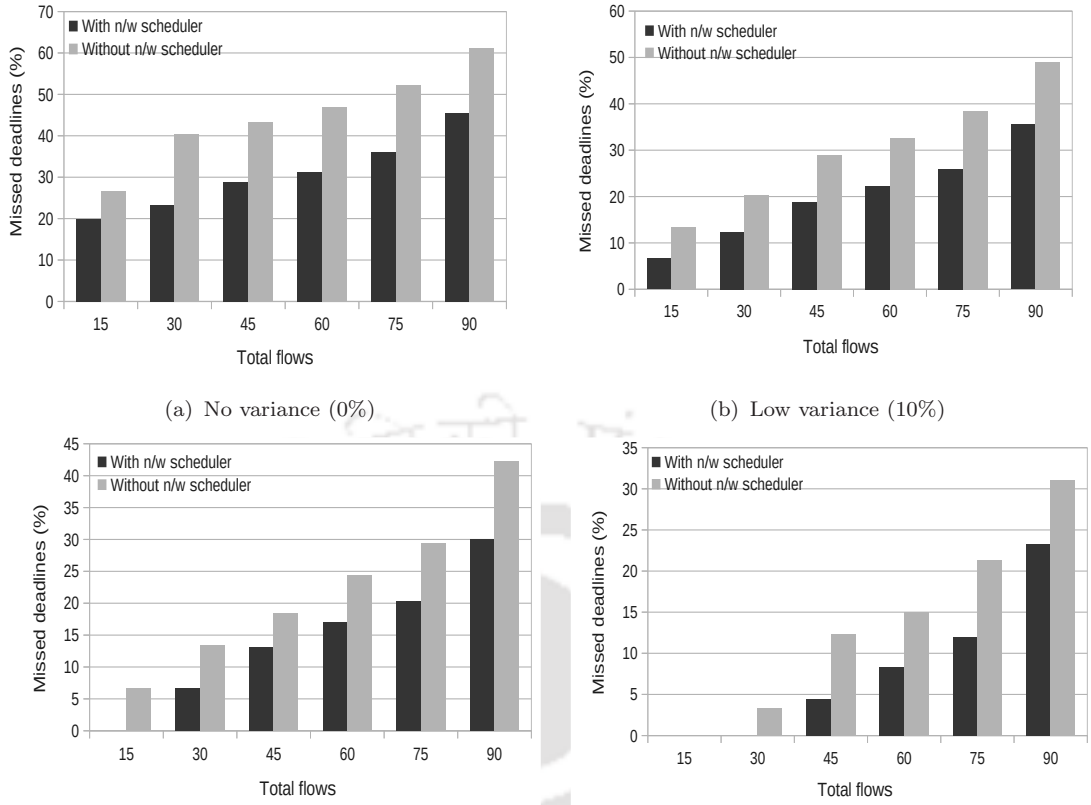
1GB and 2GB and with corresponding deadlines 100ms, 400ms, 800ms, 1200ms and 2000ms are used.

5.3.1 Performance evaluation

The performance of the packet scheduler for handling deadline is measured by varying the deadline variance and queue size. Partition/ aggregate programming model is prone to incast congestion when many workers simultaneously respond to the aggregator causing congestion at the subnet hosting aggregator. For the simulation, we considered a similar skewed traffic where traffic is generated from three $level_2$ subnet to a single $level_2$ subnet. We measured the number of flows that missed their deadlines for total flows - 15, 30, 45, 60, 75 and 90.

5.3.1.1 Impact of deadline variance

We marked the deadline values for flows by adding some variation to those values that defines the final deadline for the flows. We considered four cases based on the extent of variation we added to the flow deadlines. In the case of zero variance, we measured the number of flows strictly meet their specified deadline. Next, for the low variance of 10%, we added 10% uniform random variation to the base deadline and verified ratio of flows that missed their deadline. Similarly, for medium and high variance we considered 50% and 100% variance to the deadline. The percentage of flows that missed their flow deadline with different variance is given in Fig.5.3. The results shown are with and without our proposed packet scheduler. The results confirmed our proposed method's ability to meet the deadline for the majority of the flows. Our proposed method enable an average of 8.07% more flows to meet their deadline with no variance and enable an average of 5.83% flows to meet their deadline with 100% variance.



(a) No variance (0%)

(b) Low variance (10%)

(c) Medium variance (50%)

(d) High variance (100%)

FIGURE 5.3: Percentage of flows meet the deadline as a function of total number of flows, for different deadline variances.

5.3.1.2 Impact of queue size

Finally, simulations are conducted by varying queue sizes. For this, we measured the percentage of flows that missed their deadlines by varying maximum number of packets a queue can hold at any instance of time. The different queue sizes we considered are the queues with 10, 20, 30, 40 and 50 packets. The performance in terms of the number of flows that met their deadlines for two different total number of flows is given in Fig.5.4. From the graph, we can conclude that as the queue sizes increases the number of flows that meet their deadline is also increased. However, with the packet scheduler, there are more flows meet their deadlines.

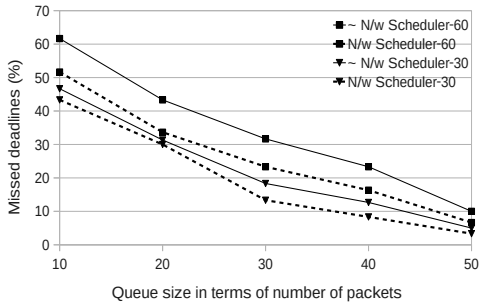


FIGURE 5.4: Missed deadline for varying queue size.

5.4 Conclusion

Earlier proposals for meeting the deadlines for the flows are constrained the flows to follow a static path. In this work, we proposed a packet scheduler to classify the packets based on the number of levels they need to travel and the remaining deadline for the packets. By ensuring individual packets of the flow meet their deadline, we will ensure the overall specified deadline for the flows is achieved. We demonstrated ability of our proposed packet scheduler in meeting for the flows in 4-4, 1-4 architecture. We used 4-4, 1-4 for our study as it has all the features required for our proposed algorithm. We studied the performance of our proposed method by measuring additional flows which are able to meet their deadline due to our proposed method. The simulation results have shown our proposed method increases the number of flows that meet their deadline by 5.83% to 8.07%. As an extension of this work, we would to like to generalize our proposed method to work on any given DCN architectures such as fat-tree, BCube, DCell and others. The major challenge in these designs is to determine the direction the packet is traversing.

Chapter 6

Modular Data Centers Simplified using IP Address Hierarchy

Source routing is an alternative routing protocol proposed for DCN to solve the scalability problem of conventional routing protocols. In source routing, the source computes a path to the destination using topology characteristics of the underlying architectures and uses the computed path for data transfer. Few example designs for source routing are DCell [28, 55], FiConn [59], BCube [29], MCube [30] and and MDCube [31]. These designs define their topology using baseline topology and growth model. Baseline topology, often referred as $level_0$ subnet defines the building block of the topology and growth model defines construction of $level_i$ subnet from a set of $level_{i-1}$ subnets. The baseline topology and growth model varies between the architecture. For example, the growth model of BCube is derived from the properties of a hypercube and DCell from the properties of complete connected graphs.

In these designs, addresses assigned to the end hosts determine their relative location in the topology. In BCube, the address assigned to the end host is a vector of indices where i^{th} index corresponds to its relative ordering in i^{th} dimension. Similarly, an i^{th} index in a DCell corresponds to its relative ordering in the i^{th} level of a complete connected graph. This way of addressing enable a source to determine path to the destination. In BCube, routing algorithm used in the

hypercube, correcting hamming distance between source and destination address, is used for computing path to the destination. Similarly, in DCell, interconnection logic used in forming a complete connected graph is used for finding the route. Thus, to record the topology information in an address, a vector of indices is used for addressing the end hosts.

Since the majority of the network components used in topology designs are TCP/IP compatible, the special addresses are mapped to the IP addresses during data transfer. In this work, we identify the limitations of using two address spaces in the source routing and propose a simple encoding scheme that evade use of two address spaces.

The following are limitations of using two address spaces. For an easy narration, we describe the limitations of DCell in particular but the limitations equally apply for other designs.

Mapping between address spaces: DCell identifies its end hosts using DCell addresses and the source computes a path to destination using these addresses. Then, during data transfer, these addresses are mapped to IP addresses since routing protocols need IP addresses of the end hosts. This requires maintaining the mapping between two address spaces. Building and maintaining mapping between DCell addresses to IP addresses is both complex and time consuming and there is no procedure available to automate this. The mapping need to be done manually.

The mapping done either locally or globally introduces an extra overhead. If the mapping is local then every server has to maintain the mapping and this results in extra space proportional to double the number of servers supported by each architecture. It is double the servers because each server has to maintain mapping for all the remaining servers. Instead, if the mapping is maintained in a central server then additional one data transfer latency is an extra overhead to fetch the mapping.

Interoperability: Most of the applications deployed on data centers use TCP/IP. In order to insulate them from newly introduced addresses, DCell and most of other designs implement their proposed routing in layer-2.5 of TCP/IP stack.

The special addresses are encoded in their proposed special headers and are encapsulate between IP header and Ethernet header. This results in many additional complexities. First, encapsulation consumes additional bandwidth overhead [112]. The bandwidth overhead for different encapsulation for different IP payloads is given in Table 6.1. Since DCell headers are of 20/24 bytes (similar in other MSC deigns), the traffic of DCell results in an extra bandwidth overhead of around 5%.

Traffic	Encapsulated header size	Additional bandwidth overhead (%) due to encapsulation
IP-in-IP	20 bytes	4.36
L2TPv3/IPv4	24 bytes	5.49
GRE/IPv4	36 bytes	8.86
MPLSoL2TPv3	36 bytes	8.86

TABLE 6.1: Overhead of different encapsulation schemes.

In this work, we propose a simple mechanism to overcome these limitations by redesigning the Modular Server Centric (MSC) designs to use a single address space for both identifying the end hosts and routing. Here too, the same property, the correlation between topology design and address assignment, that we used to design 4-4, 1-4 help to redesign MSC designs.

Since in these designs, the recursive scalability, the ability to build a larger subnet from the smaller subnets features supernetting feature of IP address hierarchy, we use this association to encode special addresses inside the IP addresses. By doing so, unnecessary mapping between two address spaces is avoided. We demonstrate the advantages of using a single address space for the designs DCell and BCube, the two popular designs for source routing. We also made a small modification to the original baseline topologies that reduced the number of interconnections in the topology design. The proposed modification also changed the original topologies into true networking topologies. Lastly, as our proposed designs use TCP/IP topology, we eliminate the need for any separate control and management planes.

Our proposed addressing mechanism also works without our proposed modification. However, we favor the change we made to topology as it brings additional

advantages such as reduced number of interconnections without effecting the design principles. Further, the regular growth model of BCube helped us to propose a location based routing similar to the one we proposed in third chapter for 4-4, 1-4. Thus, use of single address space along with the proposed modification to the baseline topology provided the following advantages.

- Eliminates the need for mapping between spacial addresses to IP addresses.
- No need of additional network control plane and management plane.
- Changes the topologies into a true network topologies.

We organized this chapter as follows. In the next section, we explain the insight for using a single address space for the MSC designs. In Section 6.2, we propose the encoding mechanism that uses a single address space. Simulations conducted and results obtained are given in Section 6.3. Finally we conclude with Section 6.4.

6.1 Insight for the proposed changes

Our main motive is to eliminate unnecessary mapping between two address spaces and use a single address space for both identifying the end hosts and routing. Before we explain the mechanism of converting these designs to use a single address space, we present insight into the proposed changes.

Recursive scalability and IP address hierarchy: In both DCell and BCube, the end servers are identified using an vector representation. An address in these designs is of the form $\langle a_k a_{k-1} \ldots a_0 \rangle$, where each a_i represents the index of i^{th} level subnet of $(i+1)^{th}$ level subnet. For example, DCell address $< 1, 2, 3, 2 >$ is the address for 3^{rd} server of 4^{th} $DCell_0$ of 3^{rd} $DCell_1$ of 2^{nd} $DCell_2$. This feature resembles super-netting feature of IP address hierarchy. Thus, our proposed encoding scheme is to encode the address vector inside an IP address to overcome the complexities result from use of two address spaces.

Network portability and interoperability : For data centers to deliver the anticipated benefits, they must be easy to use and the design must be cost-effective. To achieve these goals, the choice is to build data center as far as possible using the COTS switches. Many of the MSC designs except FiConn use multi-port servers. FiConn uses two Ethernet ports. Commodity server machines usually contain one or two built-in Ethernet ports, the second one serves as backup. Due to this reason, we modified the existing designs to use multi-port switches instead of multi-port servers. The proposed modification reduced the hop count for the path by one compared to the original designs and changed the original designs to a true network topology.

