Network topologies for cost reduction and QoS improvement in massive data centers

Zina Chkirbene

UNIVERSITY OF BURGUNDY
U.F.R. SCIENCES ET TECHNIQUE
ECOLE DOCTORAL SPIM

THESIS

Presented by:

Zina Chkirbene

Submitted in fulfilment of the requirements
for the degree of:

DOCTOR OF THE UNIVERSITY OF BURGUNDY

NETWORK TOPOLOGIES FOR COST REDUCTION AND QOS IMPROVEMENT IN MASSIVE DATA CENTERS

TOPOLOGIES RÉSEAU POUR LA RÉDUCTION DES COÛTS ET L'AMÉLIORATION DE LA QUALITÉ DU SERVICE DANS LES CENTRES DE DONNÉES MASSIVES.

Defended on 29-06-2017

Jury :

M. Ahmed LBATH, Professeur, Université Grenoble Alpes, Rapporteur

Mme. Salima BENBERNOU, Professeur, Université Paris Descartes, Rapporteur

M. Olivier TOGNI, Professeur, Université de Bourgogne, Examinateur

Mme. Tara Ali -YAHIYA, Maître de conférences HDR, Université Paris Sud 11, Examinateur

M. Sebti FOUFOU, Professeur, Université de Bourgogne, Directeur de thèse

M. Ridha HAMILA, Professeur, Université de Qatar, Co-encadrant

Acknowledgements

I would like to express my sincere gratitude to Prof. Sebti Foufou for the continuous support of my Ph.D study and related research, for his patience, motivation, and immense knowledge. His guidance helped me in all the time of research and writing of this thesis.

I would like to express my very special gratitude to Prof. Ridha Hamila for being always available to help me with any necessary technical information thanks to his vast knowledge and skills in many areas. I am very thankful to Dr. Rachid Hadjidj who was abundantly helpful and offered invaluable assistance, support and guidance.

Prof. Salima Benbernou and Prof. Ahmed Lbath have made me honored by reviewing my thesis and I would like to thank them for their time and efforts taken to review this report. I would like also to thank Prof. Olivier Togni and Dr. Tara Ali-Yahiya for being members of my Jury.

I would like to show my gratitude to my dear parents Sadok and Chwikha, my parents-in-law Mohamed and Hayet, my sister Mariouma, my brother Youx and my two best friends Emna and Abir for their love and kindness.

Special words for my husband Ala and my daughter Line. Thank you for the support and company during late nights of work.

Abstract

Data centers (DC) are being built around the world to provide various cloud computing services. One of the fundamental challenges of existing DC is to design a network that interconnects massive number of nodes (servers)[1] while reducing DC' cost and energy consumption. Several solutions have been proposed (e.g. FatTree, DCell and BCube), but they either scale too fast (i.e., double exponentially) or too slow. Efficient DC topologies should incorporate high scalability, low latency, low Average Path Length (APL), high Aggregated Bottleneck Throughput (ABT) and low cost and energy consumption. Therefore, in this dissertation, different solutions have been proposed to overcome these problems. First, we propose a novel DC topology called LCT (Linked Cluster Topology) as a new solution for building scalable and cost effective DC networking infrastructures. The proposed topology reduces the number of redundant connections between clusters of nodes, while increasing the numbers of nodes without affecting the network bisection bandwidth. Furthermore, in order to reduce the DCs cost and energy consumption, we propose first a new static energy saving topology called VacoNet (Variable Connection Network) that connects the needed number of servers while reducing the unused materials (cables, switches). Also, we propose a new approach that exploits the correlation in time of internode communication and some topological features to maximize energy saving without too much impacting the average path length.

keywords: Cloud computing services, data center network, average path length, energy consumption, infrastructure cost.

[1] In this document we will use the words "node" and "server" interchangeably.

Résumé

L'expansion des services en ligne, l'avènement du big data, favorisé par l'internet des objets et les terminaux mobiles, a entraîné une croissance exponentielle du nombre de centres de données qui fournissent des divers services de cloud computing. Par conséquent, la topologie du centre de données est considérée comme un facteur d'influence sur la performance du centre de données. En effet, les topologies des centres de données devraient offrir une latence faible, une longueur de chemin moyenne réduite avec une bande passante élevée. Ces exigences augmentent la consommation énergétique dans les centres de données. Dans cette dissertation, différentes solutions ont été proposées pour surmonter ces problèmes. Tout d'abord, nous proposons une nouvelle topologie appelée LCT (Linked Cluster Topology) qui augmente le nombre de nœuds, améliore la connexion réseau et optimise le routage des données pour avoir une faible latence réseau. Une nouvelle topologie appelée VacoNet (Variable connexion Network) a été également présentée. VacoNet offre un nouveau algorithme qui définit le exact nombre de port par commutateur pour connecter le nombre de serveurs requis tout en réduisant l'énergie consommée et les matériaux inutilisés (câbles, commutateurs). En outre, nous étudions une nouvelle technique pour optimiser la consumation d'énergie aux centres de données. Cette technique peut périodiquement estimer la matrice de trafic et gérer l'état des ports de serveurs tout en maintenant le centre de données entièrement connecté. La technique proposée prend en considération le trafic réseau dans la décision de gestion des ports.

Mot clés: centre de données, topologies, cloud computing, consommation d'énergie, le coût de l'infrastructure

Contents

List of Figures

List of Tables

Acronyms

DC	**D**ata **C**enter
DCN	**D**ata **C**enter **N**etworks
APL	**A**verage **P**ath **L**ength
ABT	**A**ggregate **B**ottleneck **T**hroughput
QoS	**Q**uality **o**f **S**ervice
MDCs	**M**odular **D**ata **C**enter
AWS	**A**mazon **W**eb **S**ervice
SDN	**S**oftware **D**efined **N**etworking
FPGA	**F**ield **P**rogrammable **G**ate **A**rray
PCPG	**P**er **C**ore **P**ower **G**ating
DVFS	**D**ynamic **V**oltage and **F**requency **S**caling
SV	**S**erver **V**irtualisation
MA	**M**oving **A**verage
ToR	**T**op **o**f **R**ack
VLB	**V**aliant **L**oad **B**alancing
NIC	**N**etwork **I**nterface **C**ard
DOS	**D**ata center **O**ptical **S**witch
AWGR	**A**rrayed **W**aveguide **G**rating **R**outer

Introduction

Data centers are being built around the world to provide various cloud computing services including search (e.g., Google, Bing), video content hosting and distribution (e.g., YouTube, Netflix), social networking (e.g., Facebook, Twitter), and large-scale computations (e.g., data mining, indexing) [1; 2; 3; 4]. Microsoft, IBM, Google, Amazon, Yahoo and eBay are running Data Centers with at least 50,000 nodes for each one of them [5; 6]. Consequently, the Data Centers infrastructure must be well designed to maintain the consumed energy and the cost of both deployment and maintenance at an acceptable level [7]. In addition, data availability and scalability are considered as critical criteria in the design of a Data Center topology because of their big impact on the infrastructure cost. Hence, the topology of the Data Center is regarded as the most significant factor, since it does not only determine the reliability of a Data Centers, but also plays a control role in network capacity, fault tolerance, latency, cost and routing efficiency.

0.1 Problem statement

Data Centers topologies should provide high scalability, low latency, small average path length, low cost and high bisection bandwidth. However, existing topologies as FatTree [8], FiConn [9], DCell [10], BCube [11], and SprintNet [12] have some limitations, as they either scale too fast (i.e. double exponentially) or too slow; they suffer from performance bottlenecks, and they are costly to implement. Moreover, the energy consumption is a critical problem in Data Centers [13]. In 2010 the total energy consumed by Data Centers around the world amounts for 1.5% of the global electrical power consumption. According to [14], Data Centers' energy consumption was estimated to be about 120 billion

Kilowatts in 2012, which is about 2.8% of the total electricity bill in the USA. Also, the use of traditional routing algorithms in some topologies increases the energy consumption in Data Centers. Basically, traditional routing algorithms forward packets to destinations without taking into account energy consumption [15]. Since only a subset of the network infrastructure (switches and links) is involved when forwarding data packets to their destinations, a significant amount of energy can be saved if only involved network resources are turned on while the other are put in sleep mode or turned off altogether.

0.2 Contributions

The main purpose of this work is to propose new scalable and cost-effective Data Centers networking infrastructures that combines the advantages of existing topologies while avoiding their limitations and reducing the Data Centers' cost and energy consumption.

Our first contribution is to propose a new Data Center topology, called LCT (Linked Cluster Topology) that combines the advantages of previous topologies while avoiding their limitations. The proposed topology uses a small node degree that matches the physical restriction for servers. Furthermore, LCT interconnects a large number of servers while reducing the wiring complexity. This strategy increases the number of directly connected clusters per layer and avoids redundant cluster connections. As a result, we get a good quality of nodes in terms of bisection bandwidth and aggregated bottleneck throughput. LCT forwards packets between nodes using a new hierarchical row-based routing algorithm. Based on the modular difference between the source and destination coordinates, the algorithm constructs the route to the source.

The second contribution is the design of a new Data Center networking topology called VacoNet (Variable Connection Network) that reduces the cost, the energy consumption and the APL compared to the existing topologies. VacoNet inspires its connection from LCT topology to avoid the redundant clusters connection towards a low latency and APL. In addition, we propose a new approach to reduce energy consumption in Data Centers for better performance. By exploiting the correlation in time of the network traffic, the proposed approach uses

the traffic matrix of the current network state, and manages the state of switch ports (on/off) at the beginning of each period, while making sure to keep the Data Center fully connected. During the rest of each time period, the network must be able to forward its traffic through the active ports.

0.3 Outline

This work will be presented through three chapters:

- In the first chapter, we give an overview of Data Centers. We also introduce the hardware used in DCN's and present the energy saving aspects in green Data Centers. Then, we analyze the topologies designs of DCN's from various aspects.
- In the second chapter, we present the LCT topology and its new physical structure and routing algorithms to interconnect nodes and transmit data.
- In the third chapter, we present the proposed dynamic and static approaches for green Data Centers.
- Chapter four gives the general conclusion and presents few ideas of future extensions of this work.

Chapter 1

Literature Review

1.1 Introduction

In this chapter, we give an overview about DCs and provide both a qualitative and quantitative analysis of their features. In this context, we present a performance comparisons between typical topologies designs, connectivity discussion on average degree, bandwidth calculation, and diameter estimation, as well as capacity enhancement of DCs. This chapter is organized as follows: In section 1.2, we present an overview about DCs and study existing techniques for green DCs in section 1.3. In section 1.4, We start with a discussion about various representative DC topologies, then compare them in section 1.5 from different perspectives to highlight the advantages and disadvantages of each topology.

1.2 An overview of DCs

A DC is regarded as a physical centralized repository for computation, storage, management, and dissemination of information and data. A typical DC consists of computers, switches, racks of servers. In this section, we present a detailed description of DC hardware and give some examples of DC.

1.2.1 Hardware of DC networks

- **Switches**
  Switches are the backbone of DCs' implementations. Generally, six types of switches are used in cloud DCs. Four categories of core switches, namely: Cisco Nexus 7000 Series DC switches [16], Huawei Cloud Engine 12 800 Series switches [17], Ruijie RG-N18000 Series switches [18], and Arista 7500E Series switches [19], and categories of Top Of Rack (ToR) switches, namely: Cisco Nexus 3064 Series [20] and Arista 7050QX Series switches [21]. Core switches are characterized by their high performance compared with the ToR switches. On the other hand, optical switches have gained great attention in recent years. An optical switch enables signals in optical fibers or integrated optical circuits to be selectively switched from one circuit to another.

- **Servers**
  Servers (nodes) are the core physical components of in DC network (DCN). Servers analyze, store, process and send massive data. They directly determine the performance of Data Centes. Servers can be divided into three categories: tower servers, rack servers (ThinkServer RD630 [22], IBM System x3650 M4 Server [23]), and blade servers (Huawei Tecal BH640 V2 Blade Server [24] and PowerEdge M820 Blade Server [25]).

- **Racks**
  Racks are essential in DCNs since they facilitate the management of DCs and allow to save space. Racks support switches, servers and storage devices and can be divided in two types: cabinet and open racks which are easy to install, configure, and manage. In contrast to open racks, cabinets are more secure and stable.

- **Cables**
  Cables interconnect components (switches, servers and storage devices) and transport electrical or optical signals. Cables are generally categorized as copper and fiber according to the medium. It is crucial to choose proper cables for different applications. The considerations include: the required

useful life of cables, the DC size, the cabling system capacity, and recommendations or specifications of vendors' equipment.

1.2.2 Example of DCs

Large IT companies construct several production DCs to support their business.

- **Google** owns 36 production DCs globally, 19 of which are in America, 12 in Europe, 3 in Asia, 1 in Russia, and 1 in South America [26]. These DCs support Google services, such as searching, Gmail, and Google Maps. In 2016-2017, Google DCs will be constructed in Oregon USA, Tokyo Japan and other ten countries and regions.

- **Microsoft** owns production DCs in America, Europe and Asia [27]. It built Washington Quincy DC on an area of 75 acres in 2007. Quincy Modular DC was online in 2011, covers 93,023 square feet and utilizes green technologies. In late 2013, Microsoft approved a corporate budget of 11 million to purchase a 200 acre land in Quincy to build a large-scale DCs.

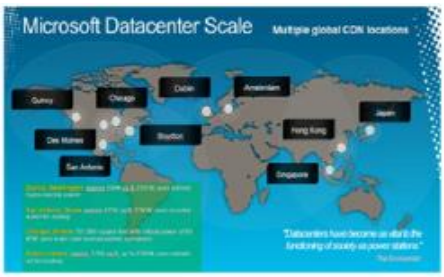
- **Dublin DC** [28] is the biggest oversea DCs of Microsoft. It covers 303,000 square feet, and achieves cooling by natural wind for saving energy. It was expanded with a new 112,000 square feet to place modular DCs (MDCs). Boydton MDC with 316,300 square feet can quickly meet customer demands for cloud services [28].

- **Amazon** owns DCs globally, which not only support e-commerce businesses, but also the services for worldwide enterprises, governments, and startup companies by Amazon Web Service (AWS) [29].

Figure 1.1 shows the maps of Microsoft, Google and Amazon DCs.

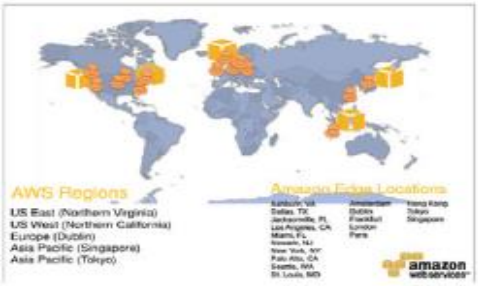
1.3 Green DCs

Energy consumption is becoming a serious issue for DCs as they grow bigger and bigger. Many researches have been recently conducted to tackle the issue



(a) Map of Microsoft DCs.

(b) Map of Google DCs.



(c) Map of Amazon DCs.

Figure 1.1: Maps of Microsoft, Google and Amazon DCs.

of energy saving in DCs, and can classify in to two categories: Dynamic energy saving approaches and Static energy saving approaches.

1.3.1 Dynamic energy saving approach

In this category, several topologies have been proposed, such as: Merge Networks [30], Elastic Tree [31], Energy-aware Routing Model[32].

- Merge Networks was proposed in [30]. Its aim it to reduce switch power consumption by combining N low traffic links into K high traffic links ($K < N$), and powering off remaining ports or putting them in low power mode.

- Authors in [31] proposed Elastic Tree. The key idea is to find the minimum-power network subsets and shutting down the unused network elements. This approach used three modules: optimizer, routing and power control. The optimizer aims to find the minimum network subset to satisfy all traffic. The routing module calculates the paths of flows. The power control module manages the states of network devices. Although this approach improves energy saving, it requires complete knowledge of the traffic matrix at each instant t.

- Energy-aware Routing Model is proposed in [32]. By using few switches and a predefined throughput threshold, this approach tries to satisfy a given traffic matrix while reducing energy consumption. To do that, a basic routing and basic throughput have to be computed by taking into consideration all the possible switches. After that, the routing is recomputed and switches eliminated until the throughput reaches the predefined threshold. The final routing will be used, while switches not involved in the routing are powered off for more energy saving.

  Energy-aware Routing model takes several seconds to calculate a non-optimal power-aware routing paths for thousands of flows. It takes even hours to calculate a near optimal solution, which has a big impact on computation efficiency and latency.

- Willow [33] is a flow scheduling algorithm for energy saving in network-limited flows DCs. The key contribution is to consider both the number of used switches and the active running duration of switches for network energy saving, then use an SDN based approach to schedule the flows. However, there are still many critical issues that still need to be addressed, such as the computing complexity, the impact on network reliability, and the impact on network performance caused by powering off devices.

1.3.2 Static energy saving approach

- A better management of data storage in DCs increases energy saving. So, the idea is to use less storage to reduce energy consumption by using new

storage resource management tools such as Automated Storage Provisioning, Data Compression and RAID Level [34].

- The use of renewable sources of energy such as wind, water, solar energy and heat pumps, reduces energy consumption. GreenHadoop [35], GreenStar Network Testbed [36], and Net-Zero Energy DCs [37] are using green and renewable sources of energy. However, these topologies are very costly, in addition to the many considerations that need to be taken into account, such as: the climate, the location of the DCs and weather conditions.

- Purchasing More Energy-Efficient hardware is another solution for energy saving. Basically, new servers use more efficient hardware as power supplies, better DC voltage regulators, processors that consume less power, and cooling fans that are more energy-efficient. Some contributions in this context are: Low Energy Switch Block for FPGAs [38], Powernap [39], Energy Management for Commercial Servers [40], Thread Motion [41], PCPG (per-core power gating) [42], and Memory Power Management via DVFS [43].

- AdyNet[44], Proteus [45], Pcube [46] are DCs network architectures designed with energy efficiency in mind. These alternative approaches are attractive, but still need to make their proofs in real settings.

- Server virtualization (SV) was proposed to consolidate servers and reduce energy consumption by running multiple different workloads on one physical host server. GreenCloud [47], TRP/VCS [48] and VPTCA [49] leverage such technology. Virtualization combines the processing power onto some servers that operate at higher total utilization rates, instead of operating many servers at lower utilization rates. Servers in SV are assigned depending on the characteristics of the applications and the network topology to improve the traffic. However, at the same time this technology brings about some additional costs induced by migrating VMs over the long-term, such as the extra time to complete the migration and the large amount of generated traffic between source and destination servers during VM migration, which is very bandwidth greedy.

1.4 Topologies of DC networks

1.4.1 Fixed topologies: Tree-based Topologies

Tree-based topologies use intelligent switches for a smart routing of packets in a DC. Some DCs topologies in this category are VL2 [50] and FatTree [8].

1.4.1.1 FatTree

FatTree is a fixed topology defined as extension of tree topology (Figure 1.2). Each n-port switch in the edge tier is connected to $\frac{n}{2}$ servers, and it has $\frac{n}{2}$ aggregated switches. The $\frac{n}{2}$ aggregation-level switches, the $\frac{n}{2}$ edge-level switches, and the servers are connected to the edge switches form a basic cell of a fat tree, which is called a *pod*. There are $(n/2)^2$ n-port switches and this topology is simple to implement. Unlike tree topologies, all the three levels use the same type of switches. High-performance switches are not necessary in the aggregate and core levels. However, the number of servers in FatTree is limited by the number of switch ports.

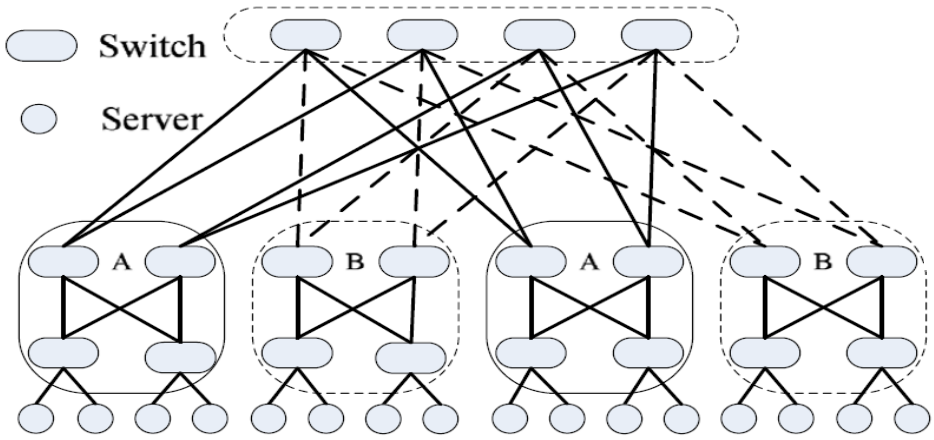


Figure 1.2: A Fat-Tree structure

1.4.1.2 VL2

VL2 overcomes some of the critical issues in conventional DCs (e.g. over-

subscription, agility and fault tolerance) by exploiting a uniform high capacity from server to server. Furthermore, it supports virtual machine migration from server to server without breaking the TCP connection and keeping the same address. Hence, VL2 topology enhances the availability and reliability of the network, especially in the presence of link or hardware failures. VL2 however uses valiant load balancing (VLB), which randomly selects an intermediate switch before forwarding a packet. This was found to be impractical in the case where two hosts, connected to the same edge switch, want to communicate.

1.4.2 Fixed topologies: Recursive Topologies

Several topologies are using parallelism to interconnect servers in date center (e.g. as DCell [10], BCube [11], HyperBcube[51], FiConn [9] and Portland [52]).

1.4.2.1 DCell

DCell is a recursive structure whose basic element called DCell_0. Each server in a DCell_0 is connected to the switch in the same DCell_0. In a DCell_k, each server will eventually have $k+1$ links: the first link (or level_0 link) is connected to a switch when forming a DCell_0, and level_i link is connected to a server in the same DCell_i (Figure 1.3). Most of DCell servers act as routers: they are equipped with multiple interface cards (NICs), and only computational servers are considered as routers. As a result, DCell topology scales double exponentially because of additional and lengthy wiring communication links between switches and servers.

1.4.2.2 BCube

BCube is a server-centric network structure, where a BCube_1 is constructed from n BCube_0 and n-port switches. It makes use of more switches when constructing a higher level topology. It requires n switches to construct a BCube_1 and connects one server in each BCube_0. Hence, a BCube_1 contains n BCube_0 and n extra switches (Figure 1.4). Thus, a BCube_k is built from n BCube_{k-1} and n^k extra n-port switches. These extra switches are connected to exactly one server in each BCube_{k-1}. BCube requires more switches when constructing higher level

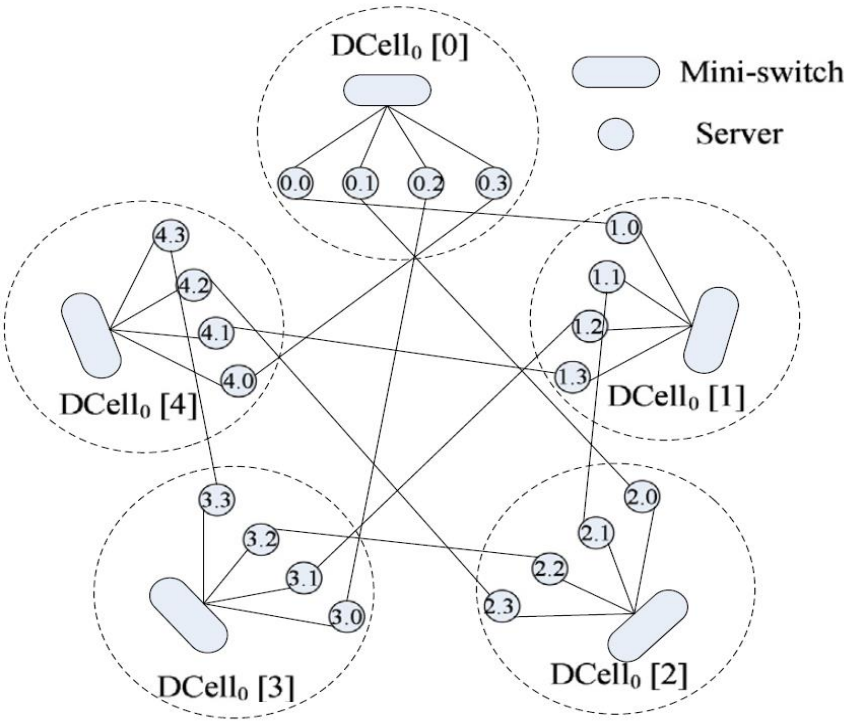


Figure 1.3: A DCell structure

structures, and DCell uses only level$_0$ n-port switches. However, both require servers to have (k+1) NICs. The implication is that servers will be involved in switching more packets in DCell than in BCube.

