Winter 3-2-2017

Power-Aware Datacenter Networking and Optimization

Qing Yi
Portland State University

10.15760/etd.5358

Power-Aware Datacenter Networking and Optimization

by

Qing Yi

A dissertation submitted in partial fulfillment of the
requirements for the degree of

Doctor of Philosophy
in
Computer Science

Dissertation Committee:
Suresh Singh, Chair
Jingke Li
Fei Xie
Branimir Pejcinovic

Portland State University
2017

ABSTRACT

Present-day datacenter networks (DCNs) are designed to achieve full bisection bandwidth in order to provide high network throughput and server agility. However, the average utilization of typical DCN infrastructure is below 10% [8] for significant time intervals. As a result, energy is wasted during these periods. In this thesis we analyze traffic behavior of datacenter networks using traces as well as simulated models. Based on the insight developed, we present techniques to reduce energy waste by making energy use scale linearly with load. The solutions developed are analyzed via simulations, formal analysis, and prototyping. The impact of our work is significant because the energy savings we obtain for networking infrastructure of DCNs are near optimal.

A key finding of our traffic analysis is that network switch ports within the DCN are grossly under-utilized. Therefore, the first solution we study is to modify the routing within the network to force most traffic to the smallest number of switches. This increases the hop count for the traffic but enables the powering off of many switch ports. The exact extent of energy savings is derived and validated using simulations. An alternative strategy we explore in this context is to replace about half the switches with fewer switches that have higher port density. This has the effect of enabling even greater traffic consolidation, thus enabling even more ports to sleep. Finally, we explore a third approach in which we begin with end-to-end traffic models and incrementally build a DCN topology that is optimized for that model. In other words, the network topology is optimized for the potential use of the datacenter. This approach makes sense because, as other researchers have observed, the traffic in a datacenter is heavily dependent on the primary use of the datacenter.

A second line of research we undertake is to merge traffic in the analog domain prior to feeding it to switches. This is accomplished by use of a passive device we call a merge network. Using a merge network enables us to attain linear scaling of energy use with load regardless of datacenter traffic models. The challenge in using such a device is that layer 2 and layer 3 protocols require a one-to-one mapping of hardware addresses to IP (Internet Protocol) addresses. We overcome this problem by building a software shim layer that hides the fact that traffic is being merged. In order to validate the idea of a merge network, we build a simple mere network for gigabit optical interfaces and demonstrate correct operation at line speeds of layer 2 and layer 3 protocols. We also conducted measurements to study how traffic gets mixed in the merge network prior to being fed to the switch. We also show that the merge network uses only a fraction of a watt of power, which makes this a very attractive solution for energy efficiency.

In this research we have developed solutions that enable linear scaling of energy with load in datacenter networks. The different techniques developed have been analyzed via modeling and simulations as well as prototyping. We believe that these solutions can be easily incorporated into future DCNs with little effort.

DEDICATION

To my family.

ACKNOWLEDGMENTS

First and foremost, I want to thank my advisor, Professor Suresh Singh. As one of his Ph.D. students, I have been very fortunate to conduct research under his guidance for the past four years. Professor Singh has been not only an excellent and inspirational advisor in my research field, but also the best mentor in my professional life. I deeply appreciate all his contributions of time, ideas, and funding to make my Ph.D. studies productive and stimulating.

For this dissertation, I would like to thank my reading committee, Professor Li Jingke, Professor Xie Fei and Professor Branimir Pejcinovic, for their time, interest, and valuable comments and insightful questions. In particular, I am grateful for the many conversations I had with Professor Xie Fei regarding my research, which gave me new perspective in many areas I had not previously considered.

I also want to thank the National Science Foundation that made my Ph.D. work possible. My time at Portland State was made enjoyable in large part due to the many friends who have become part of my life. I am grateful for the time spent with my fellow Ph.D. research students and friends, Farnoosh Moshifatemi, Cong Kai, Yang Zhengkun, Simon Niklaus, Wang Qin, Cuong Nguyen and Lin Bin, and for many other people and good memories.