However, the downside of modification is, interconnections capacity needs to scale up to compensate the decreased bisection bandwidth. But, for a large systems such as data centers, a little scale-up to reduce a large scale-out is an adroit move. For example, for DCell who's growth rate is double exponential, large number of interconnections are required to interconnect the servers. Thus, a little scale-up helps to reduce the number of interconnection brings in additional advantages such as perceived simplicity in terms of configuration and management, and efficiency in terms of energy consumption. We discuss the need and consequences of flattening the designs in Section 6.2.4.

6.2 Design of DCell-IP and BCube-IP

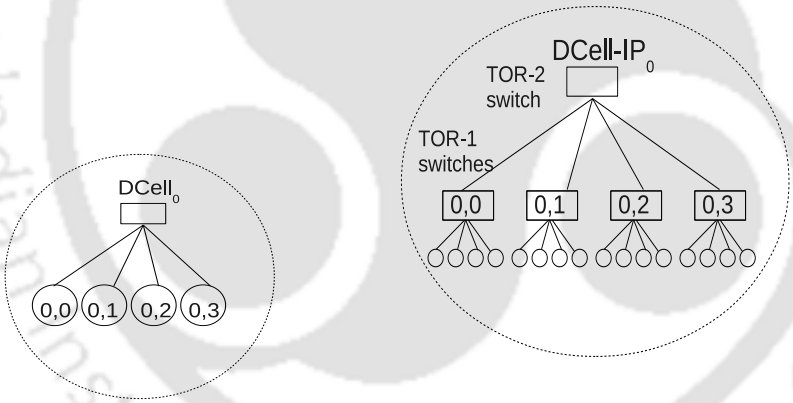
In this section, we explain the proposed encoding scheme for two designs DCell and BCube. The other designs can also be encoded in a similar manner by understanding their growth models.

6.2.1 DCell-IP

We call our proposed design DCell-IP to distinguish it from the original design DCell. In DCell-IP, we made one minor change to the original topology design.

Instead of using the servers as an end points of interconnections, servers are connected to a TOR switch and the TOR switch is used as end points of interconnections.

$DCell\text{-}IP_0$ is the basic building block to construct higher levels of DCell-IP. In $DCell\text{-}IP_0$, a TOR switch connected to a set of servers is used as end points of interconnections. We call this TOR switch a TOR-1 switch. In DCell, next levels are designed based on the value of n, number of servers connected to a TOR switch. But, in our design, since TOR-1 switch is used as a basic building block, the number of such TOR-1 switches connected to a TOR switch is used to determine the value of n. We call the later TOR switch a TOR-2 switch. We give the reason for using TOR-1 as end points of interconnections after we discuss address assignment. However, difference between $DCell\text{-}IP_0$ and $DCell_0$ is better understood from Fig.6.1.



(a) $DCell_0$ of DCell (b) $DCell\text{-}IP_0$ of DCell-IP

FIGURE 6.1: Difference between the baseline design of DCell and DCell-IP

A level-1 $DCell\text{-}IP_1$ is constructed from $n+1$ $DCell\text{-}IP_0$s. Since the value of n in $DCell\text{-}IP_0$ is four, $DCell\text{-}IP_1$ is constructed from five $DCell\text{-}IP_0$s as follows.

Assign each TOR-1 switch a 2-tuple $[a_1, a_0]$, where a_1 and a_0 are the level-1 and level-0 IDs, respectively. In our example design, range of values a_1 and a_0 take are [0,5) and [0,4) respectively. Then two TOR-1 switches with 2-tuples $[i, j-1] and [j, i]$ are connected with a link for every i and $j > i$. In $DCell\text{-}IP_1$,

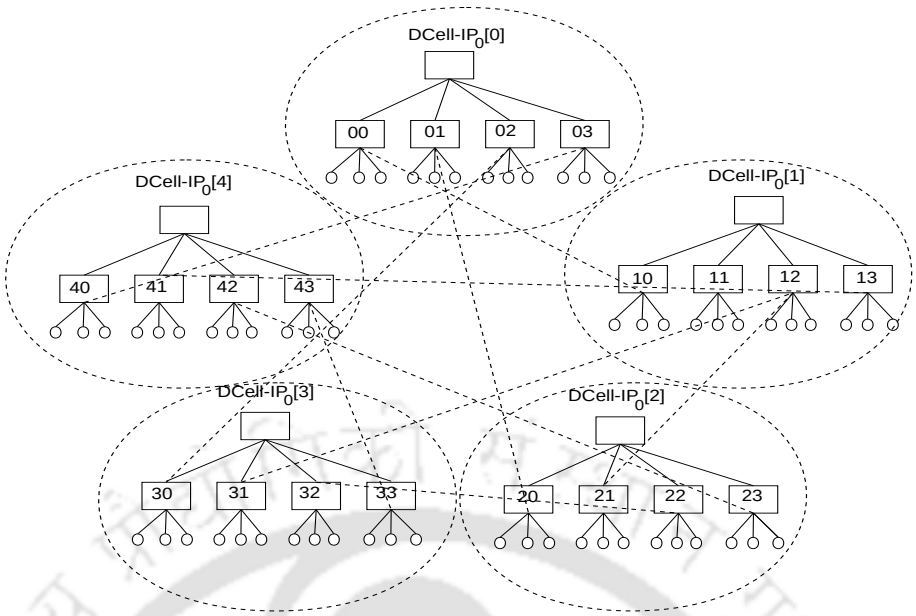


FIGURE 6.2: Design of $level-1$ of DCell-IP. $DCell\text{-}IP_0$s of DCell-IP are shown by circling then with dotted line

if each $DCell\text{-}IP_0$s is treated as a virtual node then $DCell\text{-}IP_1$ forms a fully connected graphs with $n+1$ virtual nodes. Design of $DCell\text{-}IP_1$ from five $DCell\text{-}IP_0$s is shown in Fig.6.2

Successive levels, $DCell\text{-}IP_{2,3,...,k}$ are constructed in a similar way. If $DCell\text{-}IP_{k-1}$ has t_{k-1} TOR-1 switches, then $DCell\text{-}IP_k$ is constructed from $t_{k-1}+1$ $DCell\text{-}IP_{k-1}$s. As a result, number of $DCell\text{-}IP_{k-1}$s in $DCell\text{-}IP_k$, denoted by g_k is $t_{k-1}+1$, where t_k is the number of TOR-1 switches in $DCell\text{-}IP_{k-1}$ and is defined as $g_k \times t_{k-1}$.

6.2.1.1 Address assignment

Addressing in DCell-IP is similar to DCell. But, instead of using a separate vector to store indices, indices are directly encoded in an IP address. To illustrate the encoding, we use X to denote an octet of IP address and x to denote a bit of an octet. Servers and intermediate switches are assigned IP addresses of the form 10.X.xxxxxxxx.X. The last octet (bits 0-7) of IP address is reserved for addressing the servers and interfaces of TOR-1 switches at $DCell\text{-}IP_0$ and rest of the octets are used for encoding levels. Number of bits used for encoding different levels is determined based on the growth model of respective design. For example, the

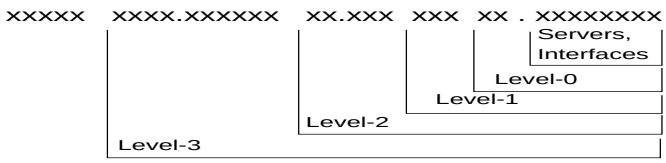


FIGURE 6.3: Encoding DCell address in an IP address

value for g_k at levels level-[0..3] are 4, 5, 21 and 421 respectively. Therefore, the number of bits used to encode these levels are - 2 ($2^2 = 4$), 3 ($2^3 = 8$), 5 ($2^5 = 32$) and 9 ($2^2 = 512$). The bits of IP address used for recording them are - 8-9, 10-12, 13-17 and 18-26 respectively. A process of encoding DCell address in an IP address is shown in Fig.6.3 and an example of encoding of few DCell addresses in IP addresses is shown in Table 6.2.

DCell Address	Encoding in IP	DCell-IP Address
$\langle 2,0,0,3\rangle$	0000101 0.000000 00. 010 000 00.3	10.0.64.3
$\langle 4,1,1,3\rangle$	0000101 0.000000 00. 100 001 01.3	10.0.133.3
$\langle 8,2,2,3\rangle$	0000101 0.000000 01. 000 010 10.3	10.1.10.3

TABLE 6.2: Encoding DCell addresses in IP addresses

6.2.1.2 Encoding complexity

In DCell and BCube, data forwarding is done using the IP addresses. Hence, special addresses in these design need to map to IP address during actual data transfer. To simplify this mapping, DCell first converts special addresses to uniform identifiers (UIDs) using the formula $uid_k = a_0 + \sum_{j=1}^{k}\{a_j \times t_{j-1}\}$ for the special address $\langle a_k, a_{k-1}, ..., a_1, a_0\rangle$. Similarly, in BCube, special address $\langle a_k, a_{k-1}, ..., a_1, a_0\rangle$is mapped to UID using the formula $uid_k = \sum_{j=0}^{k} a_j n^j$, where n is the number of TOR-1 switches at $level_0$. During routing, DCell extracts DCell addresses for the source and destination from their UID's. Then, using these DCell addresses, source determines path to the destination. Further, the computed path in DCell addresses is encoded back into UIDs and used in source routing.

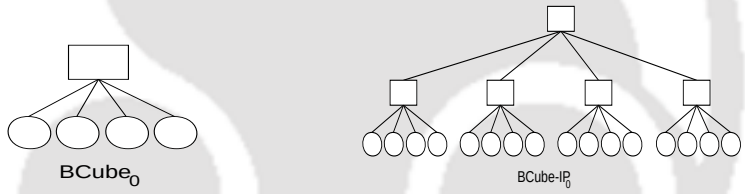
DCell-IP performs similar encoding/decoding. During the initialization, DCell addresses are encoding into IP addresses and assigned to end hosts and intermediate switches. During route calculations, the DCell addresses embedded in IP

addresses are extracted, computation of the route is done using these address and the DCell addresses in the computed path are encoded back into IP addresses.

Since both the designs result in same encoding complexity, we can claim our designs do not incur any additional complexity compared to the original design.

6.2.2 BCube-IP

Process of deriving BCube-IP from BCube is similar to the derivation of DCell-IP from DCell. In BCube-IP, a TOR-1 switch connected to servers is used as a end points for interconnections. A $BCube\text{-}IP_0$ is formed by connecting m such TOR-1 switches to a TOR-2 switch. The difference between the $level_0$ designs of BCube and BCube-IP is shown in Fig.6.4. Next, $BCube\text{-}IP_1$ is constructed from m $BCube\text{-}IP_0$s and m m-port switches.



(a) $BCube_0$ of BCube (b) $BCube\text{-}IP_0$ of BCube-IP

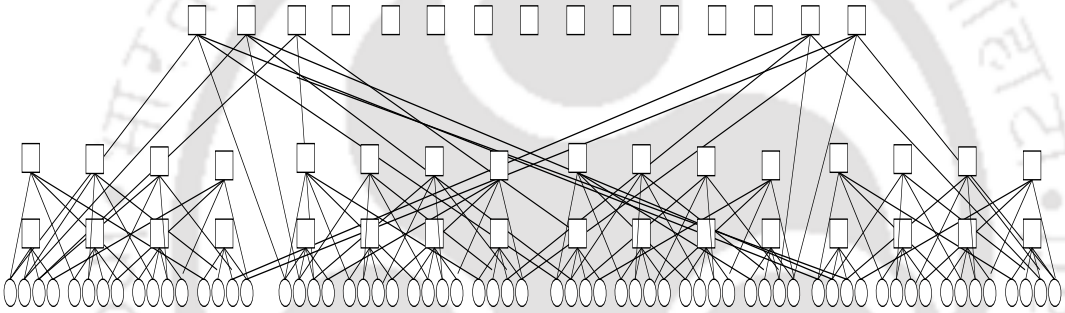
FIGURE 6.4: The difference between baseline design of BCube and BCube-IP

In general, $BCube\text{-}IP_k$ is constructed from m $BCube\text{-}IP_{k-1}$ and m^k m-*port* switches. Each TOR-1 switch in $BCube\text{-}IP_k$ has $x + k + 1$ ports. The first x ports are connected to the servers at $BCube\text{-}IP_0$ and remaining $k + 1$ ports are connected to TOR-2 switches from $level_0$ to $level_k$. The design of $BCube\text{-}IP_k$ from m $BCube\text{-}IP_{k-1}$ is done as follows. The m TOR-1 switches each from each $BCube\text{-}IP_{k-1}$ are connected through a TOR-2 switch. Precisely, all the i^{th} switches from each $BCube\text{-}IP_{k-1}$ are connected to j^{th} TOR-2 switch at $BCube\text{-}IP_k$, where both $i, j \in [0, m^k - 1]$. The following pseudocode explains this procedure.