1.4.2.3 FiConn

FiConn is a recursive structure: a high-level FiConn is built using low-level FiConn/s. FiConn uses only the existing backup port on each server for interconnection, and no other hardware cost is introduced. This topology provides improvements to FatTree. First, it uses the interconnection intelligence on servers rather than on switches, and hence it reduces the number of used switches (Figure 1.5). Indeed, if we denote by N the total number of servers connected using n-port switches, then the number of switches needed in FatTree is $\frac{5N}{n}$ (2 edges, 2

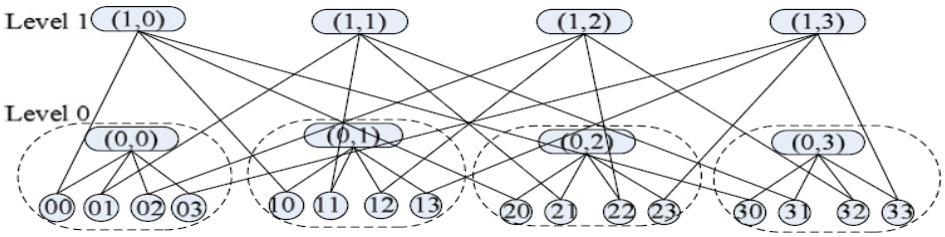


Figure 1.4: A BCube structure.

aggregated and 1 core for each pod), while FiConn requires only $\frac{N}{n}$ switches.

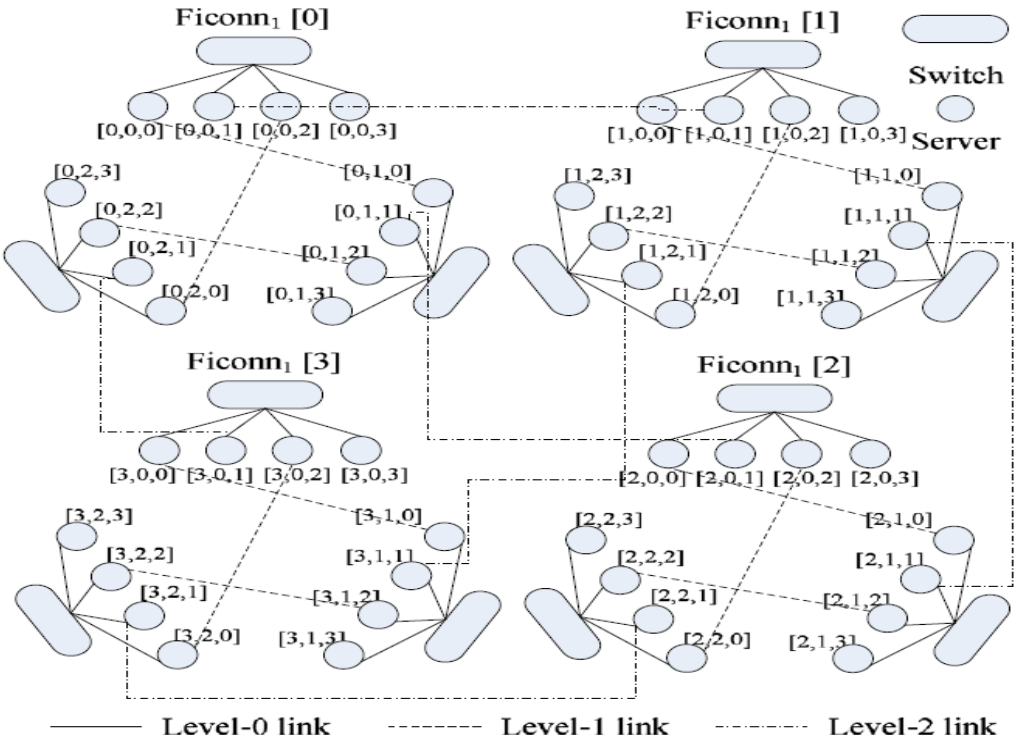


Figure 1.5: Ficonn structure

1.4.2.4 FlatNet

FlatNet is recursive topology. The first layer of the FlatNet contains n servers and one n-port switch and the second layer consists of n^2 1-layer FlatNet. A two layers FlatNet can be considered as an $n^2 * n$ matrix so it can be regarded as having n columns where each column contains exactly n^2 servers which belong to n^2 1-layer FlatNet. A column-based connection is used to connect the n^2 servers located at the same column by using exactly n n-port switches. Hence, every n servers (denoted by cluster) are directly connected to an "external" server and these clusters are connected using the connection pattern proposed in [53]. Figure 1.6 shows an example of FlatNet network for n=4.

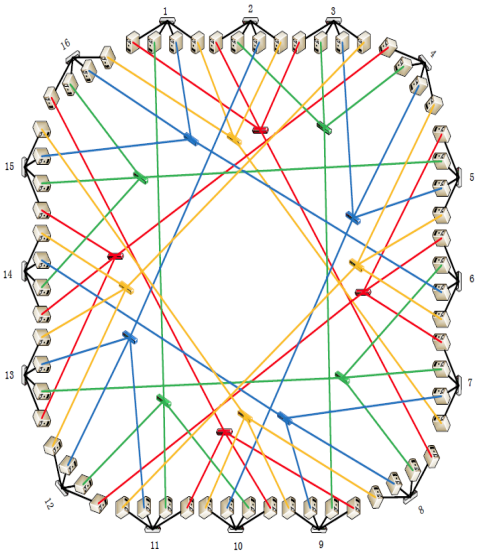


Figure 1.6: A FlatNet network for n=4

1.4.2.5 HyperBcube

HyperBcube is a recursive topology [51]. The first layer of the HyperBcube contains n servers and one n-port switch. Starting from the second layer ($k \geq 2$), HyperBcube can be considered as an $n^2 * n^{(2\times k-3)}$ matrix having $n^{(2\times k-3)}$ columns, where each column contains exactly n^2 servers which belong to a n^2 (k-1)-layer HyperBcube. A column-based connection is used to connect the n^2

servers located at the same column by using exactly n n-port switches. However, the connection pattern in HyperBcube is inefficient since it results in redundant cluster connections. In fact, if two clusters do not have any intermediate switch, 8 hops are needed to connect servers in these clusters.

1.4.3 Flexible topologies

1.4.3.1 DOS

DC Optical Switch (DOS) [54] is based on an all-optical switching fabric called Arrayed Waveguide Grating Router (AWGR). AWGR allows different inputs to reach the same output simultaneously by using different wavelengths. This characteristic allows DOS to outperform existing DC interconnects in terms of the bandwidth and the size of the switching fabric compared to electronic switches.

1.4.3.2 c-Through

HyPaC also called C-Through [55] is a hybrid network topology that makes use of both electrical packet switching network and optical circuit switching network [55]. It is composed of two parts: the first part is a tree-based electrical network which maintains connectivity between each pair of top of rack (ToR) switches; the second part is a reconfigurable optical network which offers high bandwidth interconnection between certain racks. Both HyPaC and DOS are optical circuit-switched networks. Although, they have the advantages of high bandwidth and low cost, they suffer from considerable reconfiguration time. In fact, the scheduling algorithms have to be well designed to avoid frequent circuit reconfiguration and minimize the reconfiguration time. In addition, high-speed packet buffers need to be designed to support large capacity, multiple queues and provide short response times, so as to accommodate packets temporarily and avoid unnecessary drops during the period of reconfiguration, which are as difficult as designing the DC interconnection network itself. Figure 1.7 shows HyPaC structure.

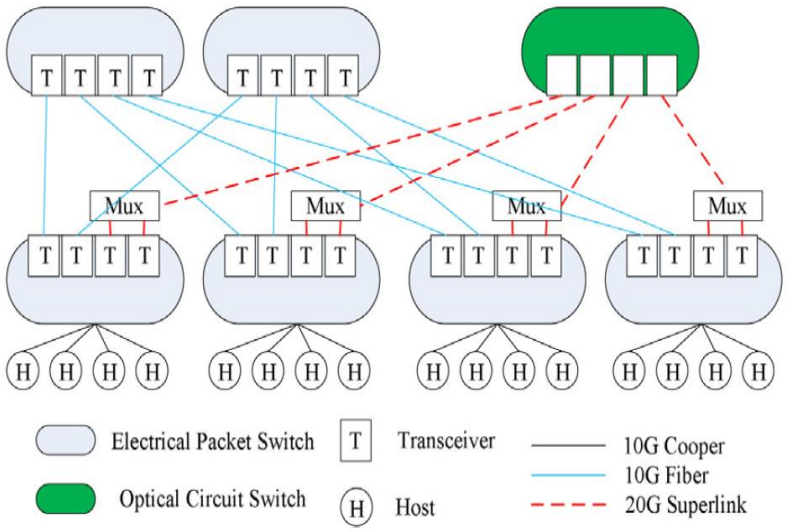


Figure 1.7: A HyPaC structure.

1.5 Comparisons of topologies

1.5.1 Comparison criteria

- **Degree of the servers**: It is the number of network ports on the servers in the DC. For the tree based topologies, only one port is needed on each server. However, in the recursive topologies, the number of ports varies according to the levels required.
- **Scalability**: It is the number of servers in a network. In order to meet the increasing demands for services and better performance, the physical structure must have good scalability enabling incremental expansion without affecting the existing servers.
- **Diameter**: Given the shortest distances between all pairs of nodes, the diameter is defined as the maximum of these distances. A smaller diameter leads to more effective routing, and lower transmission latency in practice.
- **Latency**: It consists of queuing/buffering delay at each hop, propagation delay and transmission delay.

- **Network capacity**: DC should provide high network capacity to support the high volumes of traffic generated by many online infrastructure services.
- **Fault tolerance**: A fault-tolerant architecture allows the system to continue with its current task even in the presence of failures.
- **The aggregate bottleneck throughput**: It measures the overall network capacity under the all-to-all traffic pattern, where every server connects with all other servers.
- **Bisection bandwidth**: It is the minimum number of links cut when a network is partitioned into two equal halves over all partitions.
- **Average Path length**: It measures the efficiency of packet transmission in a network. Hence, this is considered as one of the most important metric to evaluate network topologies.
- **Bandwidth**: It is used to characterize data transfer rate, i.e. the amount of data that can be carried from one point to another. There are four types of data bandwidths that can occur under different traffic patterns:
  - One-to-One bandwidth: Represents the maximum bandwidth that the topology offers when one arbitrary node sends data to another arbitrary node.
  - One-to-All bandwidth: Occurs when updating some software on all nodes.
  - One-to-Several bandwidth: Occurs when the file system is making replicas.
- **The cost**: It includes hardware cost (server, switch racks) and energy cost.
- **Energy consumption (power consumption)**: It has a high importance for DCs. [7].

1.5.2 Performance comparison

Some quantitative structural properties of existing topologies are presented in Table 1.1 and Table 1.2. Table 1.3 shows a classification of some existing topologies based on their scalability where k is the number of ports per node, n is the number of ports per switch, and c is an arbitrary constant associated with some hardware limitations. Table 1.3 reveals that topologies with a scalability $O(c)$ and $O(n^c)$ have physical limitations such as the size of the optical switching fabric and the port count per switch. In fact, VL2 (even with a three-layer network) can only connect $\frac{n^3}{4}$ nodes, which is an insufficient number of nodes for a large-scale DC. Moreover, DCell and BCube provide good scalability. However, DCell has a high wiring complexity and BCube requires more than three layers to scale up to a large size. For instance, with a 4-port switch, we need five layers to build a DC with $4^5 = 1024$ nodes. A 5-layer BCube network needs five interface cards per node, which is obviously expensive and difficult to manage in practice. Hence, BCube has scalability issues when employing cost-effective small degree node and small-port-count switches. Table 1.4 shows also that both DCell and HyperBcube provide bigger number of nodes than BCube and Tree based topologies.

	VL2	FatTree
Nodes Number	$\frac{(n-2)n^2}{4}$	$\frac{n^3}{4}$
Link Number	$\frac{(n+2)n^2}{4}$	$3\frac{n^3}{4}$
Per Node	$\frac{(n+2)}{(n-2)}$	3
Switches Number	$\frac{3n}{2} + \frac{n^2}{4}$	$5\frac{n^2}{4}$
Per Node	$\frac{(n+6)}{(6n-2n)}$	$\frac{5}{n}$
Network Diameter	6	6

Table 1.1: Comparison of topologies supporting 2 layers only.

A DC network consists of switches, nodes and links [56]. As shown in Figure 1.8, there are three types of links in a DC network: α (linking two nodes), β (linking a node and a switch) and γ (linking two switches).

	HyperBcube	DCell	BCube
Nodes Number	n^{2k-1}	$a_1 = n(k=1)$ $a_k = a_{k-1}(a_{k-1}+1)(k \geq 2)$	n^k
Node degree	k	k	k
Link Number	kn^{2k-1}	$(k+1)\frac{a_k}{2}$	kn^k
Switches Number	kn^{2k-2}	$\frac{a_k}{n}$	kn^{k-1}

Table 1.2: Comparison of layered topologies.

	c-Through	VL2	DCell	Ficonn	BCube	HyperBcube
Scalability	$O(c)$	$O(n^c)$	$O(n^{2^{(k-1)}})$	$O(n^{2^{(k-1)}})$	$O(n^k)$	$O(n * an^{(k-1)})$

Table 1.3: Categories of different DC network topologies according to their scalability.

n	Tree-based Topology			k	Recursive Topology			
	Basic Tree	Fat Tree	Clos Network		DCell	BCube	Ficonn	HyperBcube
4	9	3	3	2	20	16	16	64
				3	420	64	32	1024
				4	176820	252	64	16384
6	64	16	8	2	42	36	81	216
				3	1806	216	822	7776
				4	3×10^6	1269	42×10^6	823543
8	216	54	36	2	72	64	256	512
				3	5252	512	8192	32768
				4	27×10^6	4096	4×10^6	2×10^6
16	512	128	96	2	272	256	4096	4096
				3	74256	4096	2×10^6	1×10^6
				4	5514×10^6	65536	274×10^6	268×10^6

Table 1.4: Number of nodes under different configurations.

A DCell network has both α and β links, whereas a BCube network has only β links. However, BCube has scalability issues and DCell has a high wiring complexity [57]. FatTree and Clos-based networks are built using mainly γ links. Since γ links connects only switches, their intensive use has a negative impact on scalability [8]. An α link is the most simple and direct connection between a pair of nodes. Without any intermediate buffering, communication efficiency and maximal allowed bandwidth can be high. In contrast, a β link requires an additional intermediate switch for communication, but provides multiple non-blocking paths that allow multiple pairs of nodes to share their communication channels. As a result, a good tradeoff between cost and performance would be to use β links and small-port-count switches.

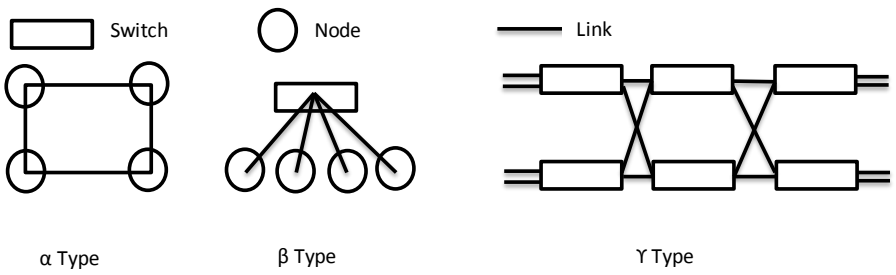


Figure 1.8: Link types in a DC network

For the bandwidth, the One-to-One, One-to-Several and One-to-All bandwidths are limited by the number of ports on each node (nodes degree k). So, for a tree-based topology, the bandwidth equal 1, while for a recursive topology the bandwidth equals k. Consequently, the basic tree topology has the smallest All-to-All bandwidth because of the limited number of switch ports at the root. In addition, this indicates that recursive topologies offers a great bandwidth performance under any traffic configuration ($k > 2$).

Furthermore, existing topologies are rigid, in the sense that if we need a specific number of nodes for our DC, the adopted topology will force us to use a generally much higher number of nodes that is statically defined given the number of ports per switch and the degree of each node. For large-scale

	BCube			HyperBcube			FatTree		
Nodes	W_n	W_{eg}	W_c	W_n	W_{eg}	W_c	W_n	W_{eg}	W_c
45	4	84	3550	19	192	7300	9	72	4950
450	34	528	23200	62	192	13400	100	156	10850
4500	124	1632	73600	413	4896	679200	420	5184	530400
45000	369	5112	228600	1659	19872	4.6×10^6	1299	15732	2×10^6

Table 1.5: Cost comparison between different topologies

DCs, as the number of nodes increases, we need to increase the number of layers and the number of ports per switches, which increases the number of links. Table.1.5 presents the estimated wasted cost for different topologies with different needed numbers of servers. We assume a price of 450 USD for an Ethernet switch port, 50 USD for each inter rack cable, and 12 watt energy consumption per switch port [58]. We denote by W_n the number of additional unneeded nodes imposed by the adopted topology, W_{eg} the wasted energy in Watts, and W_c the wasted cost in USD. The results in Table.1.5 show that the illustrated topologies have a physical limitation and their scalability relies entirely on increasing the number of ports per switch. For instance, for 4500 nodes, HyperBcube connects 4913. This results in 413 extra nodes and $46,000$ waste in cost, which is very high even for a small number of nodes.

All the above limitations have been considered in the design of new DC topologies to enhance the DC quality of service (QoS), enabling only β links and small port-count switches to provide the required performance while reducing the overall cost and energy.

1.6 Conclusion

In this chapter, we provide a comprehensive survey on the features, topologies, and hardware of DCN's. We first give an overview of DCs. Next,

we introduce the hardware of DCN's, including switches, servers, racks and cables used in industries, which are highly essential for designing DCN topologies. And then we thoroughly analyze the architectures of DCN's from various aspects and network characteristics.

Chapter 2

Enhancing QoS of Dc

2.1 Introduction

Mega DCs provide the core support infrastructure for the cloud and amounts for up to 45% of the total implementation cost. Consequently, the DC infrastructure must be well designed to improve the network performance [7]. In this chapter, we present a novel DC topology called LCT. While using the same number of links and switches per node as HyperBcube and BCube, LCT outperforms these topologies in terms of Scalability, APL, Bisection bandwidth and ABT.

This chapter is organized as follows: In section 2.2, the physical structure and routing algorithms of proposed topology LCT are presented. The key features are presented in section 2.3. The specialization of LCT is given in section 2.4.Finally, we conclude in section 2.5.

2.2 Physical structure

A 1-layer LCT network is basically composed of n nodes interconnected with one n-port switch (see Figure 2.1). A 2-layer LCT is composed of m 1-layer LCT (*cluster*) numbered from 1 to m, interconnected with m

n-port switches numbered from 1 to m we qualify as *internal switches*. The interconnection is represented as a $m \times n$ matrix L (see Figure 2.2) such that $L(i,j)$ ($\forall i \in \{1..m\}$ and $\forall j \in \{1..n\}$) is the number of the internal switch to witch node (i,j) (node j in cluster number i) is connect to.

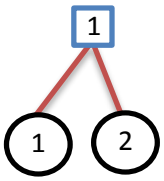


Figure 2.1: 1-layer LCT topology with n=2.

To generate matrix L we need first to generate its first row L_1, then we complete it as follows:

$$\forall i \in \{2..m\}, \forall j \in \{1..n\} \quad L(i,j) = (L(i-1,j)+1) \ mod \ m.$$

To generate L_1, we propose a novel algorithm we call *Linked Clusters Maximization* (LCM) (see Algorithm (1)). This algorithm maximizes the number of directly connected clusters, which leads to a reduction in the number of intermediate hops needed to transmit a packet to its destination (i.e., reduces the APL).

In Algorithm (1), the first element of vector L_1 is initialized to 1, then $\forall i \in \{2..n\}$, the best internal switch to be connected to node $(1,i)$ is selected by computing the size of the linked clusters set for each possible switch j ($j = 1..m$). The internal switch S^* maximizing the number of connected clusters is selected for node $(1,i)$ by setting $L_1(i) = S^*$. Note that when using n-port switches, the highest number of clusters a cluster can be connected to is equal to $n(n-1)$. This is due to the fact that a cluster has n nodes, each one of which can be connected to one distinct internal switch, which in turn can connect to $(n-1)$ different clusters. In Algorithm (1) we adopt a greedy approach to find the best internal switch. So there is no need to check all possible internal switches $\forall j \in \{1..m\}$ all the time to find the best one to connect to. In fact, at step i ($\forall i \in \{2..n\}$), if an internal switch that

Algorithm 1 Maximization Of The Linked Clusters Set

procedure LCM(n)
$j_{selected}$ is the index of the selected internal switch.
D is the Connectivity vector.
Input:
n is the column number in the matrix L.
Output:
L_1 is the first line of the matrix L.

$L_1[1] \leftarrow 1$, $j_{selected} \leftarrow 0$;
for $i \leftarrow 2$ to n **do**
$D[\,] \leftarrow \emptyset$
for $j \leftarrow 1$ to m **do**
$L_1(i) \leftarrow L_1(i-1) + j$
D(j)$\leftarrow LinkedClusters$(i,$L_1$)
if $D(j) \leftarrow i(i-1)$ **then**
Break
end if
end for
$j_{selected} \leftarrow argmax(D)$
$L_1(i) \leftarrow L_1(i-1) + j_{selected}$
end for
end procedure
function $LinkedClusters$(p,L_1)
Input:
L_1 is the first line of the matrix L.
p is the internal switch index
Output:
LC is the connected cluster vector.

$LC[\,] \leftarrow \emptyset$
for $i \leftarrow$ p downto 1 **do**
for $j \leftarrow 1 to (i-1)$ **do**
$LC \leftarrow [LC \;\; L_1(i) - L_1(i-j)]$
$\Omega \leftarrow$ Add (LC, Ω)
end for
end for
$LC \leftarrow unique(([LC\ m - LC])\ mod\ m))$
$\Omega \leftarrow$ uniqueΩ
return (Length(LC)) ;
end function

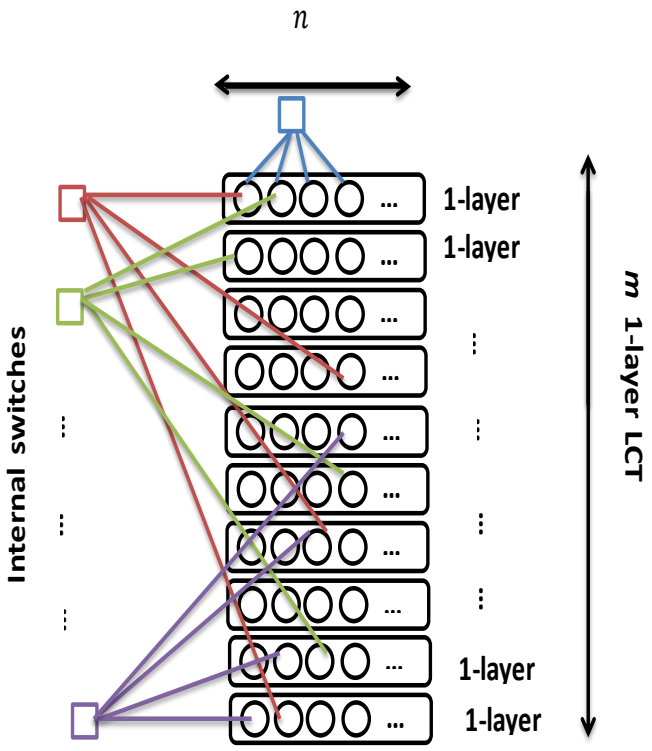


Figure 2.2: 2-layer LCT topology ($m = n^2$ and $m = \frac{n^3}{2}$).

connects a number of clusters equal to the highest number of clusters using i-port switches is found (i.e., $i(i-1)$), then it is directly selected without the need to check further internal switches. After L_1 is totally generated, the set Ω of linked clusters distances is computed as stated in function $LinkedClusters$. The set Ω is such that $\forall i \in \{1..m\}$ and $\forall j \in \Omega$, cluster i and cluster $(i+j)$ mod m are directly connected.