Lastly, I would like to thank my family for all their love and support. For my parents who raised me with a love of math and science and supported me in all my pursuits. And most of all for my loving and encouraging husband whose

faithful support and patience throughout my doctoral program is so appreciated. In addition, these acknowledgments would not be complete if I did not mention my sons, Kevin and Ryan. They are great sources of love and support for my scholarly endeavor. Their thirsty of knowledge always gives me inspiration and strength to persist during this long journey. Thank you all!

Qing Yi
Portland State University

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

INTRODUCTION

Datacenters have experienced a substantial growth in recent years due to the growing popularity of Internet services and the widespread adoption of cloud computing. From sever rooms of small-sized organizations, to enterprise datacenters and the server farms that run cloud computing services, datacenters have become the backbone of the economy. Datacenters achieve economies of scale with large numbers of servers. Many Internet service providers and cloud computing service providers have built very large-sized datacenters, often housing more than 50,000 or more servers each, at geographically distributed locations. For example, Google has built more than 30 datacenters in 15 countries with a total of approximately 900,000 servers [1]. Amazon has 11 cloud regions across the world. Each region has multiple sets of datacenters, with a typical facility containing 50,000 to 80,000 servers. A conservative estimate puts Amazon at over 1.5 million servers globally.

These gigantic datacenters consume a significant amount of electricity. In 2010, the total electricity consumption by datacenters was 235.5$BkWh$ (Billion kilo Watt hours) [58], which accounts for about 1.3% of the total electricity consumption of the entire world. In 2013, U.S. datacenters consumed an estimated 91 BkWh of electricity. This is the equivalent annual output of 34 large (500-megawatt) coal-fired power plants. As shown in Figure 1.1, datacenter electricity consumption is projected to continue to increase. Energy costs currently comprise approximately

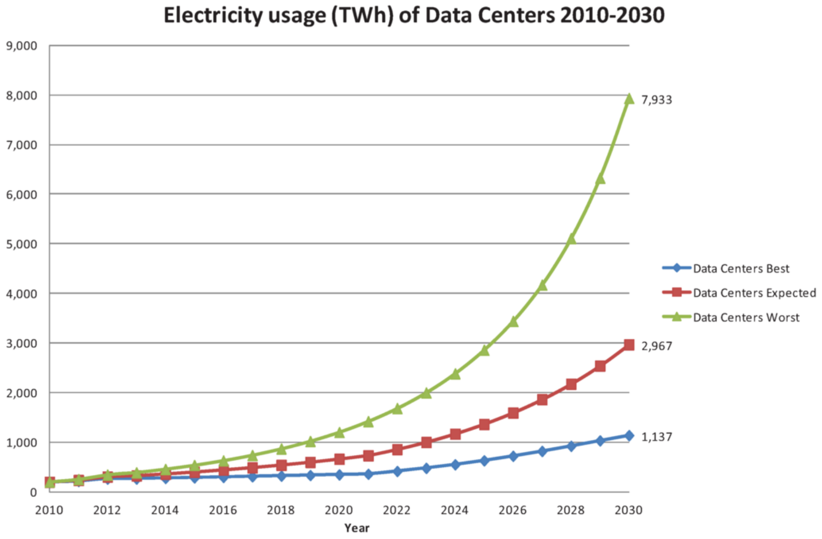


Figure 1.1. Global electricity demand of datacenters 2010 - 2030 [15].

70% of the operations costs of the average datacenter facility [72]. As a result, energy cost and energy availability have become some of the top concerns in planning future datacenter operations.

However, despite the rising energy consumption and limits on electric power, the resource utilization of most of the datacenters is poor. An investigation of 5,000 servers in a datacenter during a six-month period shows that the average CPU utilization is between 10% – 50% of the peak load [35]. Low server utilization also leads to underutilization of network infrastructure interconnecting servers. Studies show the average link utilization in the lower level of the network is only at 8% for 95% of the time [20]. Energy concerns and underutilization of IT resources are driving the industry to find ways of improving energy efficiency of datacenters. New technologies have driven up power capacity requirements in IT, mainly in server and storage-related operations. For instance, Google builds their own customized energy-efficient server for its datacenter use. Software solutions such as server virtualization have been proposed to improve the energy efficiency by reducing computing overhead.