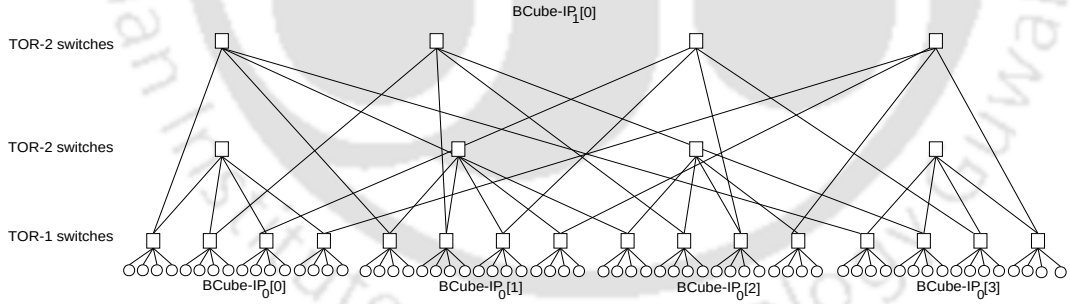
for $i = 0$ to $m^k - 1$ **do**
  {***Index for both switches at*** $level_k$ ***and TOR-1 switches in*** $BCube\text{-}IP_{k-1}$}
  for $j = 0$ to $m - 1$ **do**
    {***Index of*** $BCube\text{-}IP_{k-1}$ ***in*** $BCube\text{-}IP_k$}
    Connect i^{th} switch at $level_k$ to i^{th} TOR-1 switch of j^{th} $BCube\text{-}IP_{k-1}$
  end for
end for

Steps involved in design of BCube-IP

The difference between BCube and BCube-IP is shown in Fig.6.5.



(a) Original design of BCube



(b) BCube-IP using IP address Hierarchy

FIGURE 6.5: Reduction of number of switches and interconnection with new design

Next, encoding BCube addresses in an IP address is similar to DCell-IP and is even simple because of its regular growth model. As in DCell-IP, last octet of IP address is used for assigning addresses to servers and interfaces of $BCube\text{-}IP_0$. Next, as $BCube\text{-}IP_i$ is formed from n $BCube\text{-}IP_{i-1}$s, successive $p = [\log_2 m]$ bits starting from third octet are used to encode each level. For example, if we form

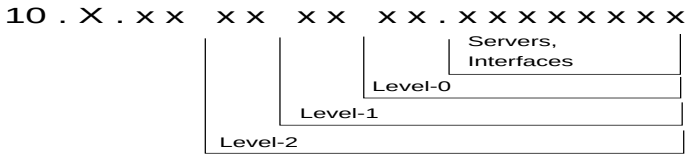


FIGURE 6.6: The bits of IP address used to encode different levels of BCube-IP

BCube-IP$_i$ from four *BCube-IP*$_{i-1}$s, then successive two bits are used to encode each level. The bits 15-14, 13-12, 11-10 and 9-8 are used to encode *BCube-IP*$_{3,2,1,0}$ respectively. The encoding scheme for BCube-IP is shown in Fig.6.6

6.2.3 Routing

DCell uses source routing for data transfer [28]. If two nodes *src* and *dst* are in same $DCell_k$, but in different $DCell_{k-1}$, then an intermediate node (n_1, n_2) that connects two $DCell_{k-1}$ is determined. Then, recursively, the paths from *src* to n_1 and n_2 to *dst* are determined. Routing in DCell-IP uses the routing algorithm of DCell to compute the route. For this, DCell-IP first extracts from the IP addresses format of source (*src*) and destination (*dst*), the vector representation of DCell addresses, $\langle a_k, a_{k-1}, ..., a_1, a_0\rangle$ and $\langle b_k, b_{k-1}, ..., b_1, b_0\rangle$, respectively. Then, by using DCell addresses, the path from source to destination is computed using original DCellRouting. Afterwards, the special addresses of the nodes in the path obtained from DCell routing are encoded into IP addresses and IPv4 source routing is used for data transfer. Routing algorithm for DCell-IP is given in Algorithm 6.1.

We explain our proposed routing algorithm with an example for the source 10.0.1.1 (10.0.000 000 01.1) and destination 10.0.18.1 (10.0.000 100 10.1). From the source and destination IP addresses, DCell addresses are extracted as $\langle a_2, a_1, a_0\rangle = \langle 0, 0, 1\rangle$ and $\langle b_2, b_1, b_0\rangle = \langle 0, 4, 2\rangle$. Let the path returned from the original source routing for DCellRouting be $(\langle 0, 0, 1\rangle\langle 0, 0, 3\rangle\langle 0, 4, 0\rangle\langle 0, 4, 2\rangle)$. Encoding this path into IP addresses, results with the path $(10.0.1.1, 10.0.3.0, 10.0.16.0, 10.0.18.1)$. This path is then used in source routing for data transfer.

Algorithm *DCell-IP-Routing(src, dst)*

Input :

src in IP Address format

dst in IP Address format

Output: *Path from src to dst*

BEGIN

Let $A = \langle a_2, a_1, a_0 \rangle$ be the DCell address of *src*

Let $B = \langle b_2, b_1, b_0 \rangle$ be the DCell address of *dst*

Mask last 8 bits of Saddr, Daddr

Extract bits 17-13, 12-10 and 9-8 of *src* into A

Extract bits 17-13, 12-10 and 9-8 of *dst* into B

$PATH$ = DCellRouting(A,B)

IP-PATH = Encode-Into-IP ($PATH$) .

RETURN *IP-PATH*

END

Algorithm 6.1: **Source routing in DCell-IP**

Routing in BCube-IP is similar to DCell-IP. First, addresses in vector form are extracted from the IP address format of source and destination addresses and the route is computed using the original BCube routing. The return path from BCube routing is then encoded back into IP addresses and IPv4 source routing is used for data transfer.

6.2.4 Effect of topology modification

Due to symmetry in interconnection, packets need to travel an extra hop in BCube and DCell. Precisely, n i^{th} server each from different $level_i$ subnets are connected through a common switch. These servers act as transporter for the message received for other servers in the same $level_0$ subnet. Thus, if the destination is other than the transporter server then packet has to move from transporter to the TOR

switch and from there to the destination. Contrast to this, in our proposed design, as the last octet of IP address is used to assign the address for all the servers connected to the TOR-1 switch, they all lie in a single broadcast domain. Thus, when a packet reaches the TOR-1 switch, it is simply broadcasted to reach all of them.

Thus, use of the TOR-1 switch as the end points of interconnections reduces hop count by one. The proposed modification also help in two ways. It reduces number of interconnections and transform the design into a true network topology. In both the designs BCube-IP and DCell-IP, the number of interconnections used at $level_k$ are equivalent to the number of interconnections used by BCube and DCell at $level_{k-1}$ respectively. This can be noticed from the Fig 6.5 in Section 6.2.2. The downside of flattening is, interconnections capacity has to scale up to compensate the decreased bisection bandwidth. But for a large designs such as DCNs a little scale-up in the place of large scale-out is an adroit move. Further, the proposed change has changed the designs into real networking topology. This is because conventional networks favors multi-port switches rather than multi-port servers. The analysis of the effect of flattening on network topologies is given below.

6.2.4.1 Properties of DCell-IP

Here, we analyze the decrease in bisection bandwidth in DCell-IP due to the proposed modification.

In $DCell\text{-}IP_k$, the number of flows that pass through $level_i$ link is smaller than $2^{k-i}\,nt_k$, where t_k are the total number of TOR-1 switches in $DCell\text{-}IP_k$ and n is the number of servers connected to TOR-1 switch of $DCell\text{-}IP_0$. Therefore, $level_1$ links carries the maximum flow. Let C be the number of flows in both the directions through $level_1$ link, that is, $C = 2\,2^k\,n\,t_k$

Note that $(n' + \frac{1}{2})^{2^k} - \frac{1}{2} < t_k < (n' + 1)^{2^k} - 1$, where n' is the number of TOR-1 switches at $DCell\text{-}IP_0$. Therefore, $C \leq 2\,n\,t^k$ [28]

Let, B_k be the bisection width of $DCell\text{-}IP_k$ and let B_G be the bisection width of directed complete graph G with $n\,t^k$ nodes. Let us embed G into $DCell\text{-}IP_k$ as

in [113] and let E denotes the set of edges in the bisection of $DCell\text{-}IP_k$, Hence $|E| = B_k$

Bisection E partitions $DCell\text{-}IP_k$ into subgraphs $DCell\text{-}IP_k^1$ and $DCell\text{-}IP_k^2$ such that $|V(DCell\text{-}IP_k)^1)| = \lfloor \frac{n\,t_k}{2} \rfloor$ and $|V(DCell\text{-}IP_k)^2)| = \lceil \frac{n\,t_k}{2} \rceil$

Where $V(H)$ is the number of nodes in any graph. Consider the subgraphs G_1 and G_2 of G induced on $V(DCell\text{-}IP_k)^1)$ and $V(DCell\text{-}IP_k)^2)$ respectively. Then, there must be atleast B_G edges with one end point in G_1 and another end point in G_2. Therefore, the total number of flows through E must be atleast B_G. We have seen that the number of flows in any link in $DCell\text{-}IP_k$ is at most C and bisection B_G in directed complete graph G is $\frac{(n\,t_k)^2}{2}$. Hence the total number of flows through E is at most $C\,|E| = C\,B_k$. Therefore,

$$C\,B_k \geq B_G$$
$$B_k \geq \frac{B_G}{C}\frac{(n\,t_k)^2}{2\,C} = \frac{n^2\,{t_k}^2}{2(2\,n\,t_k\,log_{n'}(n\,t_k))}$$
$$\Rightarrow \frac{n\,t_k}{4\,log_{n'}(n\,t_k)}$$

For the fixed number of TOR-1 switches, that is, n' is constant. Thus, the congestion (the number of flows) through any link in $DCell\text{-}IP_k$ is larger than the one in $DCell_k$

6.2.4.2 Properties of BCube-IP

In all to all traffic condition, the aggregate bottleneck throughput (ABT) is defined as the product of number of flows and the throughput of the bottleneck flow [29] where bottleneck flows are those flows which receive smallest throughput among all the flows in a system. Finally, the finish time for all the flows is calculated by dividing total data exchanged by ABT.

The ABT of the BCube network under all to all traffic model is defined as $\frac{m}{m-1}(N-1)$, where m is the number of servers at $BCube_0$ and N is the total number of servers in $BCube_k$.

Lemma 6.1. *The aggregate bottleneck throughput for BCube-IP is* $\frac{(m\times k)}{(m-1)\times(k+1)} \times (N-1)$*, Where* m *is the number of TOR-1 switches connected to BCube-IP*$_0$*,* N *is the total number of TOR-1 switches and* k *is the highest level.*

Proof. Here, we consider TOR-1 switches of *BCube-IP*$_0$ as end points of interconnections because responsibility of source routing is in only reaching the TOR-1 switches. In BCube-IP, a TOR-1 switch communicating with all the remaining $m^{k+1}-1$ TOR-1 switches are classified into k groups, where each $Group_i$ contains group of servers that are i hop away from the server. The number of servers in $Group_i$, where $(i \in [1,k])$, are $\binom{k+1}{i}(m-1)^i$. The average path length of all the flows is,

$$\begin{aligned} \textit{ave-plen} &= \tfrac{1}{m^{k+1}-m}\textstyle\sum_{i=1}^{k} i\binom{k+1}{i}(m-1)^i \\ &= \binom{k+1}{1}(m-1) + 2\binom{k+1}{2}(m-1)^2 + ... + (k+1)\binom{k+1}{k+1}(m-1)^k \\ &= (m-1)\left(\binom{k+1}{1} + 2\binom{k+1}{2}(m-1) + ... + (k+1)\binom{k+1}{k+1}(m-1)^{k+1}\right) \\ &= \tfrac{(m-1)(k+1)}{m}N. \end{aligned}$$

Using *ave-plen* , number of flows carried in each link, *f-num*, is estimated as

$$\textit{f-num} = \tfrac{(totalnumberofflows)(averagepathlebgth)}{totlnumberoflinks}.$$

In BCube-IP, total number of flows are $N\,(N-1)$, where N are the total number of TOR-1 switches in BCube-IP and total links are $N(k)$. Whereas in BCube, total number of links are $N(K+1)$. In BCube-IP number of interconnection are reduced by one level because of using TOR-1 switches as end points of interconnections.

$$\begin{aligned} \textit{f-num} &= \tfrac{N(N-1)\textit{ave-plen}}{NK} \\ &= \tfrac{N(m-1)(k+1)}{mk} \end{aligned}$$

Therefore, throughput the one flow receives is $\frac{1}{f-num}$ and ABT is $N(N-1)\frac{1}{f\text{-}num}$

$$\text{ABT} = \tfrac{mk}{(m-1)(k+1)}(N-1)$$

Therefore, ABT for the flows in BCube-IP is $\frac{mk}{(m-1)(k+1)}(N-1)$. □

Compared to this, ABT for BCube is $\frac{m}{m-1}(N-1)$. Thus compared to BCube, ABT in BCube-IP decreases by a factor of $\frac{k}{k+1}$.