Figure 2.3 shows the network topology of LCT built using 2-port switches. To connect 8 nodes based on LCT topology, 4 internal and 4 external 2-port switches are used.

For a k-layer LCT network with $k > 2$, the total number of connected nodes is $n(m)^{k-1}$. Its connection pattern follows the same pattern as a 2-layer LCT network. In fact, a 2-layer LCT connects m 1-layer LCT following the pattern computed in matrix L. Similarly, a 3-layer LCT connects m 2-layer LCT following the same pattern in matrix L. In general, a k-layer LCT

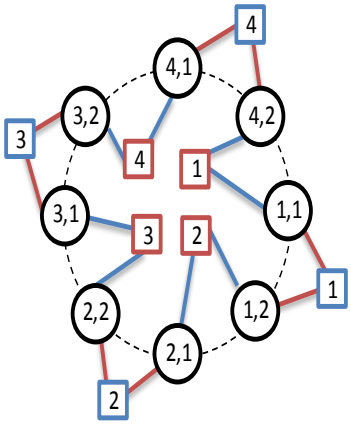


Figure 2.3: LCT network for n=2 and k=2.

connects m $(k-1)$-layer LCT following the pattern in matrix L. Following this recursive structure, the label of a node in the k-layer LCT network is built from its label in the $(k-1)$-layer LCT network where it appears, prefixed with the number of that $(k-1)$-layer LCT network. So, node labeled $(C_{k-1}, .., C_1)$ in the $(k-1)$-layer LCT network number C_k, will be relabeled $(C_k, C_{k-1}, .., C_1)$.

Algorithm 2 shows how to construct a k-layer LCT network using m $(k-1)$-layer LCT networks numbered from 1 to m.

Figure 2.4 shows an example of a LCT network where $n = 2$, $k = 3$. The network is divided into four 2-layer LCT connected by the intermediate of 16 switches.

2.2.1 Fault free routing scheme

To forward a packet from a source $(S_k, S_{k-1}, \ldots, S_1)$ to a destination $(D_k, D_{k-1}, \ldots, D_1)$, we propose a hierarchical row-based routing algorithm. A path P can be established using the following k steps, where only one coordinate is used in each step:

$P = (S_k, S_{k-1}, \ldots, S_1) \rightarrow (D_k, ?, \ldots, ?) \rightarrow (D_k, D_{k-1}, \ldots, ?) \rightarrow \ldots$
$\rightarrow (D_k, D_{k-1}, \ldots, D_2, ?) \rightarrow (D_k, D_{k-1}, \ldots, D_2, D_1)$

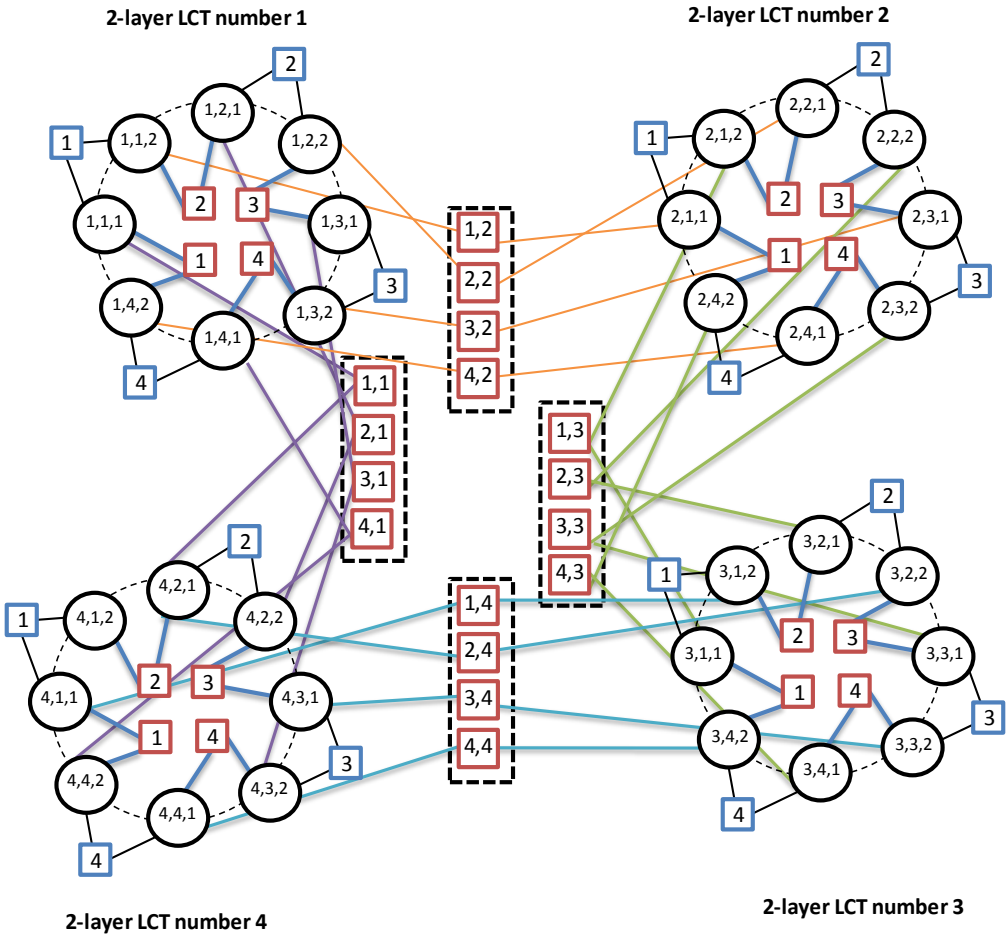


Figure 2.4: LCT network for n=2 and k=3.

Algorithm 2 Layered Linked Clusters Maximization

procedure LAYERED LCT(L)
k is the network degree.
Input:
L is the connection matrix.
m $(k-1)$-layer LCT networks numbered from 1 to m
Output:
k-layer LCT network

/* Connect nodes/*
for each tuple $(C_k, C_{k-1}, .., C_i, .., C_1) \in \{1, .., m\}^{k-1} \times \{1, .., n\}$ **do**
Connect node $(C_{k-1}, .., C_i, .., C_1)$ in the C_{k-1}-layer LCT network number C_k to the internal switch number $(C_{k-1}, .., C_2, L(C_k, C_1))$.
end for
/* Relabel nodes/*
for each tuple $(C_k, C_{k-1}, .., C_i, .., C_1) \in \{1, .., m\}^{k-1} \times \{1, .., n\}$ **do**
Change the label of node $(C_{k-1}, .., C_i, .., C_1)$ in the C_{k-1}-layer LCT network number C_k to $(C_k, C_{k-1}, .., C_i, .., C_1)$.
end for
end procedure

where "?" denotes unknown/don't care value.

For $k = 2$, to forward packets from node (S_2, S_1) to (D_2, D_1), we propose a cluster based fault free routing scheme as shown in Algorithm 3. If the source and destination have the same second coordinate, they are in the same cluster and therefore are directly connected via an external switch. However, if the modular difference between the source and destination clusters belong to Ω (i.e. $(D_2 - S_2)\ mod\ m) \in \Omega$), then an internal switch can be used to connect the nodes with a maximum of 3 hops. If the modular difference does not belong to Ω, then 2 or 3 internal switches have to be used to forward the packet with up to 5 hops.

Thanks to the incremental nature of the links matrix L, the route used to forward a packet from (S_2, S_1) to (D_2, D_1) can be directly deduced from the route used to forward the packet from $(1, S_1)$ to $((D_2 - S_2)\ mod\ m, D_1)$ by adding D_2 to the second coordinate of each node in the route. For instance, if $(1, S_1)$ is connected to (D_2, D_1) via the intermediate nodes (T_2^1, T_1^1) and (T_2^2, T_1^2) , i.e., via the path $(1, S_1) \rightarrow (T_2^1, T_1^1) \rightarrow (T_2^2, T_1^2) \rightarrow (D_2, D_1)$, then (S_2, S_1) is connected to $((D_2 - S_2)\ mod\ m, D_1)$ via the path $(d, S_1) \rightarrow ((T_2^1 + d)\ mod\ m, T_1^1) \rightarrow ((T_2^2 + d)\ mod\ m, T_1^2) \rightarrow ((D_2 + d)\ mod\ m, D_1)$, where $d = (D_2 - S_2)\ mod\ , m$. Hence, the algorithm (3) constructs the route the nodes of the first cluster to the other nodes, while the other routes can be directly deduced.

For $k > 2$, Algorithm 3 can be generalized for multi-layer LCT with two different cases as shown in Figure 2.5 and Figure 2.6. In case (a) (see Figure 2.5) when $(S_k - D_k)\ mod\ m \in \Omega$, D_k and S_k are directly connected to a common switch. Thus, a path of $(..., S_k, \dots) \rightarrow (..., D_k, \dots)$ exists. Given a random position in the source row, one additional transition through the 1-layer LCT may be required, leading to a maximum path length of two in the worst case. In case (b) (see Figure 2.6) when $(S_k - D_k)\ mod\ m \notin \Omega$, there is no direct connection between D_k and S_k , thus a path of $(..., S_k, \dots) \rightarrow (..., I_k, \dots) \rightarrow (..., D_k, \dots)$ is taken, where I_k denotes a common intermediate row between D_k and S_k. Accordingly, the path length is increased, leading to a maximum length of 3 intermediate

Algorithm 3 Fault Free Routing algorithm

1: **procedure** $FaultFreeRouting((S_2, S_1), (D_2, D_1), \Omega)$
2: Input:
3: Ω is the vector of directly connected clusters
4: (S_2, S_1) is the source coordinates
5: (D_2, D_1) is the destination coordinates
6: Output:
7: $Path$ is the path from the source to the destination
8: **if** $S_2 = D_2$ **then**
9: /∗The source and the destination are in the same cluster and are directly connected via an external switch/∗
10: $Path \leftarrow (S_2, S_1) \rightarrow (D_2, D_1)$
11: **else**
12: **if** $(S_2 - D_2)\ mod\ m \in \Omega$ **then**
13: /∗The source and the destination are directly connected/∗
14: Find T_2^1 and T_1^1 such that $P \leftarrow (S_2, S_1) \rightarrow (S_2, T_1^1) \rightarrow (D_2, T_2^1) \rightarrow (D_2, D_1)$.
15: **else if** $(S_2 - D_2)\ mod\ m \notin \Omega$ **then**
16: /∗The source and the destination are not directly connected and are linked only by the intermediate of 2 switches/∗
17: Find T_1^1, T_2^1, T_1^2 and T_2^2 such that $P \leftarrow (S_2, S_1) \rightarrow (T_2^i, T_1^i) \rightarrow (T_2^j, T_1^j) \rightarrow (D_2, D_1)$ where (i, j) is an arrangement of {1,2}
18: **else**
19: /∗The source and the destination are not directly connected and are linked only by the intermediate of 3 switches/∗
20: Find T_1^1, T_2^1, T_1^2, T_2^2, T_1^3 and T_2^3 such that $P \leftarrow (S_2, S_1) \rightarrow (T_2^i, T_1^i) \rightarrow (T_2^j, T_1^j) \rightarrow (T_2^k, T_1^k) \rightarrow (D_2, D_1)$ where (i, j, k) is an arrangement of {1,2,3}.
21: **end if**
22: **end if**
23: $Path \leftarrow P$
24: **end procedure**

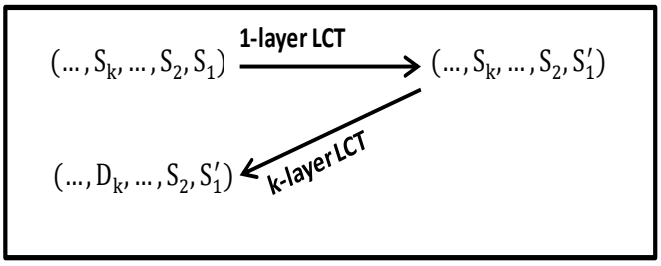


Figure 2.5: Case (a) when $(S_k - D_k)\ mod\ m \in \Omega$.

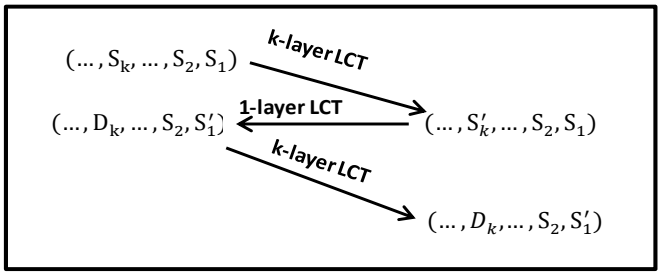


Figure 2.6: Case (b) when $(S_k - D_k)\ mod\ m \notin \Omega$.

switches in the worst case.

2.2.2 Fault tolerant routing scheme

If some links are in failure, we propose a real time fault tolerant routing algorithm to change the routing tables of some nodes. We define the $MaxLifeTime$ as the maximum length of a route to forward a packet to a destination. As shown in Algorithm 4, if no route can be found using the fault free routing algorithm, the fault tolerant algorithm allows the system to find a reachable node to resume the routing.

Figure 2.7 presents an example with multiple link failures, where a feasible path can still be established.

Algorithm 4 Fault Tolerant Routing Algorithm

procedure $FaultTolerantRouting(MaxlifeTime)$
    NHops is the used number of hops
    Intput:
    $MaxLifeTime$ is maximum number of hops
    NHops=0
    while Routing failed and NHops<MaxlifeTime **do**
        Find nearby severs in a radius of NHops and try routing by supposing the selected node as new source
        Select only Routes shorter than $MaxlifeTime$
        NHops=NHops+1
    end while
end procedure

Table 2.1: Nomenclatures table used in LCT key features computation

$Latency_{i,j}$	The latency between node i and node j
d_{qb-SR}	The queuing/buffering delays related to servers
d_{qb-SW}	The queuing/buffering delays related to switches
d_T	The transmission delays on one link
d_P	The propagation delays on one link
$N_{Lk}^{i,j}$	The number of links
$N_{SR}^{i,j}$	The total number of servers
$N_{SW}^{i,j}$	The total number of switches
$\mathrm{PL}_{i,j}$	The path length between node i and node j
T_{avg}	The average throughput
δ_i	The size of the transmitted packet i
d_i	The transmission delay of a packet i
n_f	The total number of the transmitted packets
ρ_i	Status of reception of the packet i
N_{Links}	The total of two-way communication links
NCP_{ABT}	The proportion of the overall network capacity that the aggregate bottleneck throughput can reach

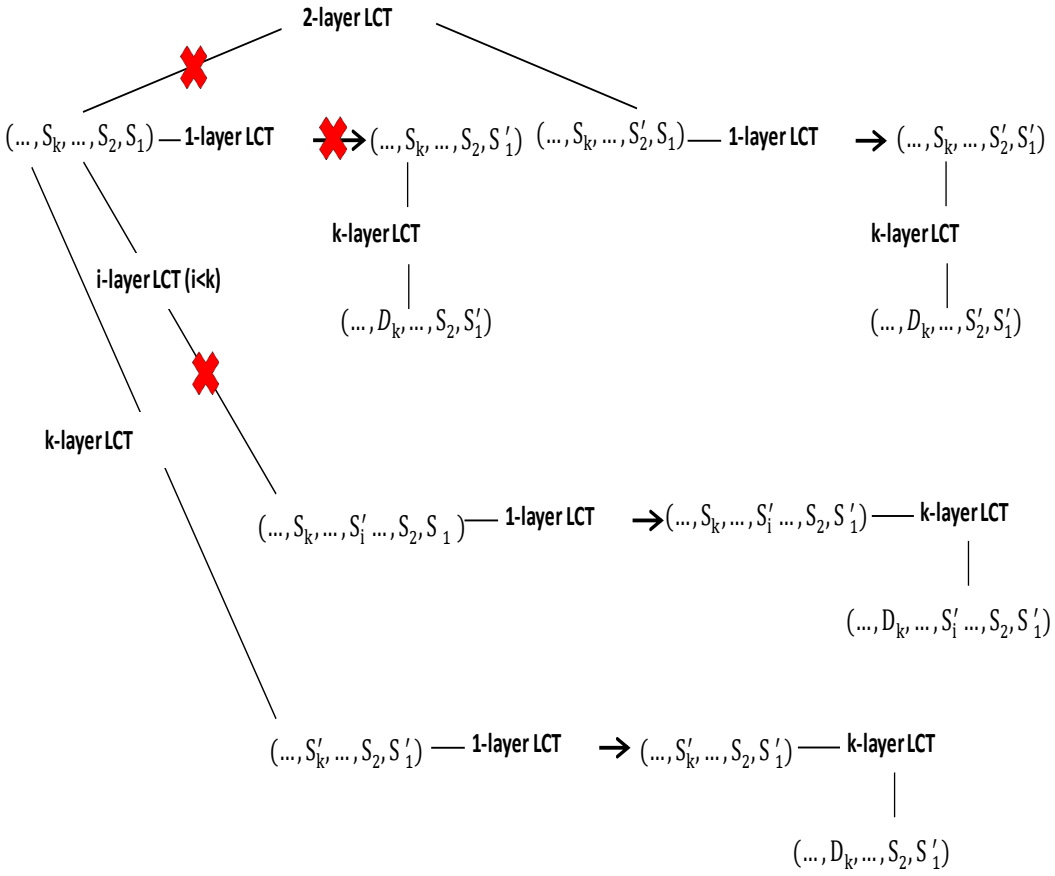


Figure 2.7: Routing in a k-layer ($k > 1$) LCT with multiple link failures.

2.3 LCT key features

2.3.1 Network latency

When a node i is transmitting data to node j, the latency is expressed as follows:

$$Latency_{i,j} = d_{qb-SR}N_{SR}^{i,j} + d_{qb-SW}N_{SW}^{i,j} + d_T N_{Lk}^{i,j} + d_P \tag{2.1}$$

where d_{qb-SR} and d_{qb-SW} are the queuing/buffering delays related to servers and switches, d_T and d_P denote the transmission and propagation delays on one link respectively (Table 2.1). For the path linking the nodes i and j, $\mathrm{N}_{Lk}^{i,j}$ denote the number of links, servers and switches used to forward the data from i to j. $\mathrm{N}_{SR}^{i,j}$ and $\mathrm{N}_{SW}^{i,j}$ represent the total number of servers and switches. We have the followings:

$$N_{SW}^{i,j} = N_{SR}^{i,j} = PL_{i,j} \tag{2.2}$$

$$N_{links} = 2PL_{i,j} \tag{2.3}$$

where $\mathrm{PL}_{i,j}$ denote the path length between node i and node j. Thus, Eq. 2.1 becomes:

$$\begin{aligned} Latency_{i,j} &= d_{qb-SR}.PL_{i,j} + d_{qb-SW}.PL_{i,j} + d_T.2PL_{i,j} + d_P \\ &= PL_{i,j}(d_{qb-SR} + +d_{qb-SW} + 2d_T) + d_P \end{aligned} \tag{2.4}$$

According to Eq. 2.4 the average latency is equal to

$$Latency = APL(d_{qb-SR} + +d_{qb-SW} + 2d_T) + d_P \tag{2.5}$$

Eq. 2.5 shows that the average latency is an increasing function of APL.

2.3.2 Fault tolerance

Table 2.2 shows that LCT has k parallel node-disjoint paths, which is similar to DCell and BCube, FatTree has no node-disjoint paths. Hence, with same number of switch per server as BCube ($\frac{k}{n}$), LCT increases the network performance since it has a multiple alternative paths which reduce the connection failure rate even with high link failure rate.

	LCT	DCell	BCube
Node degree	k	k	k
Switches Number	kn^{2k-2}	$\frac{a_k}{n}$	kn^{k-1}
Per Server	$\frac{k}{n}$	$\frac{a_k}{n(a_{k-1}(a_{k-1}+1))}$	$\frac{k}{n}$
Node-disjoint Paths	k	k	k

Table 2.2: Cost comparison between LCT, DCell and BCube.

2.3.3 Throughput

The throughput models the number of messages successfully delivered per unit time and this can be expressed as follows:

$$T_{avg} = \frac{1}{n_f} \sum_{i=1}^{n_f} (\frac{\rho_i * \delta_i}{d_i}) \tag{2.6}$$

where, δ_i is the size of the transmitted packet i, d_i represents the transmission delay of a packet i, n_f is the total number of the transmitted packets (Table 2.1) and $\rho_i \in [0, 1]$:

$$\rho_i = \begin{cases} 1 & \text{Successful reception of the packet } i \\ 0 & \text{failure reception of the packet } i \end{cases} \tag{2.7}$$

Hence, assuming that the data rate of the channel is fixed for all the topologies, the average throughput depends on the latency and the successful rate of the transmitted messages over a communication channel. LCT achieves low latency so it has low throughput.

2.3.4 Aggregate bottleneck throughput

LCT is a symmetric structure. Each node has exactly one link to the j-th layer which makes the number of total link number of different layers the same. In this way, LCT avoids bottlenecks and increases the ABT.

2.4 Specialization of LCT

2.4.1 Flat recursive topologies

Flat DC topologies simplify the DCs architecture and segment the network into simple partitions leading to a low routing complexity.

2.4.1.1 HyperFlatNet: LCT $(m = n^2, k = 2)$

HyperFlatNet characteristic [59]

In order to improve the DCs in terms of APL, we propose HyperFlatNet (LCT $(m = n^2, k = 2)$). HyperFlatNet scales similar to FlatNet while reducing the number of non-connected clusters and consequently reducing the APL and increasing the bisection bandwidth while using the same number of servers and switches as FlatNet.

Figure 2.8 shows an example of the HyperFlatNet network using 4-port switches.

HyperFlatNet simulation results

Figure 2.9 shows the APL of HyperFlatNet compared to DCell, BCube, FatTree for 1000 servers. The number of servers is varied from 1 to 1000. First, we remark that HyperFlatNet and FlatNet achieve lower APL than DCell, BCube and FlatTree. Note also that the APL of FlatNet starts to increase from 2.8 for 64 nodes to reach 3.25 for 1000 nodes. However, the APL of HyperFlatnet is less than 2.9 even when n=12. i.e. the APL in a 1728 nodes, HyperFlatnet network (n=12) is smaller than the APL of a 216 nodes, FlatNet network. This means that HyperFlanet can connect more

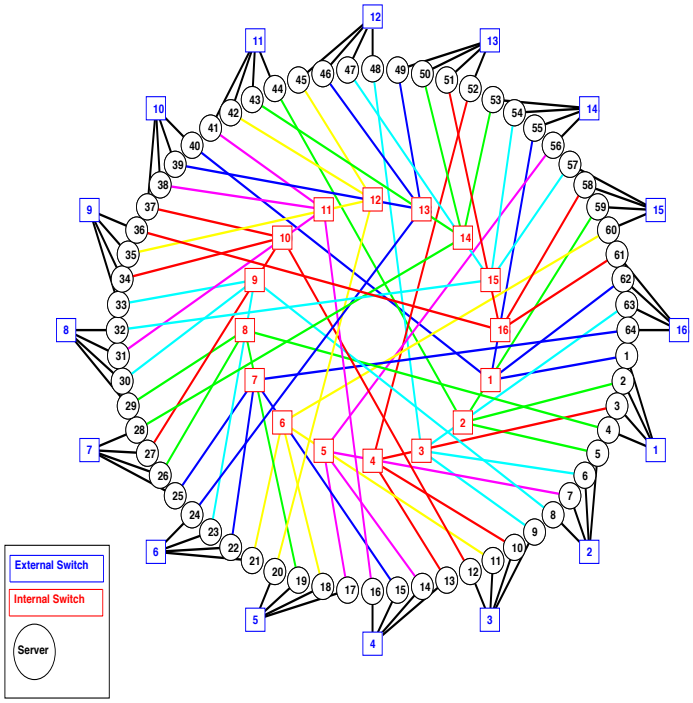


Figure 2.8: HyperFlatNet network using 4-port switches

than 8 times the number of nodes in FlatNet without increasing the APL. Thus, the proposed architecture reduces largely the latency compared to FlatNet since the latency is an increasing function of the APL.