The datacenter network has not traditionally been a major contributor to the power problems because of the relatively low consumption of power by the network compared to the overall power consumed in a datacenter. Power consumption requirements by the network have been relatively low because of the comparatively low-level of computing functions in switches and routers. Most estimates of networking infrastructure consumption range from 8 to 12 percent of the total power consumed by the entire datacenters. As the network has evolved to include higher levels of intelligence, however, its corresponding power requirements have continued to grow. In addition, with the improvement of sever and storage energy efficiency, networking components are expected to become an increasing user of datacenter energy.

1.1 MOTIVATION

A great deal of recent research has examined the question of improving server's energy efficiency (hardware and software) as well as developing better algorithms for distributed computing. However, the question of optimizing the power performance of the datacenter network has received far less attention. With the servers becoming more energy efficient, it is projected that the relative energy consumption of the network components will increase by up to 50% and become the predominant source of energy waste in datacenters. Thus, it is meaningful to consider how the datacenter network can be made more energy efficient. We can identify the following primary reasons for the energy inefficiency of datacenter networks:

- Over-provisioned Network Topology: The underlying assumption in datacenter network design is that full-bisection bandwidth needs to be supported at all times. This assumption has resulted in the use of dense network topologies such as the *fat-tree* network. While these topologies do provide high bandwidth as well as low latency, numerous measurements show that the links and switches are grossly under-utilized.

- Use of Legacy Switches: Datacenter network operators buy off the shelf switches from vendors such as Cisco with little thought to adapting the switch design to the specific needs of the datacenter. Indeed, the primary focus of switch development has been to make the interfaces faster and the switches more dense. There has been little effort at customized switches for the specific eco-system of a datacenter.

- Application Oblivious Topology Design: The philosophy of designing datacenter networks for the general case rather than the expected case causes network inefficiencies. For instance, datacenter job scheduling algorithms try to concentrate computation to within a single rack to minimize memory and disk access latencies. In these datacenters, there is no need to have high-bandwidth between racks as we do today.

- Unlinked Server and Network Energy savings: Today, servers in the datacenter are very energy-conscious and typically sleep for extended periods. However, the underlying network continues to remain fully powered on. This is clearly a waste of energy and linking the energy-conserving approach for servers to the underlying network will result in significant energy savings as well as little impact on performance.

In summary, current datacenter network architecture and network devices were never designed for energy efficiency. Network switches consume virtually the same energy whether they are forwarding packets or not. Furthermore, the architecture precludes any scaling of switch performance for energy efficiency. One approach that has been developed called IEEE 802.3az [7] changes the link rate of each interface on the switch based on load. However, as interfaces typically consume a tiny fraction of overall switch power, this is an insignificant saving.

The overall goal of this research is to develop datacenter networks whose energy cost scales linearly with load and which can support high network throughput.

In this research we will develop novel techniques to address the question of energy efficiency of datacenter networks as well as delivering high network bandwidth for distributed computations. We use a passive switching fabric to form a merge network that allows traffic from N links to be merged into fewer links into a switch. This allows us to either replace high port density switches with lower density switches or power off large parts of a switch. We examine the application this novel hardware element in the context of existing topologies and develop new topologies that exploit its properties. Furthermore, we consider replacing the datacenter network with this switching fabric. Thus, if we consider servers located on one rack, rather than using traditional switches, we can connect all the servers using a switching fabric composed of passive elements like multiplexers. Logically, this can be viewed as throwing away the network interfaces of switches and directly connecting the end-host network interfaces to the internal interconnection network found in each switch.

This research relies on detailed simulations using traffic models extracted from measurements reported in the literature as well as analysis to determine performance bounds. We examine metrics including energy cost of the new networks as compared to traditional networks, network reliability, network bisection bandwidth, and scalability.