6.2.5 Location based routing in BCube-IP

In our proposed location based routing for 4-4, 1-4, addresses assigned to the end hosts provide their relative location. For this, the address assignment must correlate with the topology design. Since topology design and address assignment in BCube-IP exhibit this correlation, we extended our proposed location based routing to BCube-IP.

4-4, 1-4, packet goes from the source to highest subnet of destination and from there to the exact smaller subnet of the destination. In BCube-IP, correcting a bit of hamming distance is equivalent one step towards reaching the destination. But, compared to 4-4, 1-4, the journey a packet takes in BCube-IP may not be in order. Based on the order in which hamming distances are corrected, the path is determined.

Location based routing for BCube-IP is similar to the one proposed for 4-4, 1-4 wherein, an intermediate TOR-1 switch determines next hop by comparing its address with the destination address. For this, the TOR-1 switch computes hamming distance between the two addresses and corrects a bit randomly to determine the next hop. This process is repeated at successive hop until the packet reaches the TOR-1 switch of the destination. The algorithm for location based routing in BCube-IP is given in Algorithm 6.2. In the algorithm, a bit is a representative for a set of bits used to encode each level.

An example for computing a path using our proposed location based routing is shown in Fig.6.7. Successive hops in the path from source S (10.0.0.1) to destination D (10.0.15.4) are shown with numbers. At every hop, the bits compared to determine the next hop are shown in boxes. For example, at the first hop, addresses compared after masking are - 10.0.00000000.0 (SW - Intermediate switch address) and 10.0.00001111.0 (D - Destination address). The intermediate switch randomly correct one of the difference bit to obtain the next hop 10.0.00001100.0. The packet is then transmitted to the first TOR-1 switch in fourth $BCube\text{-}IP_0$. Similar comparison at the first TOR-1 switch of fourth $BCube\text{-}IP_0$ transfers the packet to the first TOR-1 switch of that $BCube\text{-}IP_0$.

Algorithm *LB-Routing-BCube-IP (Swaddr, Daddr)*

Input: ***Swaddr***(Switch Address), ***Daddr***(Destination Address)

Output: *Next hop in forwarding path*

1: Mask last 8 bits (Netmask = 255.255.255.0) of Swaddr and Daddr
2: **if** Swaddr == Daddr **then**
3: Packet belongs to the same $BCube_0$, send the packet directly to destination.
4: **else**
5: Extract the bits that are different
6: Randomly correct a bit to obtain address of the next hop
7: Determine interface through which the packet to send.
8: **end if**

Algorithm 6.2: Algorithm for forwarding the packet at switch for location based routing in BCube-IP

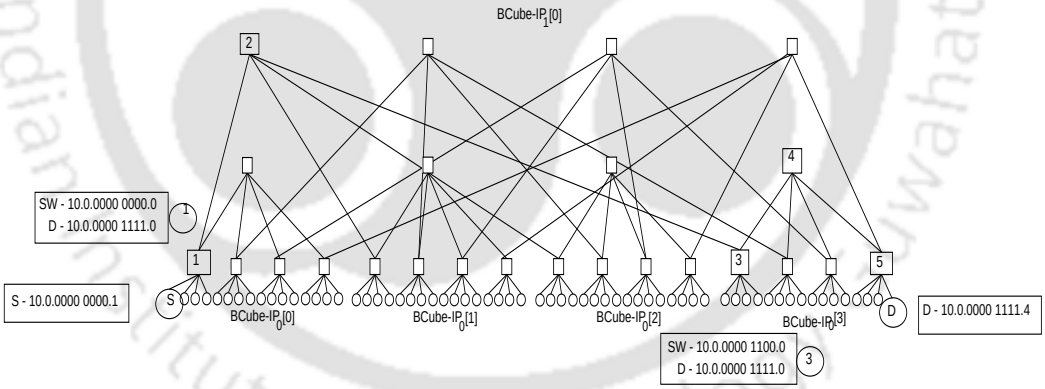


FIGURE 6.7: Location based routing in BCube-IP

Similar to BCube-IP, extending location based routing for DCell-IP is not possible because of recursive source routing. Location based routing works in a sequence to find a path from the source to destination and extending the same for recursive routing is more challenging.

Comparison between DCell-IP and BCube-IP with DCell and BCube are given in Table 6.3 and Table 6.4 respectively .

Architecture	Addressing scheme	Routing	Extra space for address translation	g_k, t_k defined in terms	Maximum path length	Diameter	ABT
DCell	DCell Address, IP Address	DCell Source Routing	$O((t_k)^2)$	Servers	$2^{k+1}-1$	$2^{k+1}-1$	$\frac{t_k}{4\,log_n\, t_k}$
DCell-IP	IP Address	IP Source Routing,	None	TOR switches	$2^{k+1}-3$	$2^{k+1}-3$	$\frac{n\,t_k}{4\,log_{n'}(n\,t_k)}$

TABLE 6.3: System Comparison between DCell and DCell-IP

Architecture	Addressing scheme	Routing	Extra space for Address Translation	Switches at at $BCube_k$	Interconnection at $BCube_k$	Diameter	ABT
BCube	BCube Address IP Address	BCube Source Routing	$O((m^{k+1})^2)$	m^k-1	$(m^k-1)\times m$	$k+1$	$\frac{m}{m-1}(M-1)$
BCube-IP	IP Address	IP Source Routing, Location Based Routing	None	$m^{k-1}-1$	$(m^{k-1}-1)\times m$	k	$\frac{mk}{(m-1)(k+1)}(N-1)$

TABLE 6.4: System Comparison between BCube and BCube-IP

6.3 Simulation

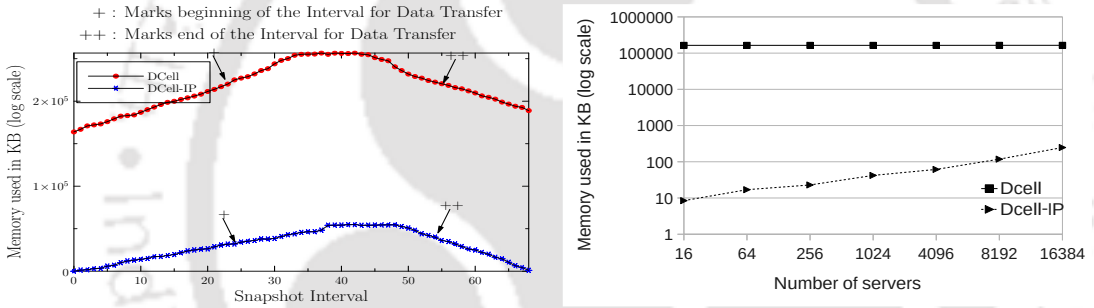
Simulations are conducted using network-simulator (NS3). To study the performance of our proposed source routing, we compare the performance of our proposed source routing with the original source routing proposed for DCell and BCube. For the simulation, we designed different DCN architectures supporting 4096 servers and their respective routing protocols for data transfer. For DCell and BCube, we stored mapping between two address in a single file and configured every node to obtain mapping from that file as oppose to real implementation where each node maintains its own mapping. This is done to keep the implementation of node is NS3 simple. To compare the performance of two variant source routing, we measured amount of resources (memory and CPU) used by individual designs for data transfer. Finally, we compare their performance by measuring delays incurred during data transfer .

6.3.1 Memory usage

The performance of routing algorithms is compared by measuring the amount of memory used by each implementation. For this, 1000 flows, each sending a data of 500 MB are used as a data traffic. Further, all the links are configured with a capacity of 10 Mbps. To compar the performance of routing algorithm, memory usage of the simulation program implementing these source routing is measured using valgrind [114]. Valgrind is a memory profile tool that provides memory usage of running program at different instances. The memory usage for different designs are given in Fig.6.8(a) and Fig.6.9(a). In the graph, the memory usage by original DCell and BCube designs are higher compared to DCell-IP and BCube-IP respectively. There are many factors that caused increase in the memory usage by original deign. First, presence of the mapping table consumes a considerable amount of memory. Next, the mapping function, which maps special addresses to IP addresses for all the packets also consumes memory. In the original simulation result, we obtained a marginal difference between the memory usage because of

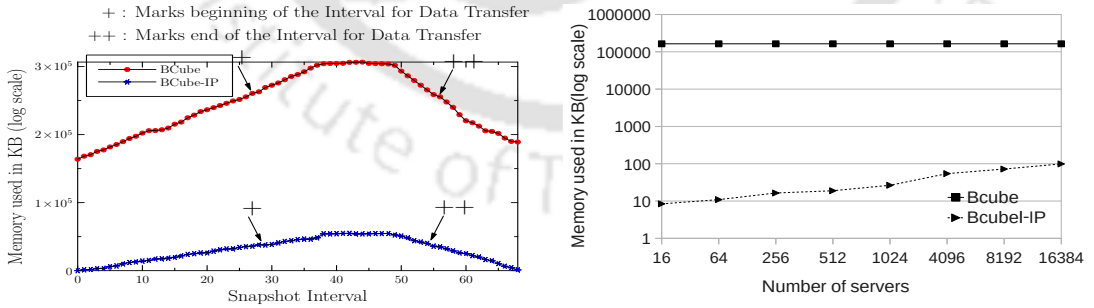
using a single file for mapping. But, as original implementations maintains mapping with every node, we extrapolated the graphs to include memory usage for mapping by all the nodes. For this, we considered, each mapping is of 10 bytes.

Next, we measured the difference between the memory usage for different implementations by varying the number of servers they support. The memory usage for each implementation is done by measuring the peak memory usage by that design during the simulation. The measured values for memory usage are given in graph Fig.6.8(b) and Fig.6.9 (b). The graphs indicate more memory used by DCell variants compared to BCube variants. This is due to the recursive routing algorithm used in DCell variants that results in more number of hops compared to BCube. For instance, maximum path length in BCube is k and in DCell is $2^{k+1}-1$, where k is the number of levels. Next, to establish the increase in mem-



(a) Memory usage for different implementations (b) Growth of memory for different implementations

FIGURE 6.8: Memory analysis for different DCell implementations.



(a) Memory usage for different implementations (b) Peak memory usage for different implementations

FIGURE 6.9: Memory usage for different implementations of BCube

ory usage due to mapping function, we measured the performance of CPU using Callgrind and Kcachegrind [115]. From the output of Kcachegrind, we measured the percentage of CPU used by routing algorithm for the simulation programs

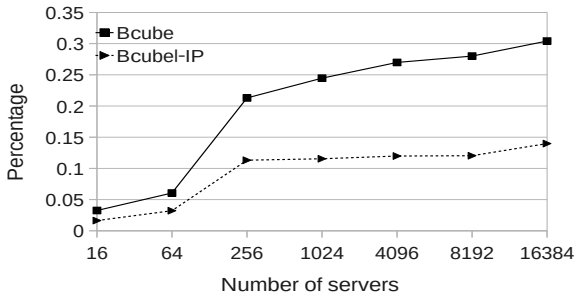


FIGURE 6.10: CPU utilization for Different BCube implementations

implementing different variants of BCube. The difference between the percentage of CPU used by each routing algorithm for different implementations of BCube is given in Fig.6.10. The call graph of Kcachegrind gives the percentage of the CPU used by each of the functions in the program. Analyzing the call graph for BCube and BCube-IP, we found increase in the percentage of CPU utilization in BCube is due to the mapping function that was called for every data transfer.