Figure 2.10 presents the connection failure rate as a function of the link failure rate of HyperFlatNet compared to FlatNet with different values of MaxLifeTime. Furthermore, we varied the link failure from 0.02 to 0.27 and the maxlifetime between 4, 5 and 6 hops and the number of servers is fixed to 1000. First, we can remark that when the MaxlifeTime = 4 hops (equal to the diameter), the difference between the two topologies can be seen even for small link failure rates. In fact, the number of linked clusters in Hyper-Flatnet is bigger than FlatNet which increases the number of alternative links in case of failure. Thus, we notice that for Maxlifetime=4hops, the new topology always outperforms FlatNet. By increasing the maxlifetime to 5 hops, HyperFlatnet still outperforms FlatNet in terms of connection

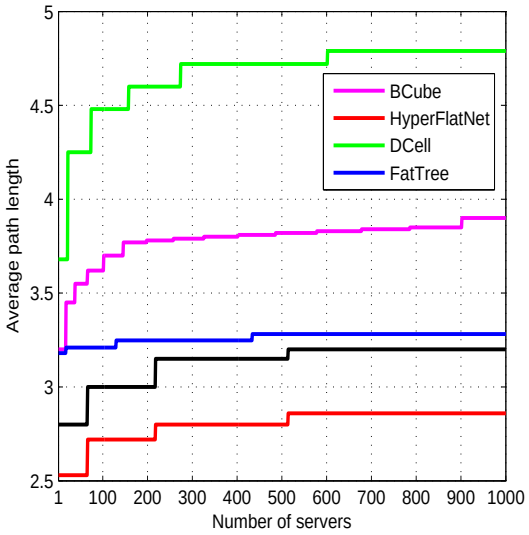


Figure 2.9: The APL of HyperFlatNet compared to DCell, BCube, FatTree for 1000 servers.

failure rate but with a smaller gap. However, when MaxLifeTime=6, there is approximately no difference between the two techniques. In fact, for a link failure rate smaller than 0.27, 5 hops is enough to find a route between any nodes that are not totally disconnected, hence using FlatNet or Hyper-Flatnet does not change a lot since 5 hops is largely enough to find routes. Consequently, Figure 2.10 proves that HyperFlatnet is more resistant than FlatNet to link failures.

2.4.1.2 ScalNet: LCT ($m = \frac{n^3}{2}, k = 2$)

ScalNet characteristic [60]

ScalNet is proposed to increase the flat network scalability; it scales faster than FlatNet, BCube and DCell with only two layers of network.

Figure 2.11 shows the network topology of ScalNet designed by using 4-port switches. To connect 128 servers based on ScalNet architecture, 32 internal and 32 external 4-port switches are used. However, for clarity reasons, only the connections of the internal switches number 1 and 32 and few external

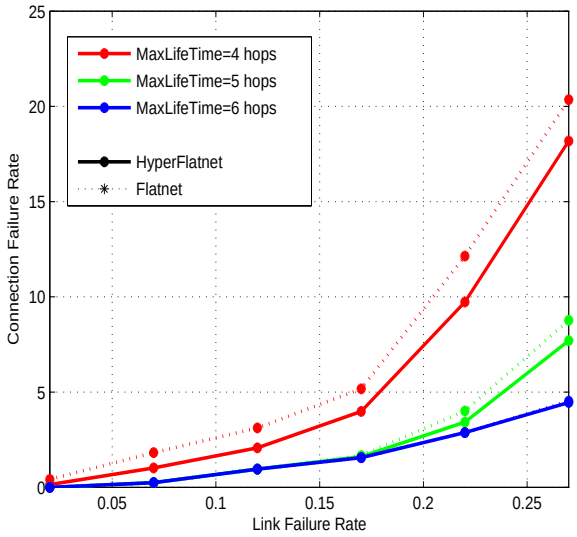


Figure 2.10: The performance of a 1000-server FlatNet/HyperFlatnet with different values of MaxLifeTime.

switches are presented. While using the same number of switches and wires per server compared to Flatnet, the proposed architecture increases the network scalability from $O(n^3)$ to $O(n^4)$. For example, when k = 2, n = 16, ScalNet can connect 32768 servers (700% and 12700% compared to Flatnet and BCube respectively). Furthermore, ScalNet reduces the number of non-connected clusters and consequently maintains low APL and network latency. Moreover, the proposed architecture achieves high bisection bandwidth and high aggregate bottleneck throughput.

The diameter of a ScalNet is 5 hops. It is slightly bigger than FlatNet and HyperFlatNet. However, this difference is still acceptable given its lower average cost and its larger scale.

ScalNet results

Figure 2.12 shows the number of nodes of ScalNet, FlatNet, DCell and BCube under different port count switches configuration. The switches port-count is varied from 4 to 12. Note that the number of nodes starts to increase from 128 when n=4 to reach 10368 when n=12 for ScalNet. However, the number of servers is 1728, 144, 156 for FlatNet, BCube and

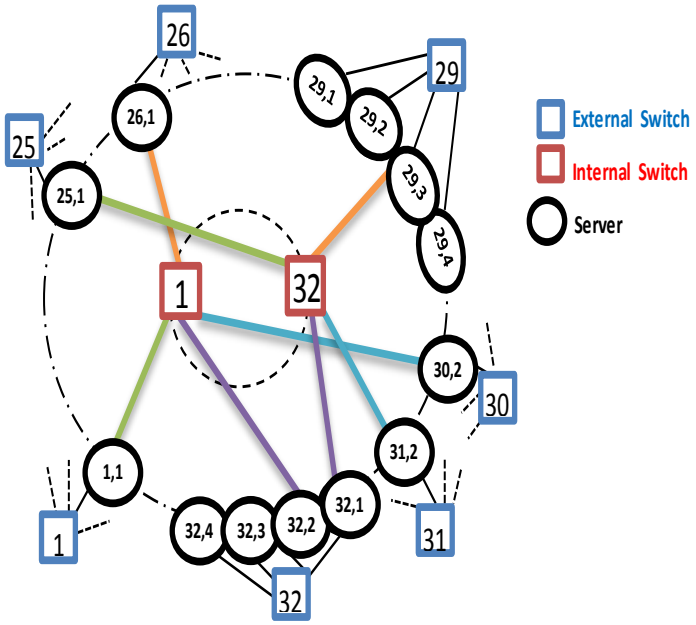


Figure 2.11: ScalNet network for n=4 and k=2.

DCell, respectively when n=12. Thus, ScalNet increases largely the number of nodes compared to all the previous architectures.

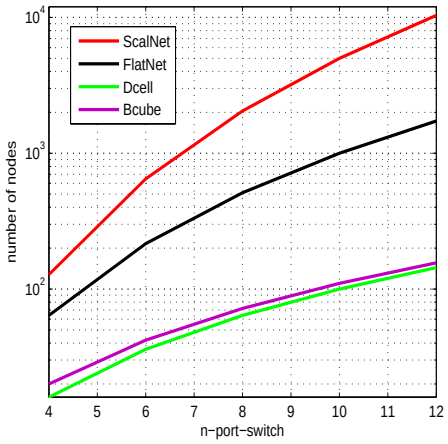


Figure 2.12: The number of nodes of ScalNet, FlatNet, DCell and BCube under different port count switches configuration.

Figure 2.13 proves that the percentage of number of nodes gain in ScalNet

compared to FlatNet, BCube and DCell is increasing in function of the port count switch. It can be seen that for n=12, the percentage reaches even 3100% , 2744%and 300% compared to DCell, BCube and Flatnet respectively. In fact, by using identical n-port switches, ScalNet can host $\frac{n^4}{2}$ servers which are approximately $\frac{n}{2}$ times that of a FlatNet and $\frac{n^2}{2}$ times that of a DCell/BCube.

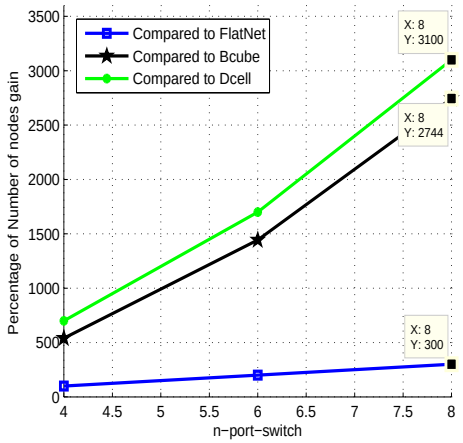


Figure 2.13: Percentage of number of nodes gain of ScalNet compared to Flatnet, BCube and DCell under different port count switches configurations.

2.4.1.3 ScalNet Vs HyperFlatNet

Figure 2.14 shows a comparison between ScalNet and HyperFlatNet. First, in terms of time transmission, HyperFlatNet achieves low APL compared with ScalNet. However in terms of scalability, ScalNet scales faster than HyperFlatNet. In fact, HyperFlaNet has more directly connected clusters compared with ScalNet which reduces the APL and the network diameter. However, ScalNet is able to connect $\frac{n}{2}$ more nodes compared with HyperFlatNet. bigger than HyperFlatNet. So, topologies can be selected according to the network requirements ScalNet or HyperFlatNet.

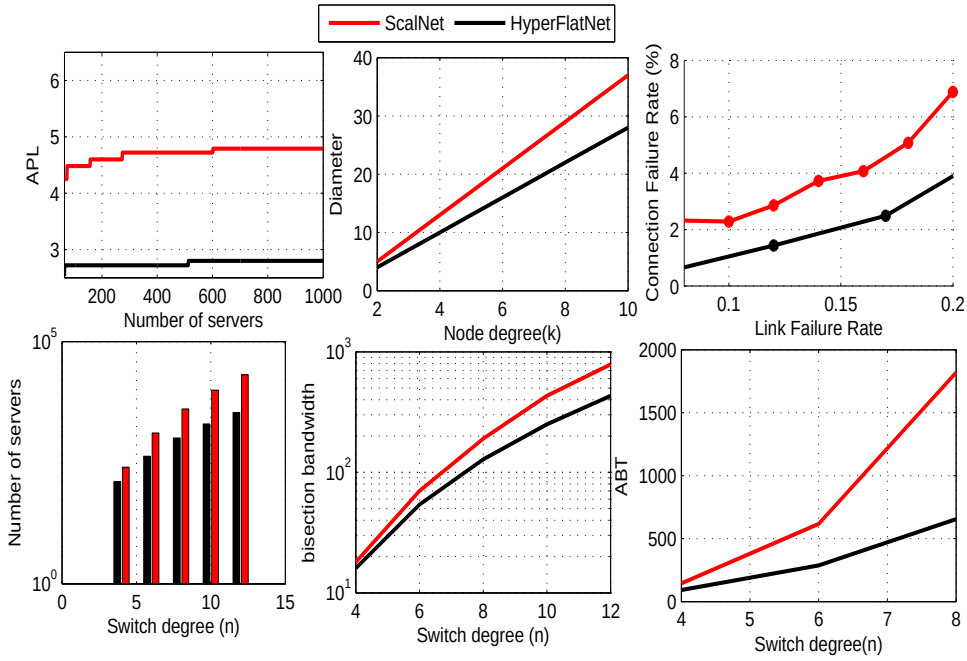


Figure 2.14: ScalNet Vs HyperFlatNet performance comparison.

2.4.2 Layered recursive topologies

2.4.2.1 LaCoDa: LCT ($m = n^2, k \geq 1$)

LaCoDa characteristic [61]

LaCoDa is an extension of HyperFlatNet, where $k \geq 1$. It scales the entire network to millions of servers using nodes with small degrees as well as switches with small port counts. We identified and evaluated different connection patterns between nodes and determined their effects on the properties of the overall topology (e.g., diameter, bisection bandwidth, APL, Latency).

LaCoDa results

Table 2.3 shows the performance under different network configurations. Both DCell and LaCoDa provide a much higher ABT than BCube. On the other hand, with smaller node degree, LaCoDa still offers equivalent scalability to ABT per link performance, leading to a more cost-effective

topology for large-scale DCs.

		DCell	BCube	LaCoDa
n=4 k=2	Nodes	20	16	64
	Links	30	32	128
	ABT	14.6	20	101.1
	APL	2.26	1.6	2.53
n=4 k=3	Nodes	420	64	1024
	Links	840	192	3072
	ABT	161	84	1417
	APL	5.16	2.29	4.33
n=4 k=4	Nodes	176820	256	16384
	Links	442050	1024	65536
	ABT	33589	340	21340
	APL	11.29	3.01	6.14
n=6 k=2	Nodes	42	36	216
	Links	63	72	432
	ABT	27	42	316.6
	APL	2.46	1.71	2.8
n=6 k=3	Nodes	1806	216	7776
	Links	3612	648	23328
	ABT	592	258	1003
	APL	5.73	2.51	4.76

Table 2.3: Performance analysis under different network configurations

Table 2.3 and Table 2.4 depict the number of nodes under different configurations. Both DCell and LaCoDa provide a much higher ABT than BCube. On the other hand, with smaller node degree, LaCoDa still offers equivalent scalability to ABT per link performance, leading to a more cost-effective topology for large-scale DCs. With only 4-port or 6-port switches, the number of nodes of the entire network could be of millions for both topologies. So, k does not need to be large to scale up.

Figure 2.15 depicts the APL of LaCoDa under different configurations. The switches port-count is varied from 4 to 10 and the node degree is varied

n	k	DCell	BCube	LaCoDa
4	2	20	16	64
	3	420	64	1024
	4	176820	256	16384
6	2	42	36	216
	3	1806	216	7776
	4	3263442	7776	279936
8	2	72	64	512
	3	5256	512	32768
	4	27630792	4096	2097152
16	2	272	256	4096
	3	74256	4096	1048576

Table 2.4: Number of nodes under different configurations

from 2 to 4. First, one can notice that LaCoDa's APL increases linearly to the node degree k, and additionally a larger sized LaCoDa holds longer APLs. Furthermore, APL takes values from 2.79 for 64 nodes (k=2,n=4) to 6.89 for 10 Million-node (k=2,n=10). So, even the number of server increases by 156250 times, APL does not exceed 7 and it also increases only by 2.46 times. Specifically, LaCoDa increases the number of directly connected clusters and reduces the number of intermediate hops during packet transmission. As a result, APL extents small values and it does not reach the maximum value (i.e diameter equal to 10).

Figure 2.16 depicts the distribution of path length with the configuration of 6-port switches and a node degree of 4. The results show that 94% of paths is between 2 and 7, and only 6% of paths reaches the maximum of 10. This explains why APL value is only 6.57 for 279936 nodes.

Figure 2.17 shows LaCoDa's aggregated bottleneck throughput under different configurations. The switches port-count varies between 4 to 10, and the node degree has a value between 2 to 4. The results show that LaCoDa's ABT increases when the node degree k increases. With 10-port switches and a node degree of 4, ABT of this 10 million-node DC is 11×10^6. If N_{Links} denotes the total of two-way communication links and NCP_{ABT} rep-

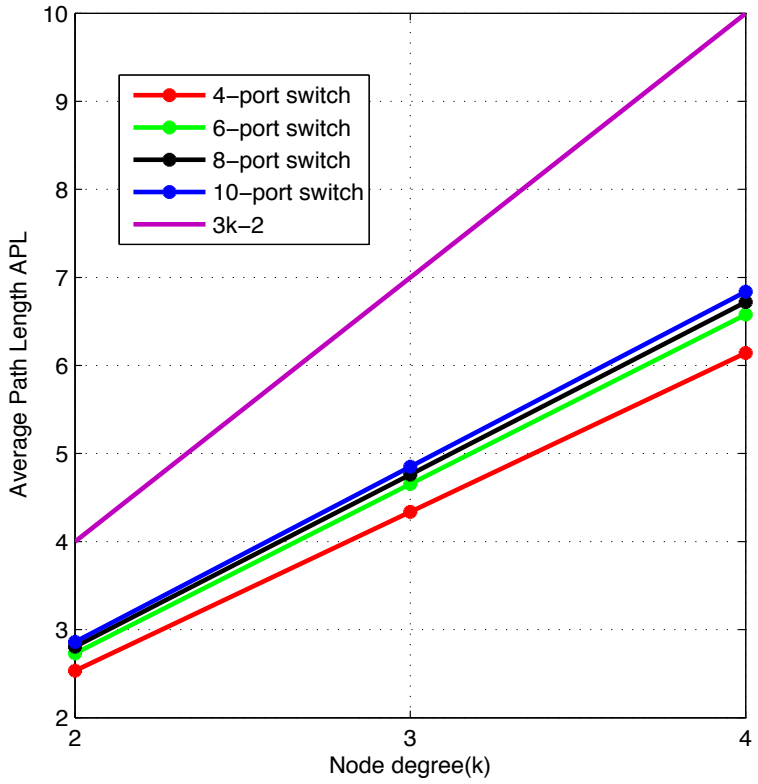


Figure 2.15: Average path length of LaCoDa under different configurations.

resents the proportion of the overall network capacity that the aggregate bottleneck throughput can reach[12], then we have the followings:

$$N_{Links} = kn^{2k-1} = 4.10^7 \tag{2.8}$$

$$\begin{aligned} NCP_{ABT} &= ABT/2N_{links} \\ &= 11.10^6/8.10^7 = 13.75\% \end{aligned} \tag{2.9}$$

This result is very close to theoretical values, namely NCP_{ABT}=1/APL =

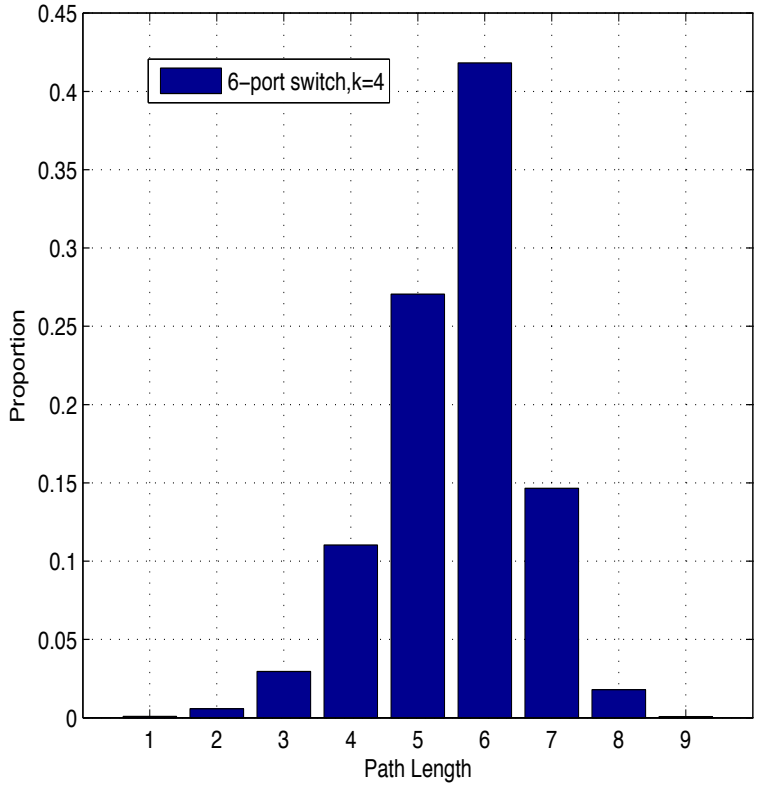


Figure 2.16: Path length distribution for $(n = 6, k = 4)$.

$1/6.9 = 14.49\%$. Thus LaCoDa has very good performance for the aggregate bottleneck throughput.

2.4.2.2 LaScaDa: LCT ($m = \frac{n^3}{2}, k \geq 1$)

LaScaDa characteristic

LaScaDa is an extension of ScalNet, it is capable of scaling the entire network bigger than LaCoDa in cost of APL.

LaScaDa simulation results

According to Table 2.5 and Table 2.6, LaScaDa provides a higher scalability and a much lower APL than the others topologies, even with small node degree ($k = 2$). In addition, by increasing k, the proposed topology still

HyperBcube				LaScaDa			
Nodes	n	k	APL	Nodes	n	k	APL
100	5	2	5.4	128	4	2	3.55
512	8	2	6	648	6	2	4.13
1331	11	2	6.1	2048	8	2	4.41
4096	16	2	6.34	32768	16	2	4.51
13824	24	2	6.36	80000	20	2	4.61

Table 2.5: The performance of HyperBcube and LaScaDa under different configurations (Fault-Free).

BCube				DCell			
Nodes	n	k	APL	Nodes	n	k	APL
100	10	2	3.55	110	10	2	4.25
625	25	2	3.75	600	24	2	4.7
1225	35	2	4	1260	35	2	4.8
4096	4	6	4.5	1806	6	3	5.73
16384	4	7	5.25	176820	4	4	11.29

Table 2.6: The performance of BCube and DCell under different configurations (Fault-Free).

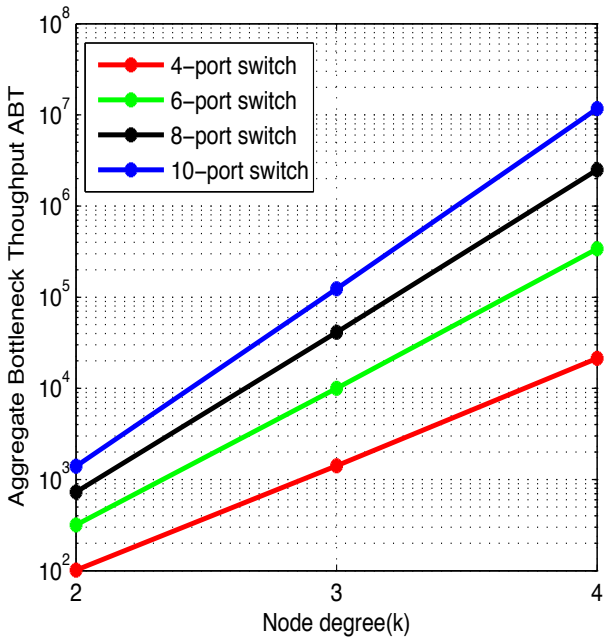


Figure 2.17: Aggregated bottleneck throughput of LaCoDa under different configurations.

provides low average path length even for a massive DC. For example, for $n = 8, k = 6$, the APL of this 8.8×10^{10} nodes network is only 18 (See Figure 2.18).

Figure 2.18 presents the APL of LaScaDa under different configurations. Switches port-count is varied from 4 to 8, while the node degree is varied from 2 to 6. First, it can be seen that the APL increases proportionally to the node degree k. So, a larger size LaScaDa has longer APL. However, we can see that by increasing the number of nodes, the APL takes values from 3.8 for 128 nodes (k=2, n=4), to 14.9 for 134×10^6-node (k=6, n=4). So, even if the number of nodes has increased by more than one million times, APL did not exceed 15, and increased only by 3.9 times. In fact, thanks to its physical structure and routing algorithms, LaScaDa increases the number of directly connected clusters while reducing the number of intermediate hops during packet transmission. Hence, the APL has small values and does not reach the maximum value.