1.2 BACKGROUND

In this section, we will lay the groundwork for some fundamental terminology and concepts. First, we will discuss the design and engineering of datacenter interconnection networks, including network topology, network architecture, and routing algorithms. Specifically, we will focus on the hierarchical network topologies that are widely used in production datacenters. Indeed, most of the work in the following chapters is based on hierarchical datacenter topologies. Second, we will review the emerging technologies in power management of network devices, and motivate our power-aware datacenter network research.

1.2.1 Datacenter Network Topology

Inside a datacenter, large number of servers are interconnected using specific datacenter network (DCN) structures. The design goals of the datacenter network are to provide high bandwidth, low latency and high throughput communications between servers through network infrastructure with low cost and high scalability. The network enables applications running on the servers to communicate and interoperate in an orchestrated and efficient way. The performance of interconnection networks plays an important role in improving the overall performance of datacenters.

Mega datacenters nowadays may contain tens of thousands of servers that are interconnected by network links and switches. A three-layer hierarchical model is widely adopted for designing a scalable internetwork for datacenters, in which switches are connected in a network structure consisting of *core*, *aggregation*, and *access* layers. Traditional hierarchical model is a tree topology with aggregation to lager-sized, higher-speed switches moving up the hierarchy. The access layer grants end servers access to the network. Switches in the access layer connect directly to servers, and are referred as edge switches. The access layer is in turn connected to the network aggregation layer, which aggregates the data received from the access layer switches before it is transmitted to the core layer for routing to its final destination. The switches in the core layer form the network backbone, providing a fabric for high-speed packet switching between aggregation switches. A single core switch, even with a density of hundreds of ports, limits the network size to a few thousand hosts. Also, one core switch only can provide limited bandwidth between servers in different racks, making it difficult for applications like MapReduce, which requires high intra-cluster bandwidth. Moreover, a single-rooted tree has problem of single point of failure, cause poor reliability. Therefore, many multi-rooted tree topologies with multiple core switches, i.e. *Clos*, are deployed for large-sized datacenters. A Multi-rooted tree topology can provide multiple paths between any pair of end hosts.

In the access layer and core layer, high-end switches with high port density and connection rate are used to provide high-speed data transmission. Due to the high costs of high-end switches, datacenter architects choose to oversubscribe the networks. The higher layers of the three-layer datacenter network are highly oversubscribed, which in turn causes poor network utilization, limiting the overall bisection bandwidth and overall network throughput. To address the problem of limited cross section bandwidth close to the roots, a *fat-tree* [11] (also known as *folded Clos*), was proposed for large-scale datacenter environments.

A *fat-tree* [11] is a multi-rooted tree topology that is a special instance of the Folded-Clos network. It has 'fatter' links in upper layers, that makes it a mesh-like network with full bisection-bandwidth. A *fat-tree* leverages off-the-shelf Ethernet switches and all the switches in different layers are identical, making it a cost-effective and easy-to-deploy solution for large-scale datacenters with tens of thousands of servers. *Fat-tree* topology is deployed in many datacenters. A k-ary *fat-tree* has k *pods*, each containing 2 layers of $\frac{k}{2}$ switches. Each edge switch in a *pod* is connected to $\frac{k}{2}$ hosts and $\frac{k}{2}$ aggregation switches. There are $\left(\frac{k}{2}\right)^2$ core switches, each of them has k ports connected to k *pods*. The *fat-tree* topology has great scalability. A k-ary *fat-tree* network can support $\frac{k^3}{4}$ hosts. Furthermore, the *fat-tree* topology has identical bandwidth at any bisection and each layer has the same aggregated bandwidth. Therefore, it can achieve full bisection bandwidth with 1:1 oversubscription ratio. Compared with other conventional tree-based topologies, a *fat-tree* network has less bandwidth bottleneck issues and can provide high bandwidth by interconnecting smaller commodity switches.