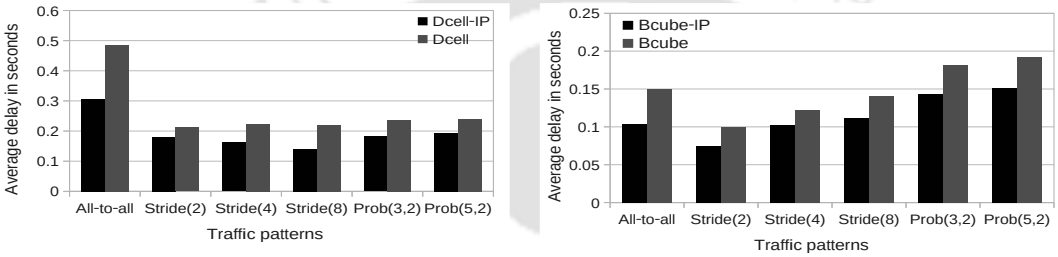
6.3.2 Route delay

To demonstrate the efficiency of our proposed routing algorithms, we measured the average delay for flows in different traffic conditions. To measure the delay, we configured each link with the capacity 10 Mbps and used bulk send application as a traffic generator with each flow of size 500 MB. The following bench mark suite is used for measuring delay.

- *All-to-all*: Every node sends traffic to every other node.
- $Stride(i)$: A host at x $DCell_0$ of y $DCell_k$ sends traffic to a host at $x+i$ $DCell_0$ of $y+i$ $DCell_k$.
- $StaggeredProb(Level3P, Level4P)$: A host at x $DCell_0$ sends traffic to a host at y $DCell_0$, where x and y belong to a same level-3 with the probability of $level-3p$. Next, a host at x $DCell_0$ sends traffic to a host at y $DCell_0$, where x and y belong to same level-4 with the probability of $level4p$. Finally,

a host at $DCell_0$ sends traffic to a host at an arbitrary $DCell_0$ with the probability of $(1 - level3P + level4P)$.

The average delay for flows in DCell is more compared against DCell-IP to study the effect of mapping function that introduces an additional delay to process the packet header. The average delay for the flows in different traffic conditions is given in Fig.6.11(a). The average delay for the flows in DCell-IP are lower compared to DCell. Similar results are obtained for the flows in BCube-IP. The comparison of route delays for BCube and BCube-IP is given in Fig.6.11(b).



(a) Average delay for a flow in different DCell implementations

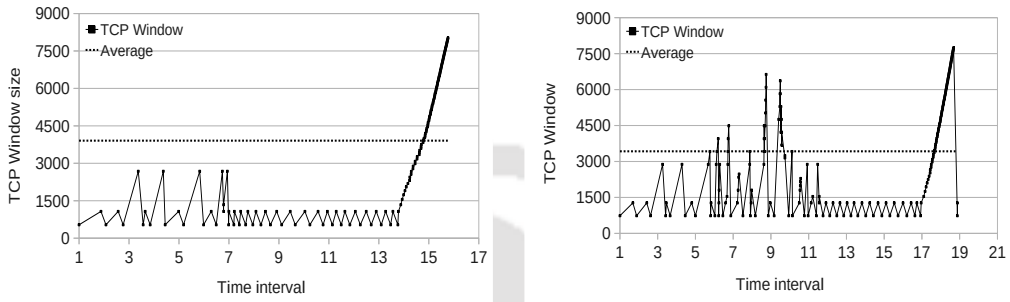
(b) Average delay for a flow in different BCube implementations

FIGURE 6.11: Delay analysis of different implementation of DCell and BCube

6.3.3 Routing in BCube-IP

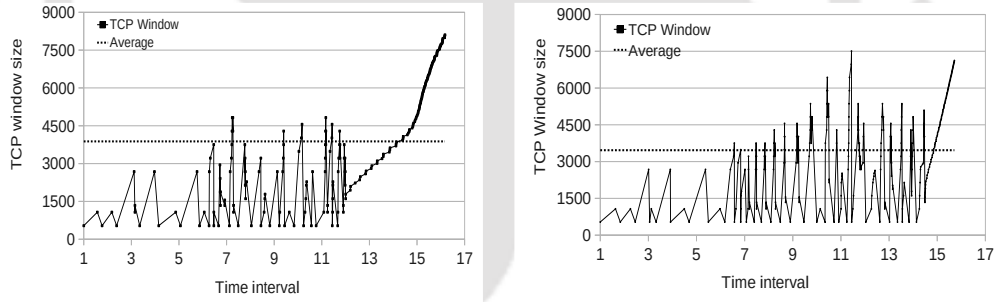
Since, algorithms defined for alternative routing are based on the underlying characteristics of the topology, in the subsection we study the performance of routing protocols for the architectures *BCube-IP* and 4-4, 1-4. For this, we first compare the performance of source routing and location based routing on *BCube-IP* and then we study the performance of location based routing for the designs *BCube-IP* and 4-4, 1-4. For source routing, we implemented load balancing where the source probes all the available multiple paths for making the routing decision. For location based routing, load balancing is done by simply distributing flows using ECMP. Finally, two kinds of traffic are used for study - *fine-spread* traffic and congestion traffic. Both the traffic patterns are based on pair of nodes selected as source and destination for each flow. In the *fine-spread* traffic we ensured no two flows

are overlapping with each other and in *congestion* traffic, we induced congestion through many overlapping flows. To study the performance of routing protocols, we observed the behavior of TCP by measuring TCP window for different flows and derived bandwidth utilization for each routing protocol. The measured window sizes for different routing protocols and for different architectures are given in Fig.6.12, Fig.6.13 and Fig.6.14.



(a) Source routing for the fine-spread traffic (b) Source routing for all-to-all traffic

FIGURE 6.12: TCP Congestion window of a flow in BCube-IP for source routing

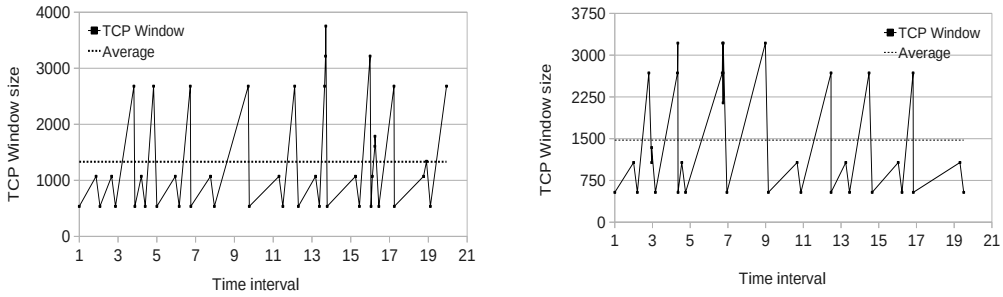


(a) Location Based Routing for fine-spread traffic (b) Location Based Routing for all-to-all traffic

FIGURE 6.13: TCP Congestion window of a flow in BCube-IP for Location based routing

6.3.3.1 Performance study based on TCP window

Each node participating in a TCP connection advertises its available buffer space using TCP window size. In a network, TCP window measures the throughput of the network as it reflects the fraction of the available bandwidth used by flows. Two factors bandwidth and latency, combine to influence the potential impact of a given TCP window size.



(a) Location Based Routing for fine-spread traffic (b) Location Based Routing for all-to-all traffic

FIGURE 6.14: TCP Congestion window of a flow in 4-4, 1-4 for Location based routing

	Average Window size	RTT	Throughput
Source routing for BCube-IP (fine-spread traffic)	3906	674.2	46.34
Source routing for BCube-IP (congestion traffic)	3624	704.2	41.17
Location based routing for BCube-IP (fine-spread traffic)	3879	791.35	39.21
Location based for BCube-IP (congestion traffic)	3464	864.42	32.05
Location based routing for 4-4,1-4 (fine-spread traffic)	1474	224.31	52.57
Location based routing for 4-4,1-4 (congestion traffic)	1331	238.38	46.64

TABLE 6.5: Estimation of throughput using the measured TCP window size and RTT. This measures gives the performance of different routing algorithms on BCube and 4-4, 1-4 architectures

Thus, we derive maximum throughput of a TCP connection using the receiver window size as

$$\text{Bandwidth} = (\text{Window size} * 8)/\text{RTT}$$

To estimate the bandwidth, we measured the average window size in each case and measured delay by taking the average of RTT measured at different intervals. The values for these parameters in different cases are given in Table.6.5. These results establish the importance of architecture features in devising routing protocols for DCN. For BCube-IP, source routing performs better compared to location based routing. However, the performance of our location based routing in 4-4, 1-4 is better compared to BCube-IP.

6.4 Conclusion

The source routing uses the topology characteristics of the underlying architecture to define their routing algorithm. For this, they encode the topology information in special addresses that enable them to determine the path during routing. However, since real data transfer needs IP addresses for identifying end hosts, the special addresses are mapped to IP addresses during routing.

In this work, we studied the complexities that result from use of two address spaces and proposed a method to overcome those complexities. The same correlation between the topology design and address assignment that helped us to propose 4-4, 1-4 is used for the proposed modification. We redesigned MSC designs to use a single address space for both identifying end hosts and routing by embedding the special addresses in IP addresses. Further, the correlation between topology design and address assignment enabled us extend our proposed location based routing proposed for 4-4, 1-4 to BCube-IP.

Chapter 7

Conclusions and future works

Data center networks are bringing new innovations in the network world in many ways. The fact that the entire network stack, from hardware to the software stack on servers, is under the control of a single administrative domain provided an opportunity to design the networks in a new way. This motivated many researchers to propose scalable protocols using the structural properties of the underlying topology designs.

4-4, 1-4 architecture, proposed based on IP address hierarchy enabled to determine the location of the end hosts directly from the addresses assigned to them. This eliminated the need for additional control plane to determine location information in the earlier designs for location based routing. It also helped in devising a location based routing which uses the location information provided by 4-4, 1-4 architecture for routing. Our proposed location based routing is better compared to earlier proposed routing schemes as it the covers entire path of the route using only the location information. We compared the topological characteristics of 4-4, 1-4 with fat-tree, BCube and DCell for cost and maximum path length. The results shown a reduction in total cost for designing 4-4, 1-4 topology compared to fat-tree. Next, the simulations conducted to measure the performance of different routing algorithms shown better performance of our proposed location based routing in terms of throughput and delay.

To conserve energy, we proposed a distributed greedy algorithm to select a

sub-network that can accommodate all the flows of a current traffic and switch *off* the remaining network to save power. The efficiency of 4-4, 1-4 in conserving energy using our proposed algorithm is compared with ElasticTree using a class of MapReduce programming model ETLMR. The simulations done to compare the power efficiency of both the designs confided better performance of 4-4, 1-4 due to the hierarchical topology and mesh type interconnections.

The packet scheduler proposed for 4-4, 1-4 effectively make use of the path diversity provided by 4-4, 1-4 to perform load balancing. The proposed packet scheduler does the load balancing by prioritizing and scheduling individual packets of the flow based on the deadline values of each packet. By doing so, the packet scheduler enables more flows to meet their deadline. The simulation results shown an increase in number of flows which meet their deadlines through our packet scheduler.

In our last work, we extended the concept of IP address hierarchy we used for designing 4-4, 1-4 to propose an encoding scheme that simplified the designs proposed for source routing. The proposed encoding enabled to use the same IP addresses for both identifying end hosts and routing. This overcomed the problem of maintaining a mapping between two address spaces in the earlier designs for source routing. The simulation results shown a reduction in resource usage in terms of memory usage and CPU usage due to our proposed encoding.

7.1 Future works

Our proposed architecture 4-4, 1-4 makes a way for similar approaches that can bring performance enhancements for the DCNs.

The proposed 4-4, 1-4 topology design uses the concept of subnetting and supernetting. It would be interesting to design new topologies based on this principle. Minimizing the number of fusion switches without compromising a performance makes an interesting study.

In multi cast routing, the receiver send the multicast group membership discovery to join or leave the group and the multicast routing protocol helps to calculate

the multicast distribution tree of receiving hosts. Studying the performance of multicast routing on 4-4, 1-4 makes an interesting study as both the protocols may benefit from hierarchical address assignment. The group join messages from different receivers may be aggregated in a path to local multicast router. Similarly, the multicast distribution tree can be effectively built for set of receivers with hierarchical addresses.

Finally, our study of meeting SLAs for the flow can be extended to study the performance of various cloud computing applications and their associated problems such as supporting the elasticity of the deployed application, their performance isolation and supporting multi tenant environment on 4-4, 1-4 architecture.

Bibliography

[1] Luis M. Vaquero, Luis Rodero-Merino, Juan Caceres, and Maik Lindner. A break in the clouds: towards a cloud definition. *SIGCOMM Comput. Commun. Rev.*, 39(1):50–55, December 2008. ISSN 0146-4833. doi: 10.1145/1496091.1496100. URL http://doi.acm.org/10.1145/1496091.1496100.

[2] Peter Mell and Tim Grance. The nist definition of cloud computing. 2011.