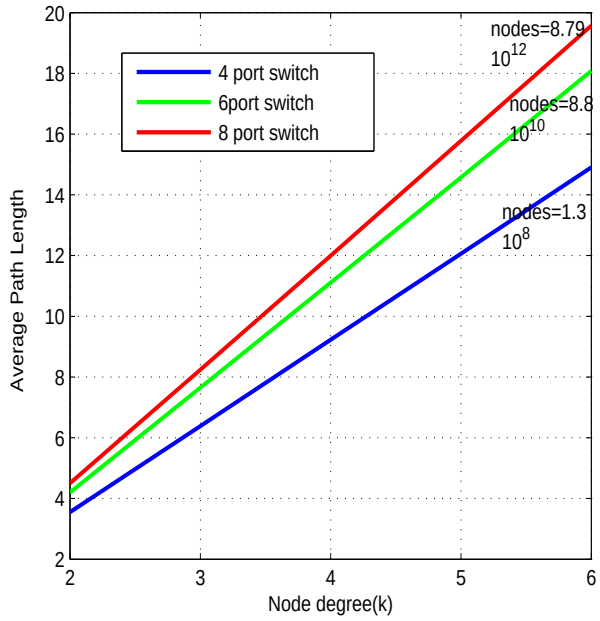


Figure 2.18: Average path length of LaScaDa under different configurations.

LaScaDa connects a greater number of nodes compared with all others topologies, it has the lowest diameter when compared with Ficonn and Flecube (See Figure 2.19). For a large value of k, the diameter of LaScaDa is approximately equal to the diameter of HyperBcube and BCube. In fact, thanks to its routing algorithm, LaScaDa reduces the APL even for a big number of nodes.

Figure 2.20 shows the scalability of LaScaDa under different port switch and node degree configurations. The figure shows that by using a small port count switch n and high node degree k, the scalability of the topology increases much faster than when using a big n and a small k. As a result, a good tradeoff between cost and performance would be to use only small-port-count switches and a high node degree.

Figure 2.21 shows the number of nodes of LaScaDa, HyperBCube, Ficonn, Flecube, DCell and BCube for switches with different port-counts and a node degree of 3. First, it can be seen that for LaScaDa the number of nodes starts to increase from 4096 when $n = 4$ to reach 6.99×10^{11} when

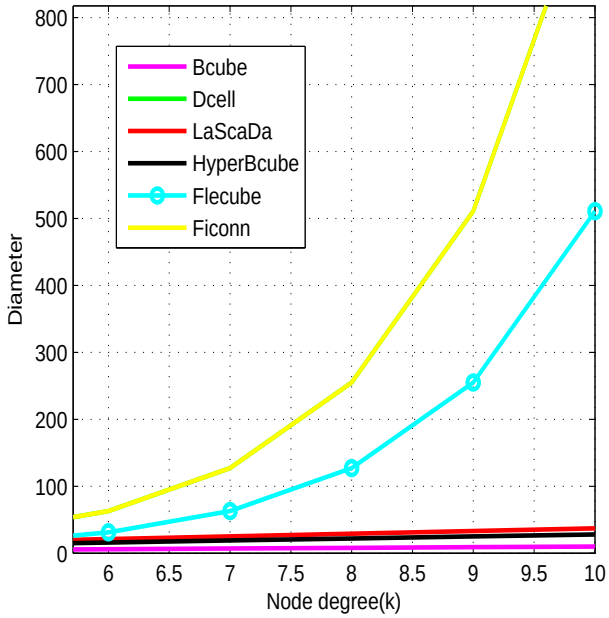
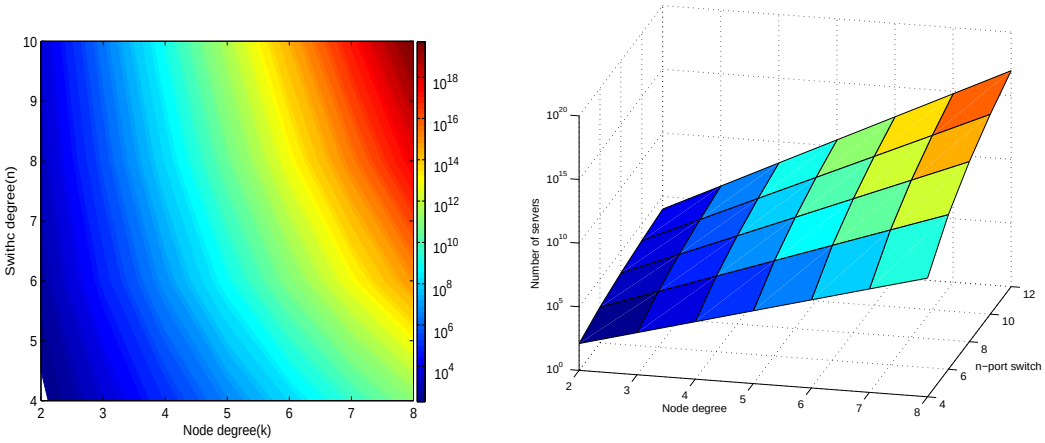


Figure 2.19: Network diameter of various layered topologies.



(a) 2D. (b) 3D

Figure 2.20: Scalability length distribution under different port switch and node degree configurations.

$n = 60$. However, the number of nodes is 7.77×10^8, 1.34×10^7, 3.46×10^6 and 2.16×10^5 for HyperBcube, Flecube, DCell, Ficonn and BCube, respectively

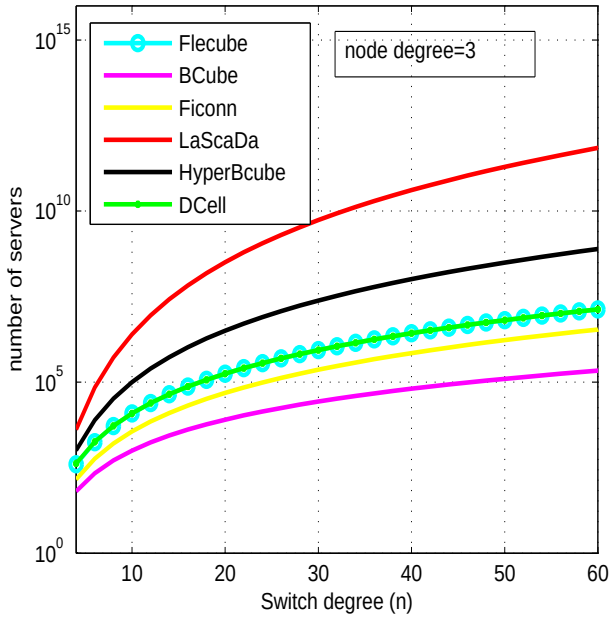


Figure 2.21: The number of nodes of LaScaDa, HyperBCube, Ficonn, Flecube, DCell and BCube under different port count switches configuration and k=3.

when $n = 60, k = 3$. Hence, LaScaDa allows for larger numbers of nodes compared with all other topologies.

Figure 2.22 depicts the distribution of the number of nodes under different configurations. Switches port-count is varied from 4 to 12, while node degree is equal to 2. First we can see that LaScaDa and HyperBcube support larger number of nodes compared with all other topologies. Besides, results show that by increasing the switch degree n, the difference between LaScaDa and all other topologies greatly increases. For instance, the number of nodes increases by 133% for $n = 12$. This shows the outperformance of LaScaDa in terms of scalability.

2.4.2.3 LaScaDa Vs LaCoDa

Figure 2.23 shows a comparison between LaScaDa and LaCoDa. We remark that LaCoDa achieves low APL compared with LaScaDa. However,

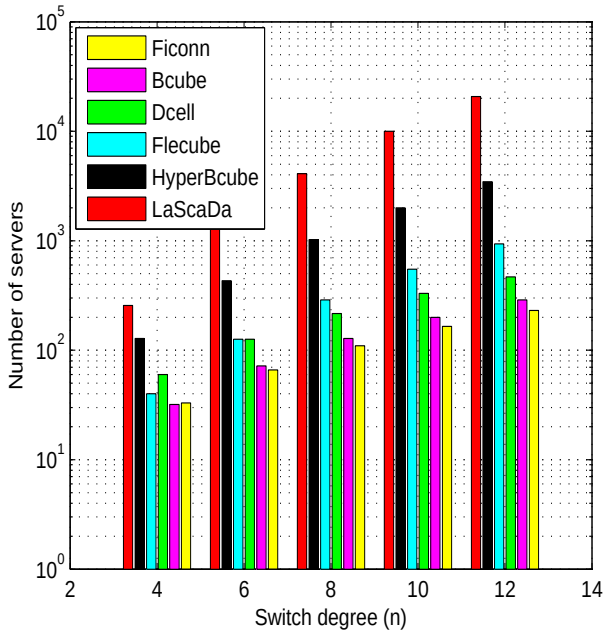


Figure 2.22: The number of nodes of LaScaDa, FlatNet, DCell and Bcube under different port count switches configurations.

LaScaDa connects bigger number of nodes than LaCoDa. So, LaScaDa and LaCoDa can be selected according to the network requirements.

2.5 Conclusion

In this chapter, we proposed and evaluated a novel topology for DCs called LCT. The proposed topology scales DCs to large sizes without a noticeable loss in performance compared to existing topologies. By using β links and small port-count switches, LCT scales up a DC to millions of nodes while preserving a good quality of service. LCT is characterized by its high Scalability, high Aggregate Bottleneck Throughput, a good Fault-Tolerance, a low Average Path Length and a high Bisection Bandwidth. Thanks to its special connections pattern and routing algorithms, LCT connects the highest number of directly connected clusters. Simulation results confirm the efficiency and outperformance of our proposed topology.

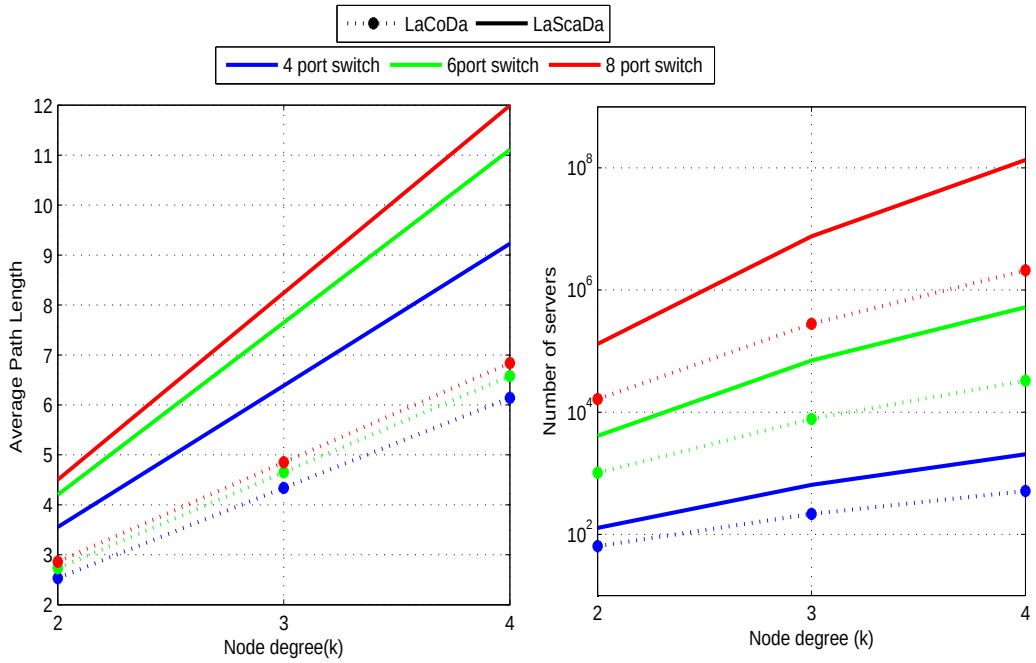


Figure 2.23: LaScaDa Vs LaCoDa performance comparison.

Chapter 3

Reducing DC cost and energy consumption

3.1 Introduction

Reducing the cost and the energy consumption is becoming a growing concern for DCs designers, operators and users. In this chapter, we introduce two techniques for green DCs. The first one is called VacoNet which performs static energy saving by allowing the exact number of nodes to be connected based on a novel DCs 's topology. This technique reduces the DCs s cost in terms of used switch and cables. We also propose a dynamic energy saving scheme that powers off the unused links and ports. This requires coordination among all nodes to ensure that the traffic packets will not take any path that would cross an inactive link. This chapter is organized as follows: In section 3.2, the proposed topology VacoNet is presented. The dynamic energy saving scheme is presented in section 3.3. Then, the cost reduction and its parameters are investigated in section 3.4. Finally, the conclusions are drawn in section 3.5.

3.2 Static energy saving

3.2.1 Physical structure

VacoNet connects n_{serv} servers using n_{sw} switches and n_1-port switch. The proposed topology is composed by two layers. The first layer of VacoNet network, denoted 1-layer, is basically composed of n_1 nodes interconnected with one n_1-port switch. The second layer, denoted 2-layer, contains n_2 1-layer interconnected with n_2 n_1-port switches numbered from 1 to n_2 (The detailed algorithm for the computation of n_1 and n_2 is presented in section 3.2.2).

The total number of servers and switches in VacoNet are given by:

$$n_{serv} = n_1 \times n_2 \tag{3.1}$$

$$n_{sw} = 2 \times n_2 \tag{3.2}$$

To interconnect the nodes, VacoNet uses the *Maximization Of The Linked Clusters Set (LCC)* of LCT topology (presented in section 2.2) with the parameters (n_1, n_2). Figure 3.1 shows the network topology of VacoNet built using 3-port switches.

3.2.2 Controlled VacoNet

In order to build a network with a specific number of nodes, the size of L can be adjusted according to the operator's requirements (from $m \times n$ to $n_2 \times n_1$). Given the needed number of nodes n_{serv}, we propose Algorithm 5 that initializes the n-port switch n_1 to $Floor(\sqrt[3]{n_{serv}})$ and the number of external switches n_2 to $Ceil(\frac{n_{serv}}{n_1})$, then the algorithm iteratively computes n_1 and n_2 based on their previous values.

If the number of rows n_2 in the matrix L is bigger than $\frac{n_1^3}{2}$, the n-port switch $n_1$will be increase by 1, which increases the total number of switches and

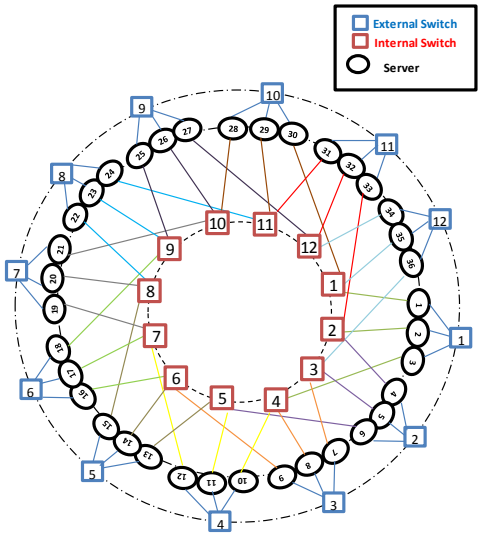


Figure 3.1: A 36-server 2-layer VacoNet constructed using 3-port switches.

Algorithm 5 n-port switch(n_{serv})

n_1 is the number of column in the matrix L.
n_2 is the number of row number in the matrix L.
$n_1 \leftarrow Floor(\sqrt[3]{n_{serv}})$
$n_2 \leftarrow Ceil(\frac{n_{serv}}{n_2})$
if $Or(n_2 > \frac{n_1^3}{2}, n_2 < n_1^2)$ **then**
    $n_1 \leftarrow n_1 + 1$
    $n_2 \leftarrow n_1^2$
end if
return n_1, n_2

Table 3.1: Nomenclatures table

Eg_{sw}	The energy consumption per switch sw
Eg_{sw}^{β}	The total energy consumed by β switches
p_i	The i^{th} port
G	The number of groups of switches
S^{Ms}	Master server
Eg^c	The energy consumption per cluster
N	The total number of nodes
M_t	The transmission matrix at time t
T	The system period
M_o	The total number of nodes
a	The fixed subset size of the moving average
λ	The rate parameter of the inter-arrival times of M_t
$\lambda^{'}$	The rate parameter of the inter-arrival times of M_v
$\lambda^{''}$	The rate parameter of the inter-arrival times of M_o
μ	The auto correlation factor
R	The connection matrix
γ	The communication threshold
M_o	The total number of nodes
P	The consumed energy for an active port
τ_d	The required time for closing a port
τ_{up}	The required time for activation a port
l_a	The maximum links to be activate per port
n_d	The number of ports being closed
n_c	The number of times closing for ports in a period T
n_{up}	The number of times activating for ports in a period T

rows in L (i.e. $n_2 = n_1^2$). Otherwise, the number of rows n_2 increases by 1, which means that the number of switches is increased by 1 while keeping the same number of n-port switch (n_1).

3.2.3 Performance evaluation

3.2.3.1 Power consumption

DCs s are some of the fastest growing infrastructures requiring lots of electrical power for their operation. Hence, reducing power consumption is im-

portant to reduce both operation cost and the impact on the environment. VacoNet physical structure is based on a connection algorithm that connect close to the exact number of needed nodes. The energy consumption per switch Eg can be written as:

$$Eg = \sum_{i=1}^{n} p_i \tag{3.3}$$

where p_i is the energy consumed by switch port number i. The total energy consumed by M switches Eg_M is:

$$Eg_M = M \times \big(\sum_{i=1}^{n} p_i\big) \tag{3.4}$$

If VacoNet reduces the number of used switches from M to G, then the consumed energy Eg_G can be computed as:

$$Eg_G = G \times \big(\sum_{i=1}^{n} p_i\big) \tag{3.5}$$

We can estimate the gain in energy consumption of VacoNet compared with existing topologies as follows:

$$
\begin{aligned}
Gain &= \frac{Eg_M - Eg_G}{Eg_M} \\
&= \frac{\left(M\sum_{i=1}^{n} p_i - G\sum_{i=1}^{n} p_i\right)}{M\left(\sum_{i=1}^{n} p_i\right)} \\
&= 1 - \frac{G\left(\sum_{i=1}^{n} p_i\right)}{M\left(\sum_{i=1}^{n} p_i\right)} \\
&= 1 - \delta
\end{aligned} \tag{3.6}
$$

where

$$
\delta = \frac{G\left(\sum_{i=1}^{n} p_i\right)}{M\left(\sum_{i=1}^{n} p_i\right)} \tag{3.7}
$$

Obviously, the number G of used switches in VacoNet is always much smaller than M. Therefore $Gain$ is always bigger than 1, which shows that the proposed topology can significantly reduce energy consumption compared with existing topologies by using only the needed number of servers.

3.2.3.2 Simulation results

Figure 3.2 shows the average path length of VacoNet compared with FlatNet, BCube, FatTree and ScalNet. The number of servers is varied from 0 to 1000. We can see that both VacoNet and FlatNet outperform FatTree, ScalNet and BCube by yielding a much shorter APL for a small number of nodes. For a larger DCs , the APL of VacoNet is less than 3, even when the number of servers reaches 1000. In addition, the APL in a 1000 nodes VacoNet network is smaller than the APL of an 83 nodes FlatNet network, a 23 nodes FlatTree network, and an 11 nodes BCube or ScalNet networks. This means that VacoNet can connect more than 12 times the number of nodes in FlatNet without increasing the APL. Thus, the proposed topology reduces largely the latency compared with ScalNet, BCube, FlatNet and FatTree since the latency increases with the APL.

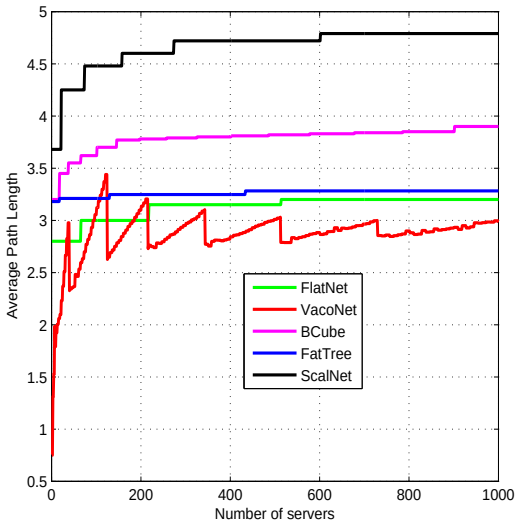


Figure 3.2: Average path length of VacoNet compared with FlatNet, BCube, FatTree and ScalNet.

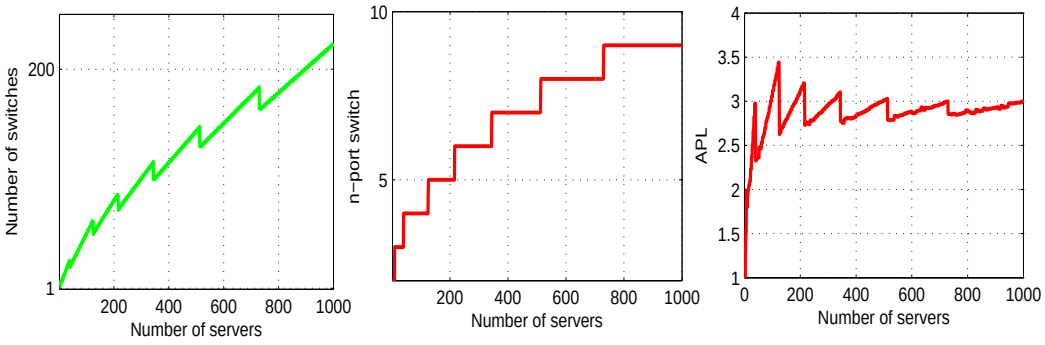


Figure 3.3: APL Vs the number of switches and n-port switch in VacoNet.

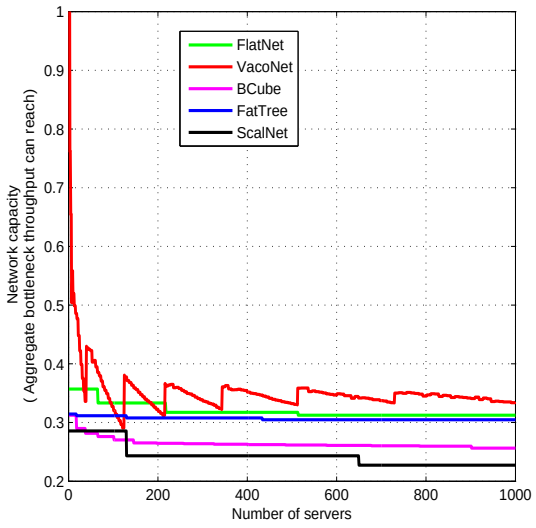


Figure 3.4: Network capacity of VacoNet compared with FlatNet, BCube, FatTree and ScalNet.

Figure 3.3 shows the APL vs the number of switches and the and the number of ports per switch (n-port) in VacoNet. The number of servers is varied from 0 to 1000. We can see that the APL for 123 nodes is less than the APL for 729 nodes. In fact for 123 nodes, the number of switches per servers is 1.98 . However, for 749 nodes it is equal to 4.45. By increasing the number of switches per server, we get more alternatives paths and the transmission of packets to the destination will be faster, which reduces the APL.

Figure 3.4 shows the network capacity of the aggregate bottleneck throughput that VacoNet can reach compared with FlatNet, BCube, FatTree and ScalNet. The number of servers is varied between 1 to 1000. We can remark that the network capacity for a 1000 nodes VacoNet is almost equal to the network capacity of a 220 nodes FlatNet, 10 nodes BCube and 1 node ScalNet. This means that VacoNet can connect more than 4 times the number of nodes in FlatNet, and 100 times the number of nodes in DCell and BCube with the same network capacity, which reveals the outperformance of VacoNet in terms of network capacity.