Recently, some relatively flat network architectures have been proposed to replace the traditional multi-layer enterprise network architecture for large datacenters to support virtualization. Virtualization is implemented in large cloud datacenters to improve efficiencies, and as a result, the network no longer connects only hardware blocks, but also interconnects virtual machines (VMs) and virtual storage volumes. Resources need to be dynamically reassigned from any point within the datacenter to another point. While the multi-layer architectures have high latencies and complex software, moving a VM not only affects the access switch configuration, it may require reconfiguration of the aggregation and core switches as well. Therefore, the multi-layered architecture is not considered efficient for VMs migration circumstance. However, flat datacenter network architectures can interconnect basic units such as virtual machines and virtual storage volumes across large, switched Ethernet fabrics.

1.2.2 Energy-Efficient Networking

The *fat-tree* topology provides a scalable and cost-effective solution for large-scale datacenters. A *fat-tree* is built from a large number of densely connected switches and can support any communication patterns with full bandwidth. However, it achieves the extra bandwidth by provisioning many redundant network links and switches, providing multiple paths between any two end servers. Richer connection can achieve high performance at peak network loads. However, the redundancy also causes waste of network resources, especially in the night hours when the network loading is extremely low. If the redundant network resources can be put in low power mode, it is possible to dynamically vary the number of active network elements to choose the necessary paths, thus to improve energy efficiency of datacenter networks.

Until recently, network interfaces and links were designed to stay awake all the time. During idle periods, the interfaces send frames periodically to provide synchronization as well as to serve as a keepalive mechanism. As a result, interfaces run at full power all the time even though there is no traffic. An increasing interest in networking energy efficiency led to the formation of the Energy Efficient Ethernet (IEEE802.3az) project in November 2006. The project task force considered many proposals for changes to the Ethernet interface standard to enable energy efficiency, and, at the same time, ensure backward compatibility and network robustness. The final IEEE802.3az Energy Efficient Ethernet standard (EEE) [7] was published in November 2010, representing the beginning of a change in networking architecture design. The core idea of the EEE standard is the low-power idle state (LPI state), first proposed by Intel [50]. All the EEE-supported PHY types in the IEEE 802.3az can configure an LPI state for the periods when there is no data sent from/to the interfaces. The EEE standard defines a signaling protocol for network devices to communicate with each other and indicates the power state of the link between them. The protocol uses an LPI signal that is a modification of the normal idle signal that is transmitted between data packets to indicate that the link can switch to sleep mode to minimize the power consumption of the device ports that connect from either side of the link. The transmitting port sends idle signals when it wishes to resume the fully functional link status. The EEE protocol awakens the link at any time, and there is no minimum or maximum sleep interval, which allows EEE to function effectively in the presence of unpredictable traffic. Therefore, by switching between higher power state (data mode) and lower power state (LPI mode) in response to whether data is flowing through it, a network device can reduce energy consumption when its utilization pattern consists of long periods of idleness.

1.3 CONTRIBUTIONS

This research proposes a framework of improving the energy-efficiency of datacenter networks. Most datacenter network topologies are designed to maximize cross-section bandwidth at minimal link and switch cost, to achieve high scalability, low latency, high throughput and low cost. Recently, however, energy efficiency has become a performance metric for datacenter networks. Al-Fares [11] shows that a *fat-tree (aka folded-Clos)* topology built from 1Gbps commodity Ethernet chips uses considerably less power than a hierarchical network composed of high-end, power inefficient 10 Gbps switches. Some other researchers reports that a *flattened-butterfly* topology itself is inherently more power efficient than the other commonly proposed topology of datacenter networks [8].

This research complements prior work by developing analytical models for energy consumption and thus enables us to study *fat-tree* DCNs theoretically. In order to dynamically adapt the datacenter topology to the network traffic load, we build mathematical models of energy usage for a *fat-tree* datacenter network. The approach can be generalized to derive energy consumption models for other network topologies as well. The model considers different traffic patterns and loadings, and can work as a reference in power-aware network design and utilized to estimate the energy consumption.