[3] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica, and Matei Zaharia. A view of cloud computing. *Commun. ACM*, 53(4):50–58, April 2010. ISSN 0001-0782. doi: 10.1145/1721654.1721672. URL http://doi.acm.org/10.1145/1721654.1721672.

[4] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Commun. ACM*, 51:107–113, January 2008. ISSN 0001-0782. doi: http://doi.acm.org/10.1145/1327452.1327492. URL http://doi.acm.org/10.1145/1327452.1327492.

[5] Michael Isard, Mihai Budiu, Yuan Yu, Andrew Birrell, and Dennis Fetterly. Dryad: distributed data-parallel programs from sequential building blocks. *SIGOPS Oper. Syst. Rev.*, 41(3):59–72, March 2007. ISSN 0163-5980. doi: 10.1145/1272998.1273005. URL http://doi.acm.org/10.1145/1272998.1273005.

[6] Theophilus Benson, Aditya Akella, and David A. Maltz. Network traffic characteristics of data centers in the wild. In *Proceedings of the 10th annual conference on Internet measurement*, IMC '10,

pages 267–280, New York, NY, USA, 2010. ACM. ISBN 978-1-4503-0483-2. doi: http://doi.acm.org/10.1145/1879141.1879175. URL http://doi.acm.org/10.1145/1879141.1879175.

[7] Theophilus Benson, Ashok Anand, Aditya Akella, and Ming Zhang. Understanding data center traffic characteristics. In *Proceedings of the 1st ACM workshop on Research on enterprise networking*, WREN '09, pages 65–72, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-443-0. doi: http://doi.acm.org/10.1145/1592681.1592692. URL http://doi.acm.org/10.1145/1592681.1592692.

[8] Brandon Heller, Srini Seetharaman, Priya Mahadevan, Yiannis Yiakoumis, Puneet Sharma, Sujata Banerjee, and Nick McKeown. Elastictree: saving energy in data center networks. In *Proceedings of the 7th USENIX conference on Networked systems design and implementation*, NSDI'10, pages 17–17, Berkeley, CA, USA, 2010. USENIX Association. URL http://portal.acm.org/citation.cfm?id=1855711.1855728.

[9] László Gyarmati and Tuan Anh Trinh. How can architecture help to reduce energy consumption in data center networking? In *Proceedings of the 1st International Conference on Energy-Efficient Computing and Networking*, e-Energy '10, pages 183–186, New York, NY, USA, 2010. ACM. ISBN 978-1-4503-0042-1. doi: http://doi.acm.org/10.1145/1791314.1791343. URL http://doi.acm.org/10.1145/1791314.1791343.

[10] Gregory F Pfister. An introduction to the infiniband architecture.

[11] Nanette J Boden, Danny Cohen, Robert E Felderman, Alan E Kulawik, Charles L Seitz, Jakov N Seizovic, and Wen-King Su. Myrinet: A gigabit-per-second local area network. *IEEE micro*, 15(1):29–36, 1995.

[12] Cisco. Cisco Data Center Infrastructure 2.5 Design Guide. http://www.cisco.com/c/en/us/td/docs/solutions/Enterprise/Data_Center/DC_Infra2_5

[13] Mohammad Al-Fares, Alexander Loukissas, and Amin Vahdat. A scalable, commodity data center network architecture. *SIGCOMM*

Comput. Commun. Rev., 38:63–74, August 2008. ISSN 0146-4833. doi: http://doi.acm.org/10.1145/1402946.1402967. URL http://doi.acm.org/10.1145/1402946.1402967.

[14] Krishna Kant. Data center evolution: A tutorial on state of the art, issues, and challenges. *Computer Networks*, 53(17):2939 – 2965, 2009. ISSN 1389-1286. doi: 10.1016/j.comnet.2009.10.004. URL http://www.sciencedirect.com/science/article/pii/S1389128609003090. Virtualized Data Centers.

[15] Kashi Venkatesh Vishwanath, Albert Greenberg, and Daniel A. Reed. Modular data centers: how to design them? In *Proceedings of the 1st ACM workshop on Large-Scale system and application performance*, LSAP '09, pages 3–10, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-592-5. doi: http://doi.acm.org/10.1145/1552272.1552275. URL http://doi.acm.org/10.1145/1552272.1552275.

[16] Lucian Popa, Sylvia Ratnasamy, Gianluca Iannaccone, Arvind Krishnamurthy, and Ion Stoica. A cost comparison of datacenter network architectures. In *Proceedings of the 6th International COnference*, Co-NEXT '10, pages 16:1–16:12, New York, NY, USA, 2010. ACM. ISBN 978-1-4503-0448-1. doi: http://doi.acm.org/10.1145/1921168.1921189. URL http://doi.acm.org/10.1145/1921168.1921189.

[17] Albert Greenberg, James Hamilton, David A. Maltz, and Parveen Patel. The cost of a cloud: research problems in data center networks. *SIGCOMM Comput. Commun. Rev.*, 39:68–73, December 2008. ISSN 0146-4833. doi: http://doi.acm.org/10.1145/1496091.1496103. URL http://doi.acm.org/10.1145/1496091.1496103.

[18] Stephen Shankland. Google spotlights data center inner workings. http://www.cnet.com/news/google-spotlights-data-center-inner-workings/.

[19] Larry L. Peterson and Bruce S. Davie. *Computer Networks: A Systems Approach*. The Morgan Kaufmann Series in Networking, 2003.

[20] Radhika Niranjan Mysore, Andreas Pamboris, Nathan Farrington, Nelson Huang, Pardis Miri, Sivasankar Radhakrishnan, Vikram Subramanya, and Amin Vahdat. Portland: a scalable fault-tolerant layer 2 data center network fabric. *SIGCOMM Comput. Commun. Rev.*, 39:39–50, August 2009. ISSN 0146-4833. doi: http://doi.acm.org/10.1145/1594977.1592575. URL http://doi.acm.org/10.1145/1594977.1592575.

[21] Kai Chen, Chengchen Hu, Xin Zhang, Kai Zheng, Yan Chen, and Athanasios V Vasilakos. Survey on routing in data centers: insights and future directions. *Network, IEEE*, 25(4):6–10, 2011.

[22] Nabil Bitar, Steven Gringeri, and Tiejun J Xia. Technologies and protocols for data center and cloud networking. *Communications Magazine, IEEE*, 51 (9):24–31, 2013.

[23] Dennis Abts and Bob Felderman. A guided tour through data-center networking. *Queue*, 10(5):10, 2012.

[24] I.D. Scherson. Orthogonal graphs for the construction of a class of interconnection networks. *Parallel and Distributed Systems, IEEE Transactions on*, 2(1):3–19, 1991. ISSN 1045-9219. doi: 10.1109/71.80185.

[25] V. Cantoni, M. Ferretti, and L. Lombardi. A comparison of homogeneous hierarchical interconnection structures. *Proceedings of the IEEE*, 79(4):416–428, 1991. ISSN 0018-9219. doi: 10.1109/5.92037.

[26] L.N. Bhuyan and D.P. Agrawal. Generalized hypercube and hyperbus structures for a computer network. *Computers, IEEE Transactions on*, C-33(4): 323–333, 1984. ISSN 0018-9340. doi: 10.1109/TC.1984.1676437.

[27] S Lakshmivarahan, Jung-Sing Jwo, and S.K Dhall. Symmetry in interconnection networks based on cayley graphs of permutation groups: A survey. *Parallel Computing*, 19(4):361 – 407", 1993. ISSN 0167-8191. doi: http://dx.doi.org/10.1016/0167-8191(93)90054-O. URL http://www.sciencedirect.com/science/article/pii/016781919390054O.

[28] Chuanxiong Guo, Haitao Wu, Kun Tan, Lei Shi, Yongguang Zhang, and Songwu Lu. Dcell: a scalable and fault-tolerant network structure for data centers. In *Proceedings of the ACM SIGCOMM 2008 conference on Data communication*, SIGCOMM '08, pages 75–86, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-175-0. doi: http://doi.acm.org/10.1145/1402958.1402968. URL http://doi.acm.org/10.1145/1402958.1402968.

[29] Chuanxiong Guo, Guohan Lu, Dan Li, Haitao Wu, Xuan Zhang, Yunfeng Shi, Chen Tian, Yongguang Zhang, and Songwu Lu. Bcube: a high performance, server-centric network architecture for modular data centers. In *Proceedings of the ACM SIGCOMM 2009 conference on Data communication*, SIGCOMM '09, pages 63–74, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-594-9. doi: http://doi.acm.org/10.1145/1592568.1592577. URL http://doi.acm.org/10.1145/1592568.1592577.

[30] Cong Wang, Cuirong Wang, Ying Yuan, and Yongtao Wei. Mcube: A high performance and fault-tolerant network architecture for data centers. In *Computer Design and Applications (ICCDA), 2010 International Conference on*, volume 5, pages V5–423 –V5–427, june 2010. doi: 10.1109/ICCDA.2010.5540940.

[31] Haitao Wu, Guohan Lu, Dan Li, Chuanxiong Guo, and Yongguang Zhang. Mdcube: a high performance network structure for modular data center interconnection. In *Proceedings of the 5th international conference on Emerging networking experiments and technologies*, CoNEXT '09, pages 25–36, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-636-6. doi: 10.1145/1658939.1658943. URL http://doi.acm.org/10.1145/1658939.1658943.

[32] Albert Greenberg, James R. Hamilton, Navendu Jain, Srikanth Kandula, Changhoon Kim, Parantap Lahiri, David A. Maltz, Parveen Patel, and Sudipta Sengupta. Vl2: a scalable and flexible data center network. *Commun. ACM*, 54:95–104, March 2011. ISSN 0001-0782. doi: http://doi.acm.org/10.1145/1897852.1897877. URL http://doi.acm.org/10.1145/1897852.1897877.

[33] Meg Walraed-Sullivan, Radhika Niranjan Mysore, Malveeka Tewari, Ying Zhang, Keith Marzullo, and Amin Vahdat. Alias: Scalable, decentralized label assignment for data centers. In *Proceedings of the ACM Symposium on Cloud Computing (SOCC)*, Cascais, Portugal, October 2011.

[34] Nathan Farrington, George Porter, Sivasankar Radhakrishnan, Hamid Hajabdolali Bazzaz, Vikram Subramanya, Yeshaiahu Fainman, George Papen, and Amin Vahdat. Helios: A hybrid electrical/optical switch architecture for modular data centers. In *Proceedings of the ACM SIGCOMM 2010 Conference*, SIGCOMM '10, pages 339–350, New York, NY, USA, 2010. ACM. ISBN 978-1-4503-0201-2. doi: 10.1145/1851182.1851223. URL http://doi.acm.org/10.1145/1851182.1851223.

[35] Guohui Wang, David G. Andersen, Michael Kaminsky, Konstantina Papagiannaki, T.S. Eugene Ng, Michael Kozuch, and Michael Ryan. c-through: part-time optics in data centers. *SIGCOMM Comput. Commun. Rev.*, 41(4):–, August 2010. ISSN 0146-4833. URL http://dl.acm.org/citation.cfm?id=2043164.1851222.

[36] Eric Osborne and Ajay Simha. *Traffic Engineering with MPLS*. Cisco Press, 2003.

[37] Gerald R. Ash. *Traffic Engineering and QoS Optimization of Integrated Voice and Data Networks*. Morgan Kaufmann, 2007.

[38] Dennis Abts, Michael R. Marty, Philip M. Wells, Peter Klausler, and Hong Liu. Energy proportional datacenter networks. *SIGARCH Comput. Archit. News*, 38(3):338–347, June 2010. ISSN 0163-5964. doi: 10.1145/1816038.1816004. URL http://doi.acm.org/10.1145/1816038.1816004.

[39] Yunfei Shang, Dan Li, and Mingwei Xu. Energy-aware routing in data center network. In *Proceedings of the first ACM SIGCOMM workshop on Green networking*, Green Networking '10, pages 1–8, New York, NY, USA, 2010. ACM. ISBN 978-1-4503-0196-1. doi: 10.1145/1851290.1851292. URL http://doi.acm.org/10.1145/1851290.1851292.

[40] Supranamaya Ranjan, J Rolia, H Fu, and E Knightly. Qos-driven server migration for internet data centers. In *Quality of Service, 2002. Tenth IEEE International Workshop on*, pages 3–12. IEEE, 2002.

[41] Rodrigo N Calheiros, Rajiv Ranjan, and Rajkumar Buyya. Virtual machine provisioning based on analytical performance and qos in cloud computing environments. In *Parallel Processing (ICPP), 2011 International Conference on*, pages 295–304. IEEE, 2011.