Figure 3.5 shows the power consumption of VacoNet compared with Flat-

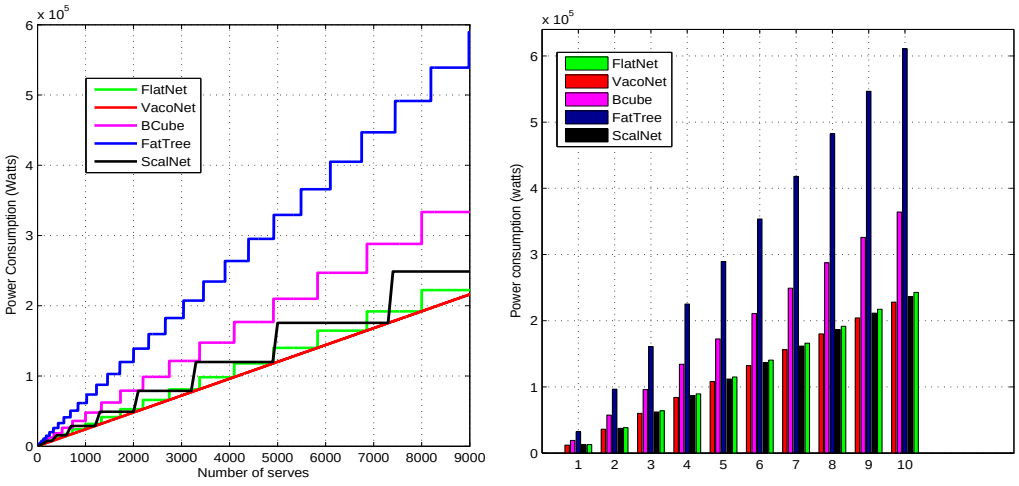


Figure 3.5: Power consumption of VacoNet compared with FlatNet, BCube, FatTree and ScalNet.

Net, BCube, FatTree and ScalNet. The number of servers is varied from 0 to 10000. It can be seen that VacoNet outperforms the other topologies in terms of power consumption. In fact, for 8200 nodes, BCube, FlatNet and FatTree exhibit high levels of power consumption reaching 5.4×10^5 watts for FatTree, 3.34×10^5 watts for BCube, 2.22×10^5 watts for FlatNet and 2.5×10^5 watts for ScalNet. On the other hand, VacoNet consumes only 1.96×10^5 watts.

3.3 Dynamic energy saving

3.3.1 Problem statement

3.3.1.1 Closing links strategy

In order to meet the quality of service requirements while increasing the total energy saving in a DCs, our approach reduces the number of active ports in the network, and increases the average link utilization. Links are divided into three types:

- **Critical cluster links (CCL)**: These are links connecting nodes to their clusters.
- **Critical non cluster links (CNCL)**: These are links that are not CCLs, but affect the connectedness of the network if they are closed.
- **Uncritical links (UL)**: these are links that do not affect the connectedness of the network if closed.

In the proposed approach, both CCLs and CNCLs are kept active in order to guarantee good performance. We deactivate a subset of ULs by deactivating the end ports of each one of them to save energy, while avoiding the problem of having disconnected nodes (see Figure 3.6).

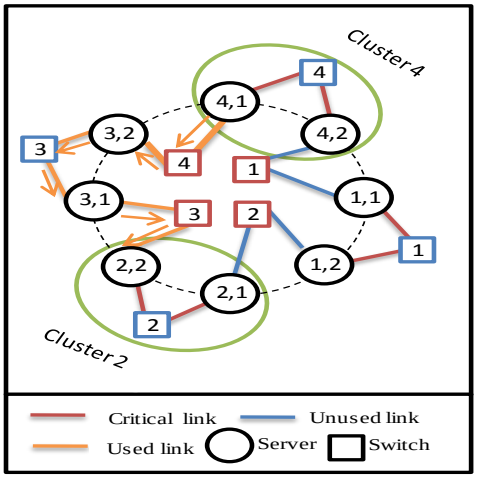


Figure 3.6: Example of network where cluster 4 communicates with cluster 2.

3.3.1.2 Routing strategy

Servers will also be classified into two categories:

- **Outreach server** (denoted by S^{out}): A node in this category has an active port other than the one that connects it to its cluster. Such a node can be involved in routing traffic outside the cluster.
- **Non outreach server**: The only active port of a node in this category is the one that connects it to its cluster.

3.3.1.3 Problem formulation

Links and ports can be deactivated to save energy without too much affecting system performance. Hence, to reduce energy consumption, we propose to reduce the number of active ports in each cluster. So, for n ports per switch and k ports per server, the energy consumption per cluster Eg^c can be written as:

$$\begin{aligned} Eg^c &= sw^c \sum_{i=1}^{n} p_i + ser^c \sum_{j=1}^{k} p_j \\ &= ser^c \left(\frac{1}{n} \sum_{i=1}^{n} p_i + \sum_{j=1}^{k} p_j \right) \end{aligned} \tag{3.8}$$

Given that k and n are not fixed as they depend on the DCs configuration chosen by the operator, the Eg^c expression becomes:

$$Eg^c = ser^c \left(\frac{1}{n} \sum_{i=1}^{min(n,k)} p_i + \sum_{j=min(n,k)}^{max(n,k)} p_j \right) \tag{3.9}$$

where sw^c is the switch number in cluster c, ser^c is the number of servers, and p_i presents the active port for node i.

The objective is to find a set of optimal routing paths that minimizes the total number of active ports:

Minimize: $\sum_{i=1}^{min(n,k)} p_i + \sum_{j=min(n,k)}^{max(n,k)} p_j$

Hence, reducing the total energy consumption in the network.

3.3.2 System model

3.3.2.1 Network traffic model

Given an N nodes network, let $M_t = (M_t(i,j), .., M_t(N,N))$ be the transmission matrix at time t, where $M_t(i,j)$ is the number of transmitted messages from node i to node j at time t. We assume that $M_t(i,j)$ fellows a Poisson distribution where inter-arrival times are exponentially distributed with rate parameter (λ). We assume that the system can perfectly estimate the traffic matrix $M(T)$ periodically with period T. We also assume that the network changes the number of active nodes smoothly in time using a moving average (MA) model with a fixed subset size equals to a (Figure 3.7). So, given $M(T)$ and a, the a first element of $M(t+1)$ are equal to the a last elements of $M(t)$. Then, the rest of elements $(N-a)$ will be regenerated. The bigger is a, the more correlated system will be (Figure 3.8). This process is repeated at the beginning of each period T.

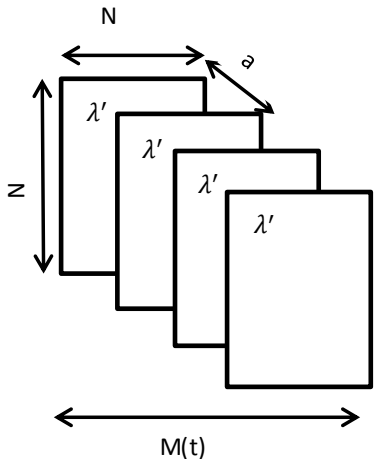


Figure 3.7: Example of a matrix moving average with a subset a.

For $t = t_0$

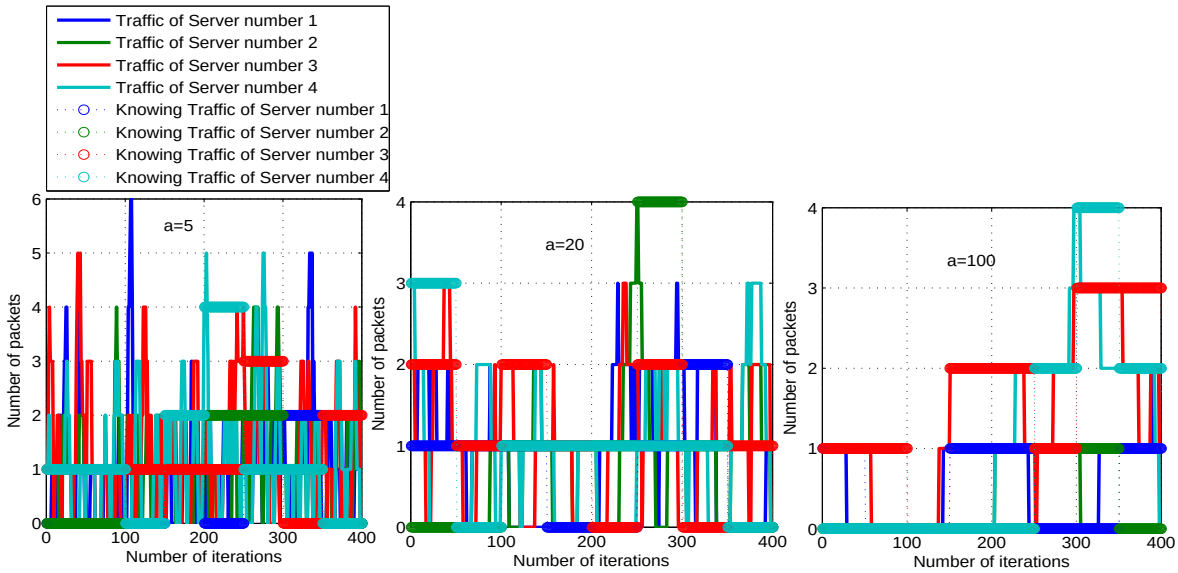


Figure 3.8: Example of the matrix correlation under different subset a

$$\begin{aligned} &\forall i \in \{1..N\}, \\ &\forall j \in \{1..N\}, \\ &M_{t_0} = \begin{cases} 0 & \text{If } (i = j) \\ \sum_{t=t_0}^{t_0+a} M_{t_0}(i,j,t) & \text{If } (i \neq j). \end{cases} \end{aligned} \tag{3.10}$$

where M_o is the application traffic matrix.

$$M_o(i,j,k) = L \tag{3.11}$$

Meaning that applications transmit L packets from node i to node j from time t to time $t+k$. We assume that $M_o(i,j,t)$ fellows a Poisson distribution with an exponentially distributed inter-arrival times with a rate parameter $(\lambda^{'})$ such as $(\lambda^{'} = \frac{\lambda}{a})$ to satisfy that M will be generated with λ. So, if we denote by $M_v(i)$ the number of transmitted and received message for node i

$$M_v(i) = \sum_{j=1}^{N} M(i,j) + \sum_{j=1}^{N} M(j,i) \tag{3.12}$$

$M_v(i)$ fellows a Poisson distribution with an exponentially distributed inter-arrival times with a rate parameter $(\lambda^{''}) = 2(N-1)\lambda$. M is correlated in time, with auto correlation factor $\mu_{x,y}$. So if we denote by:

$$\begin{aligned} X &= M_{t_0}(i,j) \\ &= \sum_{t=t_0}^{t=t_0+a} M_o(i,j,t) = \sum_{t=t_0}^{t=t_0+a} z(t) \end{aligned} \tag{3.13}$$

$$\tag{3.14}$$

$$\begin{aligned} Y &= M_{t_0+1}(i,j) \\ &= \sum_{t=t_0+1}^{t=t_0+a+1} M_o(i,j,t) = \sum_{t=t_0+1}^{t=t_0+a+1} z(t) \end{aligned} \tag{3.15}$$

$$\tag{3.16}$$

the auto correlation factor $\mu_{x,y}$ can be written as:

$$\begin{aligned} \mu_{x,y} &= corr(X,Y) \\ &= \frac{cov(X,Y)}{E(|X|^2)E(|Y|^2)} \end{aligned} \tag{3.17}$$

or

$$\begin{aligned} E(|X|^2) &= E(|Y|^2) \\ &= \lambda \end{aligned} \tag{3.18}$$

So , $\mu_{x,y}$ becomes:

$$\begin{aligned} \mu_{x,y} &= \frac{E(XY)}{E(X)E(Y)}\lambda^2 \\ &= \frac{E(XY) - \lambda^2}{\lambda^2} \end{aligned} \tag{3.19}$$

where $E(XY)$ is equal to:

$$\begin{aligned} E(XY) &= E\big(\sum_{t=t_0}^{t=t_0+a} z(t) \sum_{t=t_0+1}^{t=t_0+a+1} z(t)\big) \\ &= E\big(\sum_{t_1=t_0}^{t_1=t_0+a} \sum_{t_2=t_0+1}^{t_2=t_0+a+1} z(t_1)z(t_2)\big) \\ &= (a-1)E(|z(t)|^2) + (a^2-(a-1))E(z(t_1)z(t_2)) \\ &= (a-1)\lambda^{'} + (a^2-a+1)\lambda^{'2} \end{aligned} \tag{3.20}$$

$$\mu_{x,y} = \frac{(a-1)\lambda^{'} + (a^2-a+1)\lambda^{'2} - \lambda^2}{\lambda^2} \tag{3.21}$$

3.3.2.2 Activating and deactivating links

At the beginning of each time period T, we estimate the traffic matrix $M(T)$ and use it to decide which links to activate or deactivate. Using $M(T)$, we compute for each node i, its communication rate $M_v(i)$. If this rate is smaller than the threshold γ, the Uncritical links connected to the

node are deactivated, otherwise they are activated. Figure 3.9 shows a case where the maximum links to be activate is $CNCL$=2.

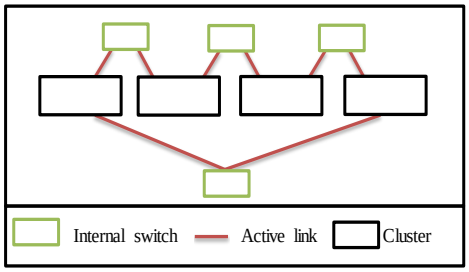


Figure 3.9: Case where the maximum links to be activated is $CNCL$=2.

The value of γ is defined based on the network load, the inter-arrival rate λ and according to the DCs requirements (time/energy). Figure 3.10 presents an example of a network with 20 nodes with the vector of number of transmitted & received message by nodes (Mv). In this example, nodes number 3, 8, 9, 10, 16, 17, 18, 19 did not receive or transmit any communication at instant t. However, nodes number 1, 6, 7, 14, 15, 20 have only one communication. So if, we initialize γ to 1, the uncriticle links connected to of servers 3, 8, 9, 10, 16, 17, 18, 19 and 1, 6, 7, 14, 15, 20 will be closed.

1	0	0	3	2	1	1	0	0	0	4	2	2	1	1	0	0	0	0	1
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20

Figure 3.10: Example of a network with 20 nodes with threshold $\gamma = 1$.

3.3.2.3 Network topology

The proposed energy saving technique can be applied for any network topology where a server has two or more links (recursive topologies), namely flat recursive topologies (FlatNet [53], HyperFlatNet [59]), layered recursive topologies(DCell [10], BCube [11], HyperBcube [51], LaCoDa [61]). These topologies can significantly increase the number of servers due to their recursive structures. Also, servers can be considered as computation

units and packet-forwarding devices. To avoid the problem of disconnected server while keeping a good network performance, the idea is to keep at least one active ports per server.

Without loss of generality, we use this approche on HyperFlatNet proposed in [59] (presented in section 2.4.1.1). HyperFlatNet network can suffer from high energy consumption as all ports are activated [62]. The authors in [62] proved that the HyperFlatNet topology is relatively stable compared with other topologies. However, HyperFlatNet has n times more links than BCube and DCell topologies, which leads to more energy consumption compared with existing topologies.

3.3.3 Closing ports management algorithm

3.3.3.1 Critical link classification algorithms

Critical links are defined as links used within the same cluster. Let n_{d1} be a node and n_{d2} be a switch. To find if an intermediate link (n_{d1}, n_{d2}) between a node and a switch is within the same cluster, we define the function $GetCommunPrefix$ in algorithm Algorithm 6 to return the common prefix of node n_1 and switch n_2. Let l_1 be the level of n_{d1} and l_2 be the level of n_{d2}. Algorithm 6 tests if the common prefix of a node and a switch is equal to their level, which means that they are in the cluster.

Algorithm 6 Critical cluster links$((n_{d1}, n_{d2}))$

$pref \leftarrow GetCommunPrefix(n_{d1}, n_{d2})$
$i \leftarrow length(pref)$
if $i = l_1$ and $i = L_2$ **then**
    Then n_{d1} and n_{d2} are known to be in the same cluster
    The links (n_{d1}, n_{d2}) is a critical links
    $L_c \leftarrow (n_{d1}, n_{d2})$
end if
return L_c

Figure 3.11 presents an example of HyperFlatNet links classification.

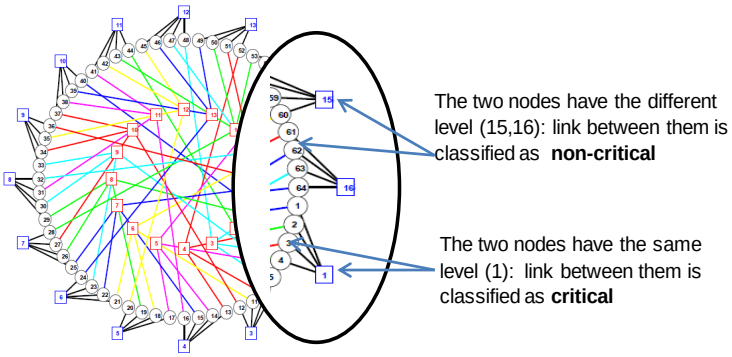


Figure 3.11: HyperFlatNet links classification.

3.3.3.2 Critical non cluster links

Let R be the connection matrix for a given network, and c be the set of its clusters $c = \{c_1, c_2, .., c_{ct}\}$, where ct is the total number of clusters in the network. The idea of Algorithm 7 is to find the intermediate links that connect each cluster with its next neighbor. *GetLink* computes the link that interconnects two neighbor clusters. Figure 3.12 presents an example of HyperFlatNet critical non cluster links .

Algorithm 7 Critical non cluster links (R)

for $c_i \rightarrow 1$ to c_{ct} **do**
  $l \leftarrow Getlink(c_i, c_{i+1})$
  $N_c \leftarrow [N_c \;\; l]$
end for
return N_c

3.3.3.3 Links deactivation algorithm

Let R be the connection matrix for a given network and n the number of port per switch. Algorithm 8 deactivates links from the Uncritical links of each cluster (denoted by L_d), and generates the updated matrix of deactivated links (denoted by R_d).

Algorithm 8 Dynamic closing ports

function $DeactivateLinks(M_k$,R,n,$\gamma)$

Input:

R is the network connection matrix.

γ is the number of communication per threshold.

Output:

N_c is the critical non cluster links

L_c is the critical cluster links

R_{up} the updated connection matrix.

M_v the communication vector

L_d links deactivate

$N_c \leftarrow GetCriticalNonClusterLinks(R)$

$L_c \leftarrow GetCriticalClusterLinks(R)$

$R_{up} \leftarrow poweron(N_c, L_c)$

for each period T **do**

  for each node i **do**

    $M_v(i) = \sum_{j=1}^{N} M(i,j) + \sum_{j=1}^{N} M(j,i)$

    Find $(M_v(i) \leq \gamma)$

  end for

  $L_d = sum(M_v \leq \gamma)$

end for

$R_{up} \leftarrow poweroff(L_d)$

return (R_{up})

end function

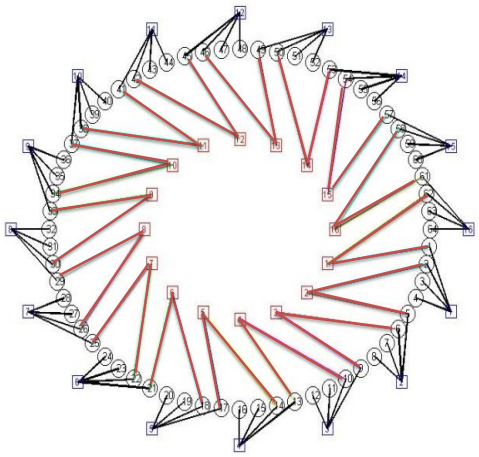


Figure 3.12: HyperFlatNet critical non cluster links.

3.3.3.4 Routing algorithm

The routing Algorithm 11 is proposed for packets transmission in the new network. We propose *linkState* and *IntraRouting* (Algoritm 9) for intra-cluster$_k$, and *localRout* for inter-cluster$_b$ routing. In a cluster$_k$, the switch cluster knows the status of all the outgoing/incoming links in its cluster$_k$. k is the number of layers in the network. For example, when k is 1 or 2 and $n = 10$ a $HyperFlatnet_k$ has 10 or 1000 servers. In the case of a disabled link, Algorithm 10 proposes function *GetOutreachServer* to find the nearest outreach server S^{out} which becomes a new source to forward the packet S_{new}. Then, it uses function *IntraRouting* to forward the packets to destinations.

3.3.4 System performance

3.3.4.1 Period study

Let n_T be the total number of ports in a network, n_a is the total number of active ports, n_d is the number of ports being closed, n_c is the number of deactivation times and n_{up} is the number of activation times for ports in a period T. Figure 3.13 shows an example, where n_c is the number of

Algorithm 9 IntraRouting algorithm

procedure IntraRouting((S_2, S_1),(D_2, D_1))
    Input:
    Ω is the vector of directly connected clusters
    (S_2, S_1) is the source coordinates
    (D_2, D_1) is the destination coordinates
    Output:
    P is the path from the source to the destination
    $pref \leftarrow GetCommunPrefix(n_{d1}, n_{d2})$
    if n_{d1} and n_{d2} have the same $pref$ **then**
        n_{d1} and n_{d2} are known to be in the same cluster
        $Path \leftarrow localRout((S_2, S_1), (D_2, D_1))$
    else
        if n_{d1} and n_{d2} have the different $pref$ **then**
            Then n_{d1} and n_{d2} are in the different cluster
            $S1_{new} \leftarrow GetOutreachServer$
            $P \leftarrow IntraRouting((S_2, S1_{new}), (D_2, D_1))$
        end if
    end if
end procedure

Algorithm 10 GetOutreachServer ((R,C_s,S_2, S_1),(D_2, D_1)

V is the source row
Input:
R is the matrix connection
C_s is the source cluster
Output:
S_s^M is the set of outreach server
$C_s \leftarrow GetSourceCluster$
$C_s = R(S_2, :)$
for $i = 1$ to length C_s **do**
    Find the set of outreach server in C_s
    $S^{out} = (C_s(i)! = 0))$
    $S_s^{out} i = S^{out}$
end for
return (S_s^{out})

Algorithm 11 IntraRouting algorithm

```
1: function IntraRouting((R, S_s^out, N_c))
2:     Input:
3:     R is the matrix connection
4:     S_s^out is the set of outreach server
5:     Output:
6:     Path is the path from the source to the destination
7:     while (Pref = null) do
8:         Pref ← GetCommunPrefix(S_s^out, N_c)
9:     end while
10:    l ← Getlink(S_s^out, Pref)
11:    Path ← [l, N_c(Pref, D_2)]
12:    return (Path)
13: end function
```

transitions from 1 to 0, n_{up} is the number of transitions from 0 to 1. n_a is the number of 1 and n_d is the number of 0. In this example, we have n_c= 2, n_{up}=1 ,n_d=5, n_a=5

1	1	0	0	0	1	1	1	0	0

Figure 3.13: Example of n_c and n_{up}.