Based on the energy consumption models, we explore maximizing the energy saving through jointly optimizing task scheduling and flow assignment for given job loads. Network traffic can be planned and consolidated through virtual machine migration or changing the size of edge switches.

We next propose a hardware merge network to consolidate the traffic at the switches automatically. We evaluate the simulation models of datacenters with merge networks and obtain almost load-proportional energy consumption in large-scale datacenters. We also build a prototype of a 2 × 2 merge network using passive optical devices and test its performance. It shows that a merge network can successfully consolidate traffic and decrease the active ports needed for datacenter switches. Especially, for edge switches in the access layer, a significant amount of energy savings can be obtained.

The major outcomes of this work are summarized as follows:

1. Systematic analysis of energy cost of *fat-tree* datacenter network topologies to model how energy usage scales with total load as well as with different types of loads;

2. Usage-based topology analysis of *fat-tree* networks with job placement and scheduling in order to minimize network energy cost;

3. Development of a traffic driven model for *fat-tree* networks that proves to be

accurate in predicting switch activity;

4. Evaluation of energy-savings when using high-port-density edge switches with certain traffic patterns and the impact on inter-pod routing;

5. Heuristic flow assignment algorithms to compute routing tables empirically in simulations of large-scale datacenters;

6. Application of traffic merging in existing topologies and a study of the energy efficiencies obtained, throughput sustained and costs; and

7. Development, performance measurement and analysis of a new prototype of merge networks using fiber interfaces and optical switches.

Chapter 2

LITERATURE REVIEW

Traditional network design has focused on providing increasingly greater bandwidth and better coverage. As a result, networking equipment such as routers and switches were designed with little consideration given to energy efficiency. Indeed, networking equipment runs at full power regardless of traffic. However, empirical measurement shows that typical Ethernet traffic remains low for most of the time with occasional bursts, giving ample opportunity for energy saving of network equipment. At the network level, the network architecture is designed and dimensioned with over-provisioning and redundancy to sustain peak hour traffic. As a result, over-provisioned networks still consume a significant amount of energy during low traffic periods. A decade ago, researchers began paying attention to this energy waste and began studying ways to reduce it. In this chapter, we first discuss energy-aware networking and then describe this problem in the context of datacenter networks.

2.1 ENERGY-EFFICIENT NETWORKING

The IEEE802.3az [7] standard published in November 2010 represents the beginning of a change in networking architecture design. An EEE compliment network device should consume power only when data is being sent. When there is a gap in the data stream, the network device or interface is in an idle state and can be put into LPI mode. Early EEE-compliant devices use a simple application of static logic design in the physical layer devices (PHYs) to transition to the Low Power Idle (LPI) mode and save energy when data is not present. Later generations of networking systems apply new architecture design that uses more aggressive energy saving techniques to be applied to all of the system silicon, extending the range of energy savings.

2.1.1 Green Network Devices

Early work first explored the energy efficiency of network devices by considering two power modes: a sleeping mode and a fully working mode. The constraints here include the non-zero wake-up time for the interfaces and the spike in power consumption when the interface powers on. Therefore, it is challenging to find the optimal trade-off between system reactivity and energy savings, and determine when to wake up the device. In early research, Gupta and Singh [48] examined the feasibility of putting different subcomponents of a network switch or router into sleep mode, and described possible impact of sleep mode on protocols, such as VLAN, STP, and channel bonding. In their subsequent research [46], the authors described different types of sleeping mode for an interface and presented an algorithm incorporated in the host operating system that can dynamically transition the power mode of the interfaces. By analyzing the packet interarrival distribution in the buffer, the host determines if the next idle period is long enough to justify the energy saving of putting the interface into sleep mode. In this work, they also proposed to modify L2 protocols to enable a more aggressive strategy to keep the Ethernet interface in the sleeping state until the buffer queue exceeds a predefined threshold. However, some low-level network protocols such as ARP and STP require periodic control frames to maintain network connectivity, which is the primary constraint on the length of the sleep state. Christensen *et al.* [28]

[54] proposed network connectivity proxy (NCP) to handle network presence tasks for an idle network host in a low-power sleep state. To address the problem of possible packet loss during port sleeping period, G. Ananthanarayanan *et al.* [14] proposed a novel architecture for buffering packets at the network ports when the port is in the low-power state. Especially, they propose using a shadow port to receive ingress packets if any of the conventional ports are in the low-power state. Each shadow port has the same hardware with regular ports and associates with a cluster of normal ports.