[42] Mohammad Alizadeh, Albert Greenberg, David A. Maltz, Jitendra Padhye, Parveen Patel, Balaji Prabhakar, Sudipta Sengupta, and Murari Sridharan. Data center tcp (dctcp). *SIGCOMM Comput. Commun. Rev.*, 40(4): 63–74, aug 2010. ISSN 0146-4833. doi: 10.1145/1851275.1851192. URL http://doi.acm.org/10.1145/1851275.1851192.

[43] Christo Wilson, Hitesh Ballani, Thomas Karagiannis, and Ant Rowtron. Better never than late: Meeting deadlines in datacenter networks. *SIGCOMM Comput. Commun. Rev.*, 41(4):50–61, aug 2011. ISSN 0146-4833. doi: 10.1145/2043164.2018443. URL http://doi.acm.org/10.1145/2043164.2018443.

[44] Balajee Vamanan, Jahangir Hasan, and T.N. Vijaykumar. Deadline-aware datacenter tcp (d2tcp). In *Proceedings of the ACM SIGCOMM 2012 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, SIGCOMM '12, pages 115–126, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1419-0. doi: 10.1145/2342356.2342388. URL http://doi.acm.org/10.1145/2342356.2342388.

[45] Chi-Yao Hong, Matthew Caesar, and P. Brighten Godfrey. Finishing flows quickly with preemptive scheduling. In *Proceedings of the ACM SIGCOMM 2012 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, SIGCOMM '12, pages 127–138, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1419-0. doi: 10.1145/2342356.2342389. URL http://doi.acm.org/10.1145/2342356.2342389.

[46] Luigi Iannone and Olivier Bonaventure. On the cost of caching locator/id mappings. In *Proceedings of the 2007 ACM CoNEXT conference*, CoNEXT '07, pages 7:1–7:12, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-770-4. doi: 10.1145/1364654.1364663. URL http://doi.acm.org/10.1145/1364654.1364663.

[47] Bruno Quoitin, Luigi Iannone, Cédric de Launois, and Olivier Bonaventure. Evaluating the benefits of the locator/identifier separation. In *Proceedings of 2nd ACM/IEEE international workshop on Mobility in the evolving internet architecture*, MobiArch '07, pages 5:1–5:6, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-784-1. doi: 10.1145/1366919.1366926. URL http://doi.acm.org/10.1145/1366919.1366926.

[48] A. R. Ashok Kumar, S. V. Rao, and Diganta Goswami. 4-4, 1-4: Architecture for data center network based on ip address hierarchy for efficient routing. In *11th International Symposium on Parallel and Distributed Computing (ISPDC), 2012*, pages 235 –242, june 2012. doi: 10.1109/ISPDC.2012.39.

[49] A. R. Ashok Kumar, S. V. Rao, and Diganta Goswami. Simpler, efficient location based routing for data center network using ip address hierarchy. In *Extended version communicated to - International journal of Network Management, Wiley*, 2014.

[50] InfiniBand. InfiniBand Architecture Specification Volume 1, Release 1.0. http://www.infinibandta.org/specs.

[51] N.J. Boden, D. Cohen, R.E. Felderman, A.E. Kulawik, C.L. Seitz, J.N. Seizovic, and Wen-King Su. Myrinet: a gigabit-per-second local area network. *Micro, IEEE*, 15(1):29–36, Feb 1995. ISSN 0272-1732. doi: 10.1109/40.342015.

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

[53] Verari Systems. The verari forest container solution: The answer to consolidation. http://www.verari.com/forestspec.asp.

[54] M Mitchell Waldrop. Data center in a box. *Scientific American*, 297(2): 90–93, 2007.

[55] M. Kliegl, J. Lee, Jun Li, Xinchao Zhang, Chuanxiong Guo, and D. Rincon. Generalized dcell structure for load-balanced data center networks. In *INFOCOM IEEE Conference on Computer Communications Workshops , 2010*, pages 1–5, 2010. doi: 10.1109/INFCOMW.2010.5466647.

[56] H. Vaishnav and M. Pedram. Pcube: A performance driven placement algorithm for low power designs. In *Design Automation Conference, 1993, with EURO-VHDL '93. Proceedings EURO-DAC '93., European*, pages 72–77, Sep 1993. doi: 10.1109/EURDAC.1993.410619.

[57] Chienhua Chen and D.P. Agrawal. dbcube: a new class of hierarchical multiprocessor interconnection networks with area efficient layout. *Parallel and Distributed Systems, IEEE Transactions on*, 4(12):1332–1344, Dec 1993. ISSN 1045-9219. doi: 10.1109/71.250115.

[58] Deke Guo, Hanhua Chen, Yuan He, Hai Jin, Chao Chen, Honghui Chen, Zhen Shu, and Guangqi Huang. Kcube: A novel architecture for interconnection networks. *Information Processing Letters*, 110(1819):821–825, 2010. ISSN 0020-0190. doi: http://dx.doi.org/10.1016/j.ipl.2010.06.010. URL http://www.sciencedirect.com/science/article/pii/S0020019010001997.

[59] Dan Li, Chuanxiong Guo, Haitao Wu, Kun Tan, Yongguang Zhang, and Songwu Lu. Ficonn: Using backup port for server interconnection in data centers. In *INFOCOM 2009, IEEE*, pages 2276–2285, 2009. doi: 10.1109/INFCOM.2009.5062153.

[60] Jose Duato, Sudhakar Yalamanchili, and Lionel Ni, editors. *Interconnection Networks an Engineering Approach*. Mogran Kaufmann Publisher, 2003.

[61] C.E. Leiserson. Fat-trees: Universal networks for hardware-efficient supercomputing. *Computers, IEEE Transactions on*, C-34(10):892–901, 1985. ISSN 0018-9340. doi: 10.1109/TC.1985.6312192.

[62] Albert Greenberg, Parantap Lahiri, David A. Maltz, Parveen Patel, and Sudipta Sengupta. Towards a next generation data center architecture: scalability and commoditization. In *Proceedings of the ACM workshop on Programmable routers for extensible services of tomorrow*, PRESTO '08, pages 57–62, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-181-1. doi: http://doi.acm.org/10.1145/1397718.1397732. URL http://doi.acm.org/10.1145/1397718.1397732.

[63] Mohammad Al-Fares, Sivasankar Radhakrishnan, Barath Raghavan, Nelson Huang, and Amin Vahdat. Hedera: Dynamic flow scheduling for data center networks. In *Proceedings of the 7th USENIX conference on Networked systems design and implementation*, pages 19–19, 2010.

[64] Lin Wang, Fa Zhang, J. Arjona Aroca, AV. Vasilakos, Kai Zheng, Chenying Hou, Dan Li, and Zhiyong Liu. Greendcn: A general framework for achieving energy efficiency in data center networks. *Selected Areas in Communications, IEEE Journal on*, 32(1):4–15, January 2014. ISSN 0733-8716. doi: 10.1109/JSAC.2014.140102.

[65] Lin Wang, Fa Zhang, Athanasios V. Vasilakos, Chenying Hou, and Zhiyong Liu. Joint virtual machine assignment and traffic engineering for green data center networks. *SIGMETRICS Perform. Eval. Rev.*, 41(3):107–112, January 2014. ISSN 0163-5999. doi: 10.1145/2567529.2567560. URL http://doi.acm.org/10.1145/2567529.2567560.

[66] William Clay Moody, Jason Anderson, Kuang-Ching Wange, and Amy Apon. Reconfigurable network testbed for evaluation of datacenter topologies. In *Proceedings of the Sixth International Workshop on Data Intensive Distributed Computing*, DIDC '14, pages 11–20, New York, NY, USA, 2014. ACM. ISBN 978-1-4503-2913-2. doi: 10.1145/2608020.2608023. URL http://doi.acm.org/10.1145/2608020.2608023.

[67] Ruoyan Liu, Huaxi Gu, Xiaoshan Yu, and Xiumei Nian. Distributed flow scheduling in energy-aware data center networks. *Communications Letters,*

IEEE, 17(4):801–804, April 2013. ISSN 1089-7798. doi: 10.1109/LCOMM.2013.022213.122757.

[68] Xiaodong Wang, Yanjun Yao, Xiaorui Wang, Kefa Lu, and Qing Cao. Carpo: Correlation-aware power optimization in data center networks. In *INFOCOM, 2012 Proceedings IEEE*, pages 1125–1133, March 2012. doi: 10.1109/INFCOM.2012.6195471.

[69] Yanpei Chen, Rean Griffith, Junda Liu, Randy H. Katz, and Anthony D. Joseph. Understanding tcp incast throughput collapse in datacenter networks. In *Proceedings of the 1st ACM Workshop on Research on Enterprise Networking*, WREN '09, pages 73–82, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-443-0. doi: 10.1145/1592681.1592693. URL http://doi.acm.org/10.1145/1592681.1592693.

[70] Vijay Vasudevan, Amar Phanishayee, Hiral Shah, Elie Krevat, David G. Andersen, Gregory R. Ganger, Garth A. Gibson, and Brian Mueller. Safe and effective fine-grained tcp retransmissions for datacenter communication. *SIGCOMM Comput. Commun. Rev.*, 39(4):303–314, August 2009. ISSN 0146-4833. doi: 10.1145/1594977.1592604. URL http://doi.acm.org/10.1145/1594977.1592604.

[71] Haitao Wu, Zhenqian Feng, Chuanxiong Guo, and Yongguang Zhang. Ictcp: Incast congestion control for tcp in data-center networks. *Networking, IEEE/ACM Transactions on*, 21(2):345–358, 2013. ISSN 1063-6692. doi: 10.1109/TNET.2012.2197411.

[72] P. Devkota and A.L.N. Reddy. Performance of quantized congestion notification in tcp incast scenarios of data centers. In *Modeling, Analysis Simulation of Computer and Telecommunication Systems (MASCOTS), 2010 IEEE International Symposium on*, pages 235–243, 2010. doi: 10.1109/MASCOTS.2010.32.

[73] White paper. Extreme Networks White Paper - "Congestion Management and Buffering in Data Center Networks".

www.extremenetworks.com/libraries/whitepapers/WPCongestionManagementandBuffering_
2009.

[74] Kyriakos Zarifis, Rui Miao, Matt Calder, Ethan Katz-Bassett, Minlan Yu, and Jitendra Padhye. Dibs: Just-in-time congestion mitigation for data centers. In *Proceedings of the Ninth European Conference on Computer Systems*, EuroSys '14, pages 6:1–6:14, New York, NY, USA, 2014. ACM. ISBN 978-1-4503-2704-6. doi: 10.1145/2592798.2592806. URL http://doi.acm.org/10.1145/2592798.2592806.

[75] Jiao Zhang, Fengyuan Ren, Xin Yue, Ran Shu, and Chuang Lin. Sharing bandwidth by allocating switch buffer in data center networks. *Selected Areas in Communications, IEEE Journal on*, 32(1):39–51, January 2014. ISSN 0733-8716. doi: 10.1109/JSAC.2014.140105.

[76] Ardalan Kangarlou, Sahan Gamage, Ramana Rao Kompella, and Dongyan Xu. vsnoop: Improving tcp throughput in virtualized environments via acknowledgement offload. In *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '10, pages 1–11, Washington, DC, USA, 2010. IEEE Computer Society. ISBN 978-1-4244-7559-9. doi: 10.1109/SC.2010.57. URL http://dx.doi.org/10.1109/SC.2010.57.

[77] Sahan Gamage, Ardalan Kangarlou, Ramana Rao Kompella, and Dongyan Xu. Opportunistic flooding to improve tcp transmit performance in virtualized clouds. In *Proceedings of the 2Nd ACM Symposium on Cloud Computing*, SOCC '11, pages 24:1–24:14, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0976-9. doi: 10.1145/2038916.2038940. URL http://doi.acm.org/10.1145/2038916.2038940.

[78] Kai Chen, Chuanxiong Guo, Haitao Wu, Jing Yuan, Zhenqian Feng, Yan Chen, Songwu Lu, and Wenfei Wu. Generic and automatic address configuration for data center networks. *SIGCOMM Comput. Commun. Rev.*, 40(4): 39–50, August 2010. ISSN 0146-4833. doi: 10.1145/1851275.1851190. URL http://doi.acm.org/10.1145/1851275.1851190.

[79] D. Meyer, L. Zhang, and K. Fall. Report from the iab workshop on routing and addressing. RFC 4984 (Informational), sep 2007. URL http://www.ietf.org/rfc/rfc4984.txt.

[80] Internet Users. Internet live stats. http://www.internetlivestats.com/internet-users

[81] Dabbous Ruiz-Sanchez, Biersack. Survey and taxonomy of ip address lookup algorithms. *Network.IEEE*, 15:8–23, 2001. ISSN 0001-0782.