P_T is the total energy consumption for the original system. P'_T is the total energy consumption for the proposed model (the green area presents the consumed energy and the red area is the saved energy). P is the energy consumed by an active port. τ_d and τ_{up} are the required time for activation and deactivating a ports (as provided by the manufacturer) (Figure 3.14).

$$P_T = P \times T \times n_T \tag{3.22}$$

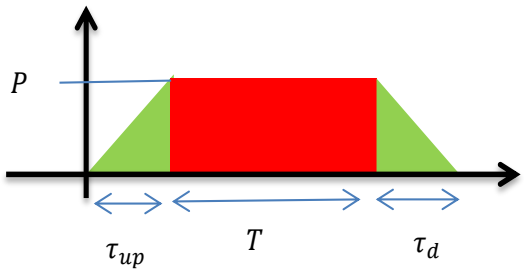


Figure 3.14: The consumed and saved energy during a period T.

$$P'_T = P \times T \times n_a + \left(n_c(\frac{\tau_d}{2}) + n_{up}(\frac{\tau_{up}}{2})\right) \tag{3.23}$$

So, the saving energy denoted by S_{eg} is

$$\begin{aligned} S_{eg} &= P_T - P'_T \\ &= P \times T \times n_d - P \times \left(n_c(\frac{\tau_d}{2}) + n_{up}(\frac{\tau_{up}}{2})\right) \end{aligned} \tag{3.24}$$

$S_{eg} > 0$ means that

$$T\frac{n_d}{n_c} > \frac{\tau_d}{2} + \frac{\tau_{up}}{2} \tag{3.25}$$

For a large number of nodes and periods, we assume that $n_c \approx n_{up}$,

$$S_{eg} = P \times \left(T \times n_d - (n_c(\frac{\tau_d}{2}) + (\frac{\tau_{up}}{2})\right) \tag{3.26}$$

So, to save energy the period T should such that:

$$T > \frac{n_c}{n_d}(\frac{\tau_d}{2} + \frac{\tau_{up}}{2}) \tag{3.27}$$

3.3.4.2 Energy consumption

The energy consumption in the network as presented in Figure 3.14 for one period is:

$$P'_T = P \times T \times n_a + \left(n_c(\frac{\tau_d}{2}) + n_{up}(\frac{\tau_{up}}{2})\right) \tag{3.28}$$

So, the total energy consumed over time can be written as:

$$Eg = \sum P'_T \tag{3.29}$$

For instance, for HyperFlatNet, the total number of ports that should be kept active to avoid the problem of disconnect nodes is $(n^3 + l_a)$. So, the energy consumed in one period is:

$$Eg^{'} = \sum P'_T + 2(n^3 + l_a)P \times T \tag{3.30}$$

For the original HyperFlatNet, the total number of active ports is $2n^3$. So, the total energy consumption is:

$$Eg = 2P \times T \times (2n^3) \tag{3.31}$$

If we compare the two systems we get:

$$\begin{aligned} E_{Saving} &= Eg - Eg^{'} \\ &= PT(4n^3 - l_a) - \left(n_c(\frac{\tau_d}{2}) + n_{up}(\frac{\tau_{up}}{2})\right) \end{aligned} \tag{3.32}$$

For a big time period T, the number of n_c, n_{up} ,n_d and n_a will be negligible, witch mean an increasing in energy saving.

3.3.5 Performance evaluation

3.3.5.1 Traffic pattern

We use HyperFlatNet network with 1000 servers to evaluate the impacts of power savings under different network loads, using All-to-All traffic pattern. The traffic matrix fellows a Poisson distribution with an exponentially distributed inter-arrival times with a rate parameter λ.

3.3.5.2 Simulations results

Energy saving

Figure 3.15 shows the energy consumption by the tested network under different correlation values a, and compare the results with the original HyperFlatNet. λ is varied from 10^{-2} and 10^2. Figure 3.16 on the other hand depicts the energy consumption where the network load is varied from 1 to 100 %. We remark that for a long period T, the tested network consumes less energy than HyperFlaNet. However, for a short period T, the correlation has a big impact on energy consumption. For instance, for $(T = 2, a = 2, \lambda = 1)$, the consumed energy for the tested network is 5.2×10^4. For $(T = 10, a = 20/100, \lambda = 1)$, the consumed energy is 4.2×10^4, while the original HyperFlatNet consumes 4.7×10^4. This means that HyperFlatNet consumes 5×10^3 less than the tested network for $(T = 2, a = 2, \lambda = 1)$. For a small period T, the energy consumption

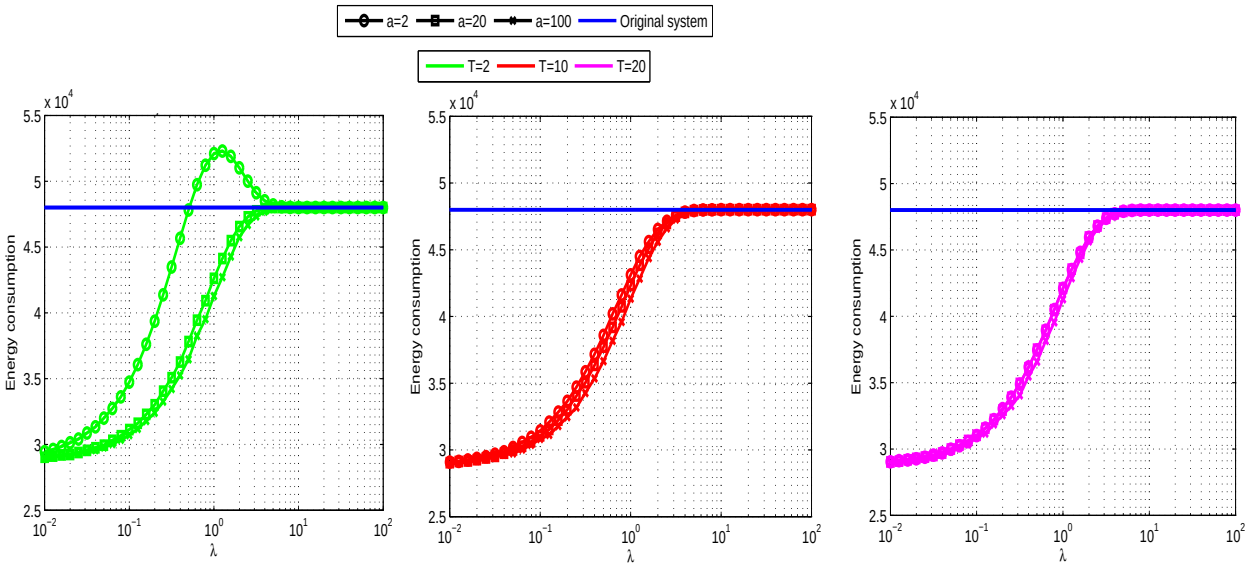


Figure 3.15: The energy consumption of the tested network under different correlation values (a) compared to the original system HyperFlatNet with a varied λ

induced by port status changing overpasses the energy saving. Additionally, a small period induces more delays for traffic to reach its destination. Hence, the length of period T is a critical parameter for a less traffic correlated system. Moreover, by increasing λ, the energy consumption of the tested network starts increasing from 2.8×10^4 when $\lambda = 0.01$ to attain $4.7 \times 10^4$when $\lambda = 100$. Also, when the network load increases, the energy consumption increases too. Consequently, for a traffic correlated system, the period T has less impact on energy saving than for a not correlated system.

Figure 3.17 shows the effect of period T on the energy consumption under different correlation values a compared with the original HyperFlatNet. The network load is varied from 1 to 100 %, while the period is fixed to 2, 10 and 20. We can observe that the period does not impact energy consumption for a large correlation value a, contrary to a system with a small correlation value a.

Figure 3.18 shows the energy consumption of the tested network under

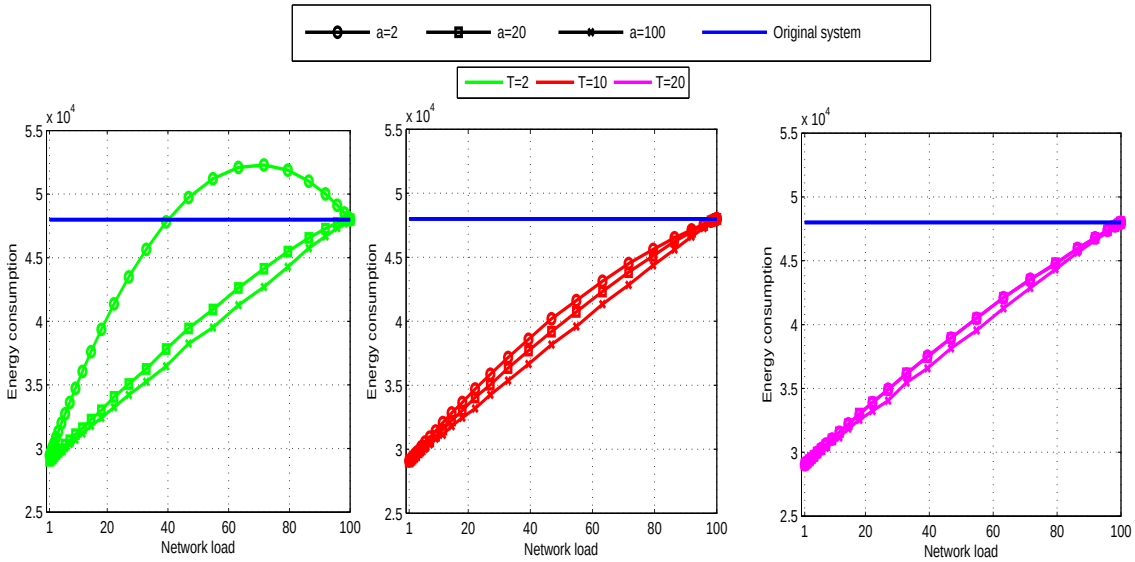


Figure 3.16: The energy consumption of the tested network under different correlation values (a) compared to the original system HyperFlatNet with a varied network load.

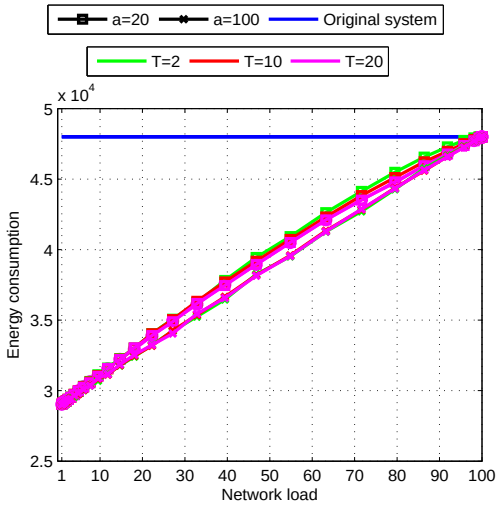


Figure 3.17: Effect of the period T on the energy consumption.

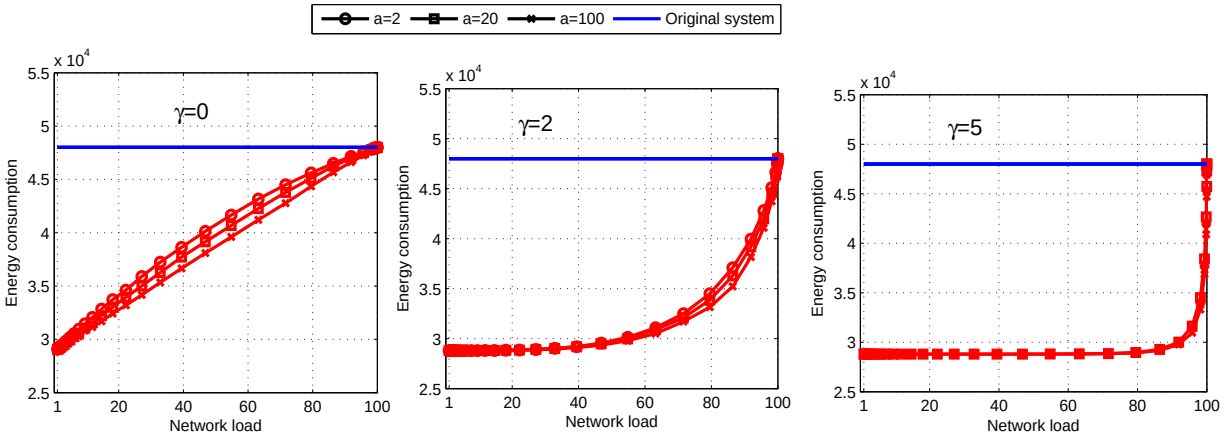


Figure 3.18: Effect of γ on the system energy consumption.

different correlation values a compared with the original HyperFlatNet. γ is varied between 0, 2, and 5. The network load is varied from 1 to 100%. We observe that the energy consumption deceases when γ increases. For example, for (network load=60%, a=2, $\gamma = 2$) the energy is 3.5×10^4 and 2.8×10^4 for (network load=60%, a=2, $\gamma = 5$). This means that the larger γ is the more is the energy saving.

Figure 3.19 proves the big impact of γ in the energy saving.

Average path length

Figure 3.20 shows the APL of the tested network for different correlation values a compared to the original system HyperFlatNet. λ is varied from 0 to 100. Figure 3.21 on the other hand depicts the APL of the tested network under different correlation values a compared with HyperFlatNet. The network load is varied from 1 to 100 %. We observe that whatever a is, for a long period T, the tested network has long APL than for a short period. For example, for $(T = 2, a = 2, \lambda = 10^{-2})$, the APL of the tested network is 7.5, while for $(T = 10, a = 100, \lambda = 0.01)$ the APL is 4.2. In

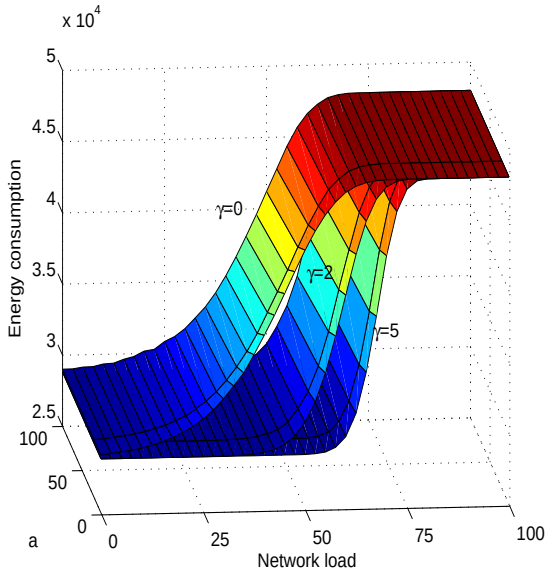


Figure 3.19: Effect of γ on the system energy consumption (3D) for T=2

fact, for a small period T, the state of ports follows closely the traffic state (ports will be disabled for a low traffic and activated for a high traffic). The same situation happens with large traffic correlation values a even for a long T . Similarly for λ, when the network load increases, the APL of the tested network decreases too. In fact, for a high traffic load, the tested network reduces the number of closed ports, and converges to the original HyperFlaNet . Consequently, its APL gets closer to the APL of HyperFlatNet.

Figure 3.22 shows the effect of the correlation on APL. We can see that for the same period, the correlation reduces largely the APL. In fact, for a correlated system, traffic matrices during a time period do not change much from the traffic matrix at the beginning of the period. Consequently, the APL will not highly increase compared with HyperFlatNet.

Figure 3.23 shows the effect of γ on the APL in the tested network. Although, γ improves energy saving as it increases, it also increases the APL. In fact, a larger γ results in a higher number of closed ports in HyperFlaNet, consequently resulting in more energy saving. However, for a small γ, the number of active ports increases and the APL decreases.

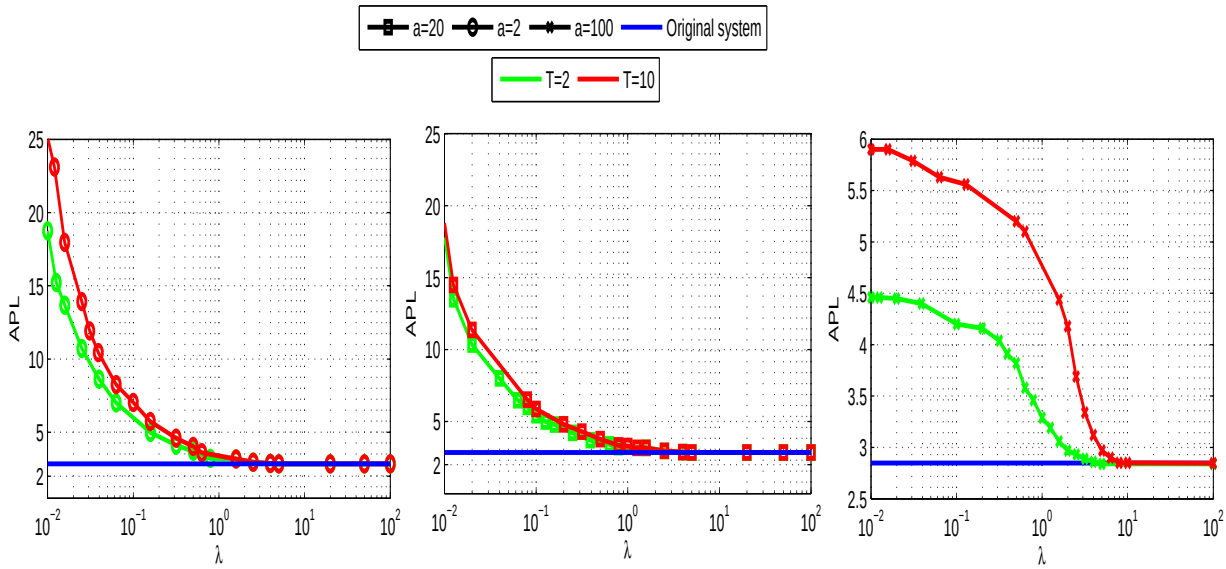


Figure 3.20: The APL of the tested network under different correlation values a, compared with the original HyperFlatNet network for different values of λ

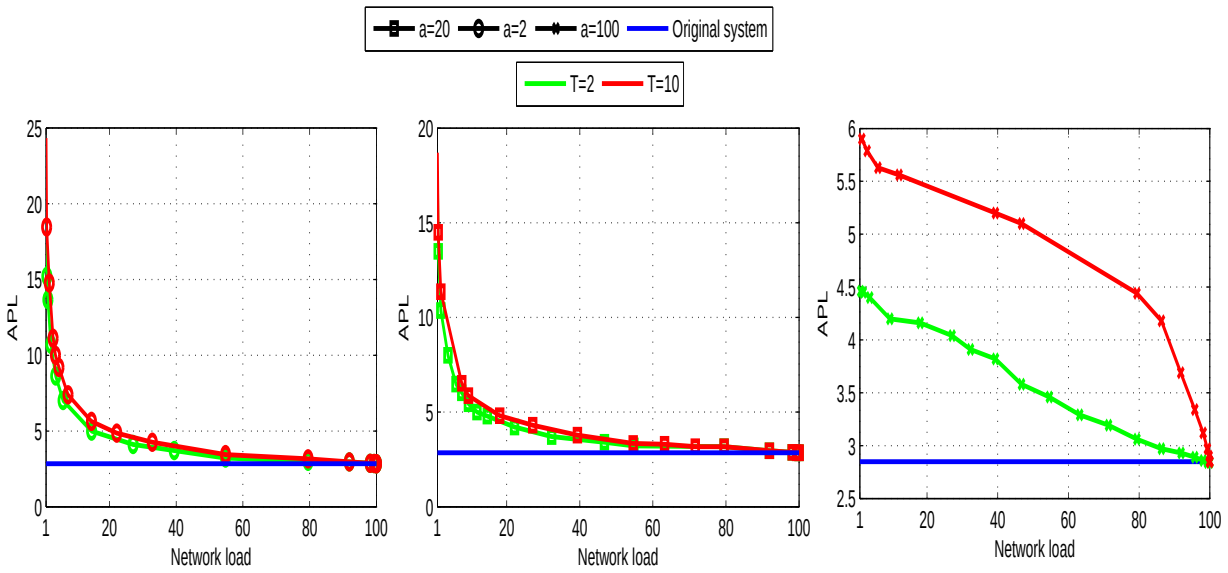


Figure 3.21: The APL of the tested network under different correlation values a compared with the original HyperFlatNet network with a varied network traffic

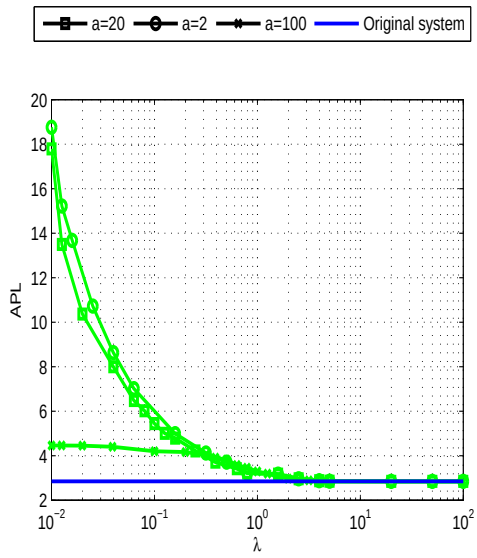


Figure 3.22: The APL of the tested network for different values of λ and correlation values a, when T=2

3.4 Cost reduction

The cost is one of the most critical parameters in designing DCs topologies. The proposed VacoNet presents a very interesting solution for companies thanks to its physical structure algorithm and the number of ports per switch selection algorithm. VacoNet increases the efficiency and improves asset utilization.

3.4.1 Switches cost

The cost of switches can be computed as:

$$\begin{aligned} Cost(switches) &= \sum_{i=1}^{n} p_i \times N_{sw} \\ &= n \times N_{sw} \end{aligned} \tag{3.33}$$

where N_{sw} denotes the total number of switches. Thanks to its special structure which optimizes the usage of switches, only the needed number of

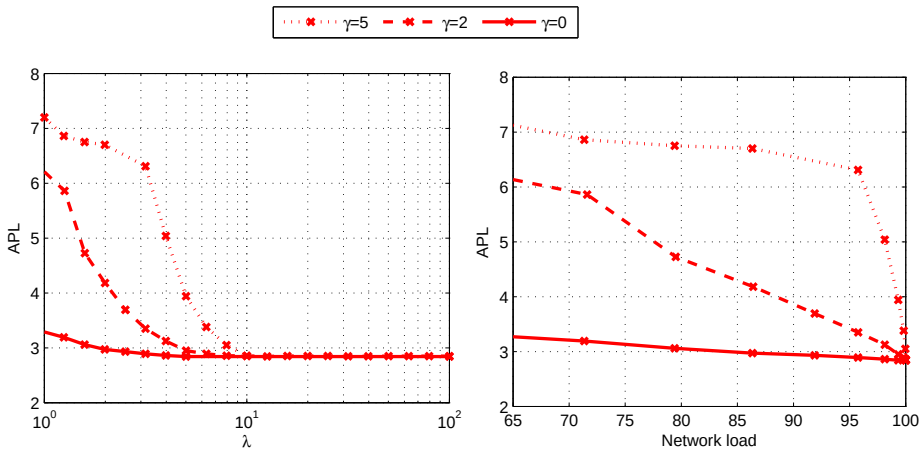


Figure 3.23: Effect of γ on the APL of tested network (T=10, a=100)

servers will be used, which reduces largely the network's cost. If we assume that the price of a switch in proportional to number of its port, according to Figure 3.24, for 3500 servers, the cost of switches would be 3.15×10^6 for VacoNet, 3.686×10^6 for FlatNet, 5.53×10^6 for BCube , and 8.789×10^6 for FatTree. This means that VacoNet decreases the cost of switches by 14.27%, 43% and 64.15% respectively.

3.4.2 Cabling cost

The cabling cost can be estimated as:

$$Cost(Cables) = N_{cb} \times C_{cb} \tag{3.34}$$

where C_{cb} and N_{cb} denote the cost per cable and the total number of cables respectively. The total number of used cables is computed as:

$$N_{cb} = n \times N_{sw} \tag{3.35}$$

Hence, the total cost $Cost_T$ of a topology can be expressed as:

$$\begin{aligned} Cost_T &= Cost(Cables) + Cost(switches) \\ &= N_{cb} \times C_{cb} + n \times N_{sw} \\ &= n \times C_{cb} \times N_{sw} + n \times N_{sw} \\ &= n \times N_{sw}(C_{cb} + 1) \end{aligned} \tag{3.36}$$

So, the total cost is proportional to the number of switches. VacoNet reduces the number of switches compared with all existing topologies. According to Figure 3.24, VacoNet decreases switches cost by 14.66% , 43.19% 65.35%compared with FlatNet, BCube and FatTree, respectively.

3.4.3 Performance evaluation

3.4.3.1 Simulation results

Figure 3.24 shows switches cost for VacoNet compared with FlatNet, BCube, FatTree and ScalNet, where the number of servers is varied from 0 to 10000. It can be seen that VacoNet reduces largely this cost compared with the other topologies. In fact for 8200 servers, the switches cost for VacoNet is 7.32×10^6, where as for FlatNet it is 8.34×10^6, for BCube it is 1.08×10^7, for FatTree it is 0.9×10^7, and for FatTree it is 1.67×10^7, which represents more than 1020000 USD in cost saving for VacoNet.

Figure 3.25 shows the cabling cost of VacoNet compared with FlatNet, BCube, FatTree and ScalNet. The number of servers is varied from 0 to

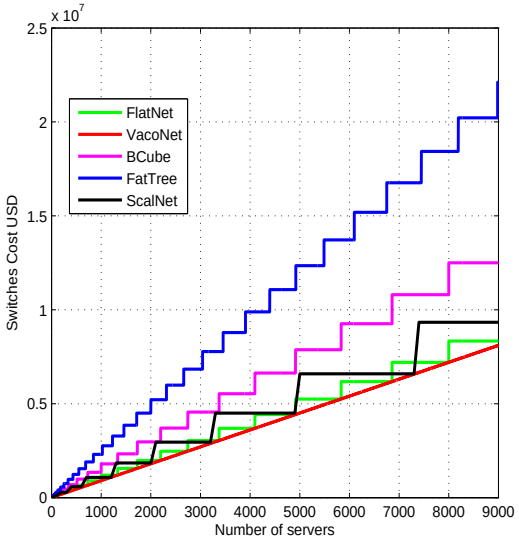


Figure 3.24: Switch Cost of VacoNet compared with FlatNet, BCube, FatTree and ScalNet.

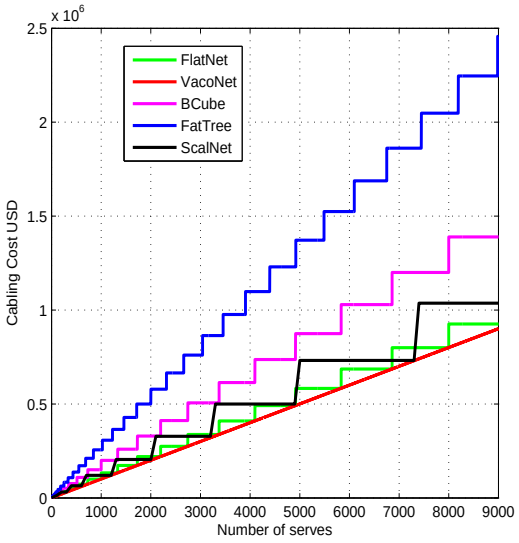


Figure 3.25: Cabling Cost of VacoNet compared with FlatNet, BCube, FatTree and ScalNet.

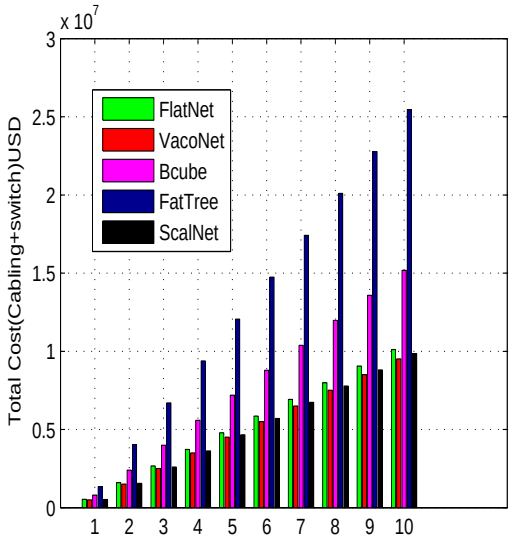


Figure 3.26: The histogram of the total cost for VacoNet compared with FlatNet, BCube, FatTree and ScalNet for 10000 servers.

10000. It can be seen that VacoNet reduces largely the cabling cost compared with the other topologies. In fact, for 8200 servers, cabling cost for VacoNet is 8.02×10^5, while it is 9.26×10^5 for FlatNet, 1.38×10^6 for BCube, and 2.24×10^6 for FatTree.

Figure 3.26 shows the total cost (including switches and cabling costs) for configurations of up to 10000 servers (for clarity, the number of nodes is scaled down by a factor of 1000 in the figure). As it can be seen, the difference between the different topologies is clear. In fact, the cost increases considerably as the number of servers increases, reaching 2.55×10^7, 1.52×10^7, 1.01×10^7, 9.51×10^6, 9.85×10^6 for FatTree, BCube, FlatNet, VacoNet and ScalNet respectively when the number of servers reaches 10000. Note that the difference between FlatNet and VacoNet is small compared with the other topologies, but still significant.

3.5 Conclusion

In this chapter, we proposed two approaches for static and dynamic energy saving DCs. Fisrt, we propose new topology called VacoNet that improves the DCs energy consumption and the cost by reducing largely the unused number of nodes and cables compared with the existing topologies. In addition, a new approach for DC recurssive topology has been proposed to maximize the energy saving. Our approach dynamically controls the number of active communication links by turning off and on ports in the network (switches ports and nodes ports). Simulation results prove the efficiency and feasibility of the proposed techniques for DCs.

Chapter 4

Conclusion and Future Research

4.1 Conclusion

Large-scale data centers form the core infrastructure support for the ever expanding cloud based services. However, the advantages of cloud computing come at a cost, the huge amount of energy data centers consume yearly. In this dissertation, we focused on enhancing the QoS and reducing cost and the energy consumption in Data Center. First, in chapter 2, a new efficient data center topology, called LCT is proposed combining the advantages of previous topologies while avoiding their limitations. LCT scales a data center to a mega level with only small-port count switches and small node degree. It strikes a compromise between the excessive scalability of DCell and high cost of BCube. Given an equal sized data center, the cost of LCT in terms of number of links and switches is roughly $\frac{1}{2}$ that of BCube, while still providing comparable overall performance. LCT is also fault-tolerant and load-balancing in nature due to its special structure design and the low-time-complexity routing protocol on top of its network topology. Moreover, we analyzed the Data Centers energy consumption and infrastructure cost in chapter 3. We proposed new topology called Va-

coNet. It is a new variable connection topology that connects any needed number of servers while reducing the unused materials. Detailed results with different configurations have proved that VacoNet can reduce the cost and the power respectively while providing a high network performance. Furthermore, new appraoch for dynamic energy saving is proposed. Our approach powers on and off network resources (switch ports) depending on the level of their involvement in the network traffic. The decision to close or open a port is based on a threshold value γ, such that the port is closed if the sum of the traffic generated by its connected node is less than γ, and opened otherwise. Simulation results proved the efficiency and feasibility of the proposed approaches.

4.2 Future research

Many perspectives may be taken into consideration in future works.

First, and with the rapid increasing number of cloud users and the quantity of data stored on cloud, greater security risks will be generated, especially on public cloud which sells services to anyone on the Internet. We will consider the problem of secure public cloud so that cloud users can access safely the resources of computing, storage and network by renting from cloud providers.

In the other hand, learning-based methods for security applications and traffic prediction are gaining popularity in the literature with the advents in machine learning techniques. We will consider these methods in obtaining traffic information to manage the port changing management for more DCs energy saving. Also, we will explore this information to secure the data of cloud users.

Chapter 5

Publications

5.1 Journal Paper

Accepted journal papers

1. Zina Chkirbene, Sebti Foufou , Ridha Hamila, Z Tari, AY Zomaya " LaCoDa: Layered connected topology for massive data centers," published in Journal of Network and Computer Applications
   http://dx.doi.org/10.1016/j.jnca.2017.01.020

Submitted journal papers

1. Zina Chkirbene, Rachid Hadjidj, Sebti Foufou, Ridha Hamila, Ibrahim Khalil "VacoNet: Varied Connected Topology For High Energy Efficient and Cost Reduction In Data Center Networks," submitted to Journal of Network and Computer Applications.

2. Zina Chkirbene, Rachid Hadjidj, Sebti Foufou, Ridha Hamila, "LaScaDa: A Novel Scalable Topology for Data Center Network based on Layered Linked Clusters Maximization Algorithm," Submitted to IEEE/ACM Transactions on Networking.

3. Zina Chkirbene, Ala Gouissem, Rachid Hadjidj, Sebti Foufou, Ridha Hamila, " Energy Saving for Data Centers Topologies Using Time Traffic Correlation and Port Activation management," to be submitted.

5.2 Conference papers

Accepted conference papers

1. Zina Chkirbene, Sebti Foufou, Ridha Hamila, Mounir Hamdi and Ridha Hamila, "Hyper-Flatnet: A novel network architecture for data centers," 2015 IEEE International Conference on Communication Workshop (ICCW), London, 2015, pp. 1877-1882.*doi* : 10.1109/*ICCW*.2015.7247454

2. Zina Chkirbene, Sebti Foufou, Ridha Hamila, "ScalNet: A Novel Network Architecture for Data Centers," 2015 IEEE Globecom Workshops (GC Wkshps), San Diego, CA, 2015, pp. 1-6. doi: 10.1109/GLOCOMW.2015.7414092

3. Zina Chkirbene, Sebti Foufou, Ridha Hamila, "VacoNet: Variable and connected architecture for data center networks," 2016 IEEE Wireless Communications and Networking conference, Doha, 2016, pp. 1-6.
doi: 10.1109/WCNC.2016.7564867

4. Zina Chkirbene, Sebti Foufou, Ridha Hamila, "Adaptative Network Topology for Data Centers," Accepted in ARC 2016: First winner in the ICT pillar.

5. Zina Chkirbene, Sebti Foufou, Ridha Hamila, "Integrating Variability Management in Data Center Networks," accepted in 2017 IEEE Wireless Communications and Networking Conference

Submitted conference papers

1. Zina Chkirbene, Sebti Foufou, Ridha Hamila, "A Novel Cluster-Based Energy Efficient Routing for Data Centers Network," to be submitted.

2. Zina Chkirbene, Sebti Foufou, Ridha Hamila, "Optimization on Ports Activation towards Energy Efficient Data Center Networks", to be submitted.

3. Zina Chkirbene, Ala Gouissem, Sebti Foufou, Ridha Hamila, "Reducing Energy Consumption for Data center Network," to be submitted.

Bibliography

[1] A. Carter, "Do it green: media interview with michael manos." http://edge.technet.com/Media/Doing-IT-Greel. xiv

[2] L. Rabbe, "Powering the yahoo! network'." http://yodel.yahoo.com/2006/11/27/powering-the-yahoo-networl. xiv

[3] M. Bari, R. Boutaba, R. Esteves, L. Granville, M. Podlesny, M. Rabbani, Q. Zhang, and M. Zhani, "Data center network virtualization: A survey," *Communications Surveys Tutorials, IEEE*, vol. 15, pp. 909–928, Second 2013. xiv

[4] T. Wang., Z. Su., Y. Xia., J. Muppala., and M. H. ., "Designing efficient high performance server-centric data center network architecture ," *computer networks*, January 2015. xiv

[5] P. Ruiu, A. Bianco, C. Fiandrino, P. Giaccone, and D. Kliazovich, "Power comparison of cloud data center architectures," in *2016 IEEE International Conference on Communications (ICC)*, pp. 1–6, May 2016. xiv

[6] E. Baccour, S. Foufou, and R. Hamila, "Ptnet: A parameterizable data center network," in *2016 IEEE Wireless Communications and Networking Conference*, pp. 1–6, April 2016. xiv

[7] S. Stroh., G. Schröder., and F. Gröne., "Keeping the Data Center Competitive Six Levers for Boosting Performance, Reducing Costs, and Preparing for an On-Demand World," nov 2012. xiv, 14, 20

[8] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," *SIGCOMM 2008 conference on Data communication*, pp. 63–74, August 2009. xiv, 7, 17

[9] D. Li, C. Guo, H. Wu, K. Tan, Y. Zhang, and S. Lu, "Ficonn: Using backup port for server interconnection in data centers," in *INFOCOM 2009, IEEE*, pp. 2276–2285, April 2009. xiv, 8

[10] C. Guo, H. Wu, K. Tan, L. Shi, Y. Zhang, , and S. Lu, "Dcell: A scalable and fault-tolerant network structure for data centers," *SIGCOMM 2008 on Data communication*, pp. 75–86, August 2008. xiv, 8, 67

[11] C. Guo, H. Wu, K. Tan, K. Shi, Y. Zhang, and S. Lu, "Bcube: A high performance, server-centric network architecture for modular data centers," *SIGCOMM 2009 on Data communication*, pp. 63–74, August 2009. xiv, 8, 67

[12] T. Wang, Z. Su, Y. Xia, Y. Liu, J. Muppala, and M. Hamdi, "Sprintnet: A high performance server-centric network architecture for data centers," in *Communications (ICC), 2014 IEEE International Conference on*, pp. 4005–4010, June 2014. xiv, 43

[13] A. Beloglazov, R. Buyya, Y. C. Lee, and A. Y. Zomaya, "A taxonomy and survey of energy-efficient data centers and cloud computing systems," *CoRR*, vol. abs/1007.0066, 2010. xiv

[14] R. Buyya, A. Beloglazov, and J. H. Abawajy, "Energy-efficient management of data center resources for cloud computing: A vision, architectural elements, and open challenges," *CoRR*, vol. abs/1006.0308, 2010. xiv

[15] E. Baccour, S. Foufou, R. Hamila, and M. Hamdi, "wflatnet: Introducing wireless in flatnet data center network," in *2015 IEEE Globecom Workshops (GC Wkshps)*, pp. 1–6, Dec 2015. xv

[16] "Cisco, Cisco Nexus 7000 Series Switches." http://www.cisco.com/c/en/us/products/collateral/switches/nexus-7000-10-slot-switch/Data Sheet C78-437762.html, 2013. 2

[17] Huawei, " CloudEngine 12800 Series High-Performance Core Switches, ." http://www.huaweienterpriseusa.com/products/network/switches/. 2

[18] Ruijie, "Ruijie RG-N18000 Series Data Center Switch." http://www.ruijie.com.cn/product/Switches/data-center-switch/RG-N18000, 2013. 2

[19] Arista, " Arista 7500 series data center switch." http://www. aristanetworks.com/media/system/pdf/Datasheets/7500 Datasheet.pdf, 2014. 2

[20] " Cisco Nexus 3064 Series Switches." "http://www.cisco.com/c/en/us/products/collateral/switches/nexus-3000-series-switches/data sheet c78-651097.htm. 2

[21] Arista, " Arista 7050QX series data center switch." http://www.arista.com/assets/data/pdf/7050QX QuickLook.pdf, 2016. 2

[22] Lenovo, "ThinkServer RD630 Rack Server." http://shop.lenovo.com/us/en/servers/thinkserver/racks/rd630/, 2013. 2

[23] IBM, "IBM System x3650 M4." http://www.redbooks.ibm.com/abstracts/tips0850.html, 2014. 2

[24] Huawei, "Tecal BH640 V2 Blade Server." http://enterprise.huawei.com/en/products/itapp/server/e-series-blade-server/hw-149397.htm, 2013. 2

[25] Dell, "PowerEdge M820 Blade Server." http://www.dell.com/us/business/p/poweredge-m820/pd, 2013. 2

[26] E. Schonfeld, "Where are all the google data centers? ." http://techcrunch.com/2008/04/11/where-are-all-the-google-datacenters/, 2008. 3

[27] " How Microsoft Manages Its Cloud Infrastructure at a Huge Scale." "http://channel9.msdn.com/Events/TechEd/NorthAmerica/2011/COS211. 3

[28] Microsoft, " Microsoft Data Centers." http://www.globalfoundationservices.com, 2014. 3

[29] R. LeMay, "Amazon confirms Sydney CDN node." http://delimiter.com.au/2012/06/20/amazon-confirms-sydney-cdn-node/, 2012. 3

[30] A. Carrega, S. Singh, R. Bolla, and R. Bruschi, "Applying traffic merging to datacenter networks," in *Proceedings of the 3rd International Conference on Future Energy Systems: Where Energy, Computing and Communication Meet*, e-Energy '12, (New York, NY, USA), pp. 3:1–3:9, ACM, 2012. 4

[31] B. Heller, S. Seetharaman, P. Mahadevan, Y. Yiakoumis, P. Sharma, S. Banerjee, and N. McKeown, "Elastictree: Saving energy in data center networks," in *Proceedings of the 7th USENIX Conference on Networked Systems Design and Implementation*, NSDI'10, (Berkeley, CA, USA), pp. 17–17, USENIX Association, 2010. 4, 5

[32] Y. Shang, D. Li, and M. Xu, "Energy-aware routing in data center network," in *Proceedings of the First ACM SIGCOMM Workshop on Green Networking*, Green Networking '10, (New York, NY, USA), pp. 1–8, ACM, 2010. 4, 5

[33] D. Li, Y. Yu, W. He, K. Zheng, and B. He, "Willow: Saving data center network energy for network-limited flows," *IEEE Transactions on Parallel and Distributed Systems*, vol. 26, pp. 2610–2620, Sept 2015. 5

[34] "https://www.energystar.gov/products/low carbon it campaign/12 ways save energy data center." 6

[35] I. n. Goiri, K. Le, T. D. Nguyen, J. Guitart, J. Torres, and R. Bianchini, "Greenhadoop: Leveraging green energy in data-processing frameworks," in *Proceedings of the 7th ACM European Conference on Computer Systems*, EuroSys '12, (New York, NY, USA), pp. 57–70, ACM, 2012. 6

[36] K. K. Nguyen, M. Cheriet, M. Lemay, M. Savoie, and B. Ho, "Powering a data center network via renewable energy: A green testbed," *IEEE Internet Computing*, vol. 17, pp. 40–49, Jan 2013. 6

[37] M. Arlitt, C. Bash, S. Blagodurov, Y. Chen, T. Christian, D. Gmach, C. Hyser, N. Kumari, Z. Liu, M. Marwah, A. McReynolds, C. Patel, A. Shah, Z. Wang, and R. Zhou, "Towards the design and operation of net-zero energy data centers," in *13th InterSociety Conference on Thermal and Thermomechanical Phenomena in Electronic Systems*, pp. 552–561, May 2012. 6

[38] V. George, H. Zhang, and J. Rabaey, "The design of a low energy fpga," in *Proceedings of the 1999 International Symposium on Low Power Electronics and Design*, ISLPED '99, (New York, NY, USA), pp. 188–193, ACM, 1999. 6

[39] D. Meisner, B. T. Gold, and T. F. Wenisch, "Powernap: Eliminating server idle power," *SIGARCH Comput. Archit. News*, vol. 37, pp. 205–216, Mar. 2009. 6

[40] E. N. Elnozahy, M. Kistler, and R. Rajamony, "Energy-efficient server clusters," in *Proceedings of the 2Nd International Conference on Power-aware Computer Systems*, PACS'02, (Berlin, Heidelberg), pp. 179–197, Springer-Verlag, 2003. 6

[41] K. K. Rangan, G.-Y. Wei, and D. Brooks, "Thread motion: Fine-grained power management for multi-core systems," *SIGARCH Comput. Archit. News*, vol. 37, pp. 302–313, June 2009. 6

[42] J. Leverich, M. Monchiero, V. Talwar, P. Ranganathan, and C. Kozyrakis, "Power management of datacenter workloads using per-core power gating," *IEEE Comput. Archit. Lett.*, vol. 8, pp. 48–51, July 2009. 6

[43] H. David, C. Fallin, E. Gorbatov, U. R. Hanebutte, and O. Mutlu, "Memory power management via dynamic voltage/frequency scaling," in *Proceedings of the 8th ACM International Conference on Autonomic Computing*, ICAC '11, (New York, NY, USA), pp. 31–40, ACM, 2011. 6

[44] Z. Chkirbene, S. Foufou, and R. Hamila, "Integrating variability management in data center networks," in *2016 IEEE Wireless Communications and Networking Conference*, pp. 1–6, April 2017. 6

[45] A. Singla, A. Singh, K. Ramachandran, L. Xu, and Y. Zhang, "Proteus: A topology malleable data center network," in *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, Hotnets-IX, (New York, NY, USA), pp. 8:1–8:6, ACM, 2010. 6

[46] L. Huang, Q. Jia, X. Wang, S. Yang, and B. Li, "Pcube: Improving power efficiency in data center networks," in *2011 IEEE 4th International Conference on Cloud Computing*, pp. 65–72, July 2011. 6

[47] L. Liu, H. Wang, X. Liu, X. Jin, W. B. He, Q. B. Wang, and Y. Chen, "Greencloud: A new architecture for green data center," in *Proceedings of the 6th International Conference Industry Session on Autonomic Computing and Communications Industry Session*, ICAC-INDST '09, (New York, NY, USA), pp. 29–38, ACM, 2009. 6

[48] G. Wu, M. Tang, Y.-C. Tian, and W. Li, "Energy-efficient virtual machine placement in data centers by genetic algorithm," in *Proceedings of the 19th International Conference on Neural Information Processing - Volume Part III*, ICONIP'12, (Berlin, Heidelberg), pp. 315–323, Springer-Verlag, 2012. 6

[49] T. Yang, Y. C. Lee, and A. Y. Zomaya, "Energy-efficient data center networks planning with virtual machine placement and traffic configuration," in *2014 IEEE 6th International Conference on Cloud Computing Technology and Science*, pp. 284–291, Dec 2014. 6

[50] A. Greenberg, J. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. Maltz, P. Patel, and S. Sengupta, "Vl2: a scalable and exible data center network," *SIGCOMM Computer Communication*, pp. 51–62, August 2009. 7

[51] D. Lin, Y. Liu, M. Hamdi, and J. Muppala, "Hyper-bcube: A scalable data center network," in *2012 IEEE International Conference on Communications (ICC)*, pp. 2918–2923, June 2012. 8, 11, 67

[52] R. Mysore, A. Pamporis, N. Farrington, N. Huang, P. Miri, R. Radhakrishnan, V. Subramanya, and A. Vahdat, "PortLand: A Scalable, Fault-Tolerant Layer 2 Data Center Network Fabric," *SIGCOMM2009 on Data communication*, pp. 39–50, August 2009. 8

[53] D. Lin, Y. Liu, M. Hamdi, and J. Muppala, "Flatnet: Towards a flatter data center network," in *Global Communications Conference (GLOBECOM), 2012 IEEE*, pp. 2499–2504, Dec 2012. 11, 67

[54] X. Yin, S. Yoo, P. Mejia, Proietti.R., and V. Akella, "DOS – A Scalable Optical Switch for Datacenters," *ACM ANCS'10*, pp. 63–74, October 2010. 12

[55] K. Xia, M. Kao, Y.and Yang, and J. Chao, "Petabit Optical Switch for Data Center Networks," *Technical Report*, 2010. 12

[56] Y. Liu., J. Muppla., M. Veeraghavan., D. Lin., and M. Hamdi., "Data center network." http://www.amazon.ca/Data - center - Network, 2013. 15

[57] D. Lin., Y. Liu., M. Hamd.i, and J. Muppala., "Hyper-BCube: A Scalable Data Center Network," 2012. 17

[58] L. Popa., S. Ratnasamy., G. Iannaccone., A. Krishnamurthy., and I. Stoica., "A cost comparison of datacenter network architectures," in *ACM New York, NY, USA*, 2010. 18

[59] Z. Chkirbene, S. Foufou, M. Hamdi, and R. Hamila, "Hyper-flatnet: A novel network architecture for data centers," in *2015 IEEE International Conference on Communication Workshop (ICCW)*, pp. 1877–1882, June 2015. 34, 67, 68

[60] Z. Chkirbene, S. Foufou, M. Hamdi, and R. Hamila, "Scalnet: A novel network architecture for data centers," in *2015 IEEE Globecom Workshops (GC Wkshps)*, pp. 1–6, Dec 2015. 36

[61] Z. Chkirbene, S. Foufou, R. Hamila, T. Zahir, and A. Y. Zomaya, "Lacoda: Layered connected topology for massive data centers," in *Journal of Network and Computer Applications*, Feb 2017. 40, 67

[62] R. Qi, W. Liu, J. Gutierrez, and A. W. Malik, "Crash me if you can: Rethinking sustainable data center networking from a topological perspective," in *2016 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pp. 566–571, April 2016. 68