[82] BHUVAN URGAONKAR, ARNOLD L. ROSENBERG, and PRASHANT SHENOY. Application placement on a cluster of servers. *International Journal of Foundations of Computer Science*, 18 (05):1023–1041, 2007. doi: 10.1142/S012905410700511X. URL http://www.worldscientific.com/doi/abs/10.1142/S012905410700511X.

[83] Eddie Kohler, Robert Morris, Benjie Chen, John Jannotti, and M. Frans Kaashoek. The click modular router. *ACM Trans. Comput. Syst.*, 18(3): 263–297, August 2000. ISSN 0734-2071. doi: 10.1145/354871.354874. URL http://doi.acm.org/10.1145/354871.354874.

[84] Rui Zhang-Shen and N. McKeown. Designing a fault-tolerant network using valiant load-balancing. In *INFOCOM 2008. The 27th Conference on Computer Communications. IEEE*, pages 2360 –2368, April 2008. doi: 10.1109/INFOCOM.2008.305.

[85] NS3. Click Modular Router Integration. http://www.nsnam.org/docs/release/3.11/models/html/click.html, .

[86] Theo Benson and Aditya Akella. Data set for imc 2010 data center measurement. http://pages.cs.wisc.edu/~tbenson/IMC10_Data.html.

[87] Gustavo Carneiro, Pedro Fortuna, and Manuel Ricardo. Flowmonitor: a network monitoring framework for the network simulator 3 (ns-3). In *Proceedings of the Fourth International ICST Conference on Performance Evaluation Methodologies and Tools*, VALUETOOLS '09, pages 1:1–1:10, ICST, Brussels, Belgium, Belgium, 2009. ICST (Institute for Computer

Sciences, Social-Informatics and Telecommunications Engineering). ISBN 978-963-9799-70-7. doi: 10.4108/ICST.VALUETOOLS2009.7493. URL http://dx.doi.org/10.4108/ICST.VALUETOOLS2009.7493.

[88] Maurizio Pizzonia and Massimo Rimondini. Netkit: Easy emulation of complex networks on inexpensive hardware. In *Proceedings of the 4th International Conference on Testbeds and Research Infrastructures for the Development of Networks & Communities*, TridentCom '08, pages 7:1–7:10, ICST, Brussels, Belgium, Belgium, 2008. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering). ISBN 978-963-9799-24-0. URL http://dl.acm.org/citation.cfm?id=1390576.1390585.

[89] Marcelo Ribeiro Nascimento, Christian Esteve Rothenberg, Marcos Rogério Salvador, and Maurício Ferreira Magalhães. Quagflow: partnering quagga with openflow. *SIGCOMM Comput. Commun. Rev.*, 40:441–442, August 2010. ISSN 0146-4833. doi: http://doi.acm.org/10.1145/1851275.1851252. URL http://doi.acm.org/10.1145/1851275.1851252.

[90] Charles Lefurgy, Malcolm Allen-Ware, John Carter, Wael El-Essawy, Wes Felter, Alexandre Ferreira, Wei Huang, Anthony Hylick, Tom Keller, Karthick Rajamani, et al. Energy-efficient data centers and systems. In *Proceedings of the IEEE International Symposium on Performance Analysis of Systems and Software,Austin,Texas*, 2011.

[91] Derya Cavdar and Fatih Alagoz. A survey of research on greening data centers. In *IEEE Global Communications Conference (GLOBECOM), Anaheim, USA*, 2012.

[92] Rich Gadomski. Reducing Energy Consumption and Cost in the Data Center. http://www.datacenterknowledge.com/archives/2014/12/11/reducing-energy-consumpti

[93] Priya Mahadevan, Puneet Sharma, Sujata Banerjee, and Parthasarathy Ranganathan. A power benchmarking framework for network devices. In

Proceedings of the 8th International IFIP-TC 6 Networking Conference, NETWORKING '09, pages 795–808, Berlin, Heidelberg, 2009. Springer-Verlag. ISBN 978-3-642-01398-0. doi: 10.1007/978-3-642-01399-7_62. URL http://dx.doi.org/10.1007/978-3-642-01399-7_62.

[94] Di Xie, Ning Ding, Y. Charlie Hu, and Ramana Kompella. The only constant is change: Incorporating time-varying network reservations in data centers. In *Proceedings of the ACM SIGCOMM 2012 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, SIGCOMM '12, pages 199–210, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1419-0. doi: 10.1145/2342356.2342397. URL http://doi.acm.org/10.1145/2342356.2342397.

[95] Xiufeng Liu, Christian Thomsen, and Torben Bach Pedersen. Mapreduce-based dimensional etl made easy. *Proc. VLDB Endow.*, 5(12):1882–1885, August 2012. ISSN 2150-8097. URL http://dl.acm.org/citation.cfm?id=2367502.2367528.

[96] Xiufeng Liu, Christian Thomsen, and TorbenBach Pedersen. Etlmr: A highly scalable dimensional etl framework based on mapreduce. In Alfredo Cuzzocrea and Umeshwar Dayal, editors, *Data Warehousing and Knowledge Discovery*, volume 6862 of *LNCS*, pages 96–111. Springer, 2011. ISBN 978-3-642-23543-6. doi: 10.1007/978-3-642-23544-3_8. URL http://dx.doi.org/10.1007/978-3-642-23544-3_8.

[97] Amar Phanishayee, Elie Krevat, Vijay Vasudevan, David G. Andersen, Gregory R. Ganger, Garth A. Gibson, and Srinivasan Seshan. Measurement and analysis of tcp throughput collapse in cluster-based storage systems. In *Proceedings of the 6th USENIX Conference on File and Storage Technologies*, FAST'08, pages 12:1–12:14, Berkeley, CA, USA, 2008. USENIX Association. URL http://dl.acm.org/citation.cfm?id=1364813.1364825.

[98] Darren Erik Vengroff. Reclab: A system for ecommerce recommender research with real data, context and feedback. In *Proceedings of the 2011 Workshop on Context-awareness in Retrieval and Recommendation*, CaRR '11, pages 31–38, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0625-6. doi: 10.1145/1961634.1961641. URL http://doi.acm.org/10.1145/1961634.1961641.

[99] Diana Moise, Denis Shestakov, Gylfi Gudmundsson, and Laurent Amsaleg. Indexing and searching 100m images with map-reduce. In *Proceedings of the 3rd ACM Conference on International Conference on Multimedia Retrieval*, ICMR '13, pages 17–24, New York, NY, USA, 2013. ACM. ISBN 978-1-4503-2033-7. doi: 10.1145/2461466.2461470. URL http://doi.acm.org/10.1145/2461466.2461470.

[100] K. Namitha, A. Jayapriya, and G. Santhosh Kumar. Rainfall prediction using artificial neural network on map-reduce framework. In *Proceedings of the Third International Symposium on Women in Computing and Informatics*, WCI '15, pages 492–495, New York, NY, USA, 2015. ACM. ISBN 978-1-4503-3361-0. doi: 10.1145/2791405.2791468. URL http://doi.acm.org/10.1145/2791405.2791468.

[101] Gösta Grahne, Shahab Harrafi, Iraj Hedayati, and Ali Moallemi. Dfa minimization in map-reduce. In *Proceedings of the 3rd ACM SIGMOD Workshop on Algorithms and Systems for MapReduce and Beyond*, BeyondMR '16, pages 4:1–4:10, New York, NY, USA, 2016. ACM. ISBN 978-1-4503-4311-4. doi: 10.1145/2926534.2926537. URL http://doi.acm.org/10.1145/2926534.2926537.

[102] EC2-Spot. EC2 Spot Instance. http://aws.amazon.com/ec2/spot-instances/.

[103] xStream. xStream. http://www.virtustream.com/.

[104] Ron Kohavi, Randal M. Henne, and Dan Sommerfield. Practical guide to controlled experiments on the web: Listen to your customers not to the

hippo. In *Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '07, pages 959–967, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-609-7. doi: 10.1145/1281192.1281295. URL http://doi.acm.org/10.1145/1281192.1281295.

[105] Emmanuel Cecchet, Julie Marguerite, and Willy Zwaenepoel. Performance and scalability of ejb applications. *SIGPLAN Not.*, 37(11):246–261, nov 2002. ISSN 0362-1340. doi: 10.1145/583854.582443. URL http://doi.acm.org/10.1145/583854.582443.

[106] Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall, and Werner Vogels. Dynamo: Amazon's highly available key-value store. *SIGOPS Oper. Syst. Rev.*, 41(6):205–220, oct 2007. ISSN 0163-5980. doi: 10.1145/1323293.1294281. URL http://doi.acm.org/10.1145/1323293.1294281.

[107] B.D. Goodman and R.S.H. Zhou. Bridging to the cloud: Solution design trends helping "legacy" systems leverage cloud computing. In *Computational Science and Engineering (CSE), 2010 IEEE 13th International Conference on*, pages 433–438, 2010. doi: 10.1109/CSE.2010.62.

[108] facebook. How many users on facebook. http://www.benphoster.com/facebook-user-growth-chart-2004-2010.

[109] Todd Hoff. Latency is everywhere and it costs you saleshow to crush it. High Scalability, July, 25, 2009.

[110] Rajeev Sivaram, C.B. Stunkel, and D.K. Panda. Hipiqs: a high-performance switch architecture using input queuing. In *Parallel Processing Symposium, 1998. IPPS/SPDP 1998. Proceedings of the First Merged International ... and Symposium on Parallel and Distributed Processing 1998*, pages 134–143, Mar 1998. doi: 10.1109/IPPS.1998.669901.

[111] Deng Pan and Yuanyuan Yang. Fifo-based multicast scheduling algorithm for virtual output queued packet switches. *Computers, IEEE Transactions on*, 54(10):1283–1297, Oct 2005. ISSN 0018-9340. doi: 10.1109/TC.2005.164.

[112] C. Metz, C. Barth, and C. Filsfils. Beyond mpls ... less is more. *Internet Computing, IEEE*, 11(5):72–76, 2007. ISSN 1089-7801. doi: 10.1109/MIC.2007.100.

[113] F. Thomson Leighton. *Introduction to Parallel Algorithms and Architectures: Array, Trees, Hypercubes*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1992. ISBN 1-55860-117-1.

[114] NS3. Valgrind to debug memory problems. http://www.nsnam.org/wiki/index.php/HOWTO_use_Valgrind_to_debug_memory_problems, .

[115] Kcachegrind. CPU profiling using Kcachegrind. http://kcachegrind.sourceforge.net/html/Home.html.

Publications out of the work

Journals

1) **A. R. Ashok Kumar**, S. V. Rao, and Diganta Goswami. Simpler, Efficient Location Based Routing for Data Center Network using IP address Hierarchy. ***Communicated to*** Internatinal journal of Network Management.

Conferences/Workshops

1) **A. R. Ashok Kumar**, S. V. Rao, and Diganta Goswami.Greening 4-4,1-4 data center network: A greedy approach for finding an energy efficient sub-network- ***accepted for*** Emerging Issues in Cloud (EIC) workshop on hot topics in cloud computing in conjunction with 6th IEEE international conference on cloud computing technology and science to be held at Singapore.

2) **A. R. Ashok Kumar**, S. V. Rao, and Diganta Goswami. Network Simulator 3 for a study of Data Center Networks: In 12th International Symposium on Parallel and Distributed Computing (ISPDC), 2013, pages= 224-231, Romania, IEEE, June 2013

3) **A. R. Ashok Kumar**, S. V. Rao, and Diganta Goswami. BCube-IP - BCube with IP address hierarchy for efficient routing: In International Conference on Advanced Computing, Networking, and Informatics (ICACNI - 2014),319-326, Springer LNCS, Raipur, India, June - 2013

4) **A. R. Ashok Kumar**, S. V. Rao, and Diganta Goswami. 4-4, 1-4: Architecture for data center network based on IP address hierarchy for efficient routing. In 11th International Symposium on Parallel and Distributed Computing (ISPDC), 2012, pages 235 242, IEEE, Germany, june 2012.

5) **A. R. Ashok Kumar**, S. V. Rao, Diganta Goswami, and G. Sahukari. DCell-IP: DCell emboldened with IP address hierarchy for efficient routing. In Proceedings of International Conference on Advances in Computing, volume 174 of Advances in Intelligent Systems and Computing, pages 739746. Springer India, 2013.

6) **A. R. Ashok Kumar**, S. V. Rao, Diganta Goswami. New architecture for Data Center Network. In Proceedings of the Second Workshop on Issues in Virtualization and Cloud Computing WIVCC 2012, IIT Bombay, February 2012.