Besides using sleep mode to reduce energy consumption during the idle period, newer network devices can scale power dynamically when they are in active state. During the low utilization period, network devices can lower the working rate of processing engines, and reduce link transmitting rate, resulting in significant reduction of energy consumption. In general, operating a device at a lower frequency allows substantial energy savings. For some network equipment (e.g. linecards, transceivers) that supports frequency scaling and dynamic voltage scaling, the reduction in power consumption is scaled cubically with operational frequency [70].

There is a great deal of research about dynamically adapting the link rate to the real load transmitted. C. Gunaratne *et al.* [40] show the measurement of the power consumption of Ethernet NICs and switches for a range of data rates and utilization levels. They first propose the notion of adaptive link rate (ALR) for Ethernet. In their work, the Ethernet link data rate is scaled as a function of queue length in both the PC and LAN switch. In a later work, the same authors propose a refined rate control policy based on dual buffer threshold [42]. Successively, they developed and evaluated a utilization-threshold policy and a time-out-threshold policy to eliminate rate oscillations in the rate transition process [41].

In more complicated switches and routers, the linecard accounts for most of the energy consumption. M. Mandviwalla *et al.* [65] study linecard architecture in backbone routers and dynamically scale the linecard's power to the predicted workload. The simulation results show that it can achieve 60% energy savings through dynamic power scaling. Hu *et al.* [52] propose reconfigurable router architecture that supports multiple router operational states (e.g., energy saving state) with fast switching ability. The router settings, including routing path, clock frequency, and supply voltage, are reconfigurable, aiming to support rate adaptive processing and power-aware routing. [77] measures the power consumption of NetFPGA-based gigabit routers in standard and low-frequency modes with different numbers of activated ports. They compare the internal power usage of a gigabit router at the granularity of packet and byte level, and analyzed the impact of router frequencies, numbers of activated ports, traffic loads and packet sizes on potential network power savings. In another work, the same authors propose a practical implementation of power scaling algorithms to modulate router frequency on a periodic or threshold basis adaptively [78].

Some other work compares the sleeping mode scheme and rate adaptation scheme. S. Nedevschi *et al.* [70] evaluate the two approaches regarding achieved energy saving, QoS, packet delay and loss rate. Both of the schemes can offer a substantial reduction in power consumption with minimum packet loss and a relatively small increase in network latency. However, there does exist a boundary utilization below which sleeping mode offers better energy savings than adaptive line rate, depending on how much a device's power consumption scales with frequency and the magnitude of its active-to-idle power draw. Furthermore, the authors compare two sets of data rates. In the first set, rates distribute exponentially (e.g., 10Mbps, 100Mbps, 1000Mbps), while in the second set they are distributed uniformly (e.g., 330Mbps, 660Mbps, 1000Mbps). Interestingly, the uniformly distributed rates have a lower additional delay and achieve a greater energy saving compared to the exponentially distributed set since the first set of rates require fewer rate transitions, which causes transition delays and leads to reduced energy saving and higher overall delay. However, the authors also mention that more supported rates increase the management complexity and cause extra overhead. Other work [67] [83] also conducts a comparison of sleeping mode and ALR mode, and concludes that rate adaption is more robust during bursty load periods while the sleeping mode has a much lower complexity and overhead with comparable performance.

Meanwhile, new hardware technologies enable re-design and re-engineering the network devices to improve hardware energy efficiency. For example, novel energy-efficient silicon (ASICs and FPGAs) contributes to performance gains of packet processing engines, allowing higher clock frequencies and fast packet forwarding, and achieves better energy cost per gigabit. Among this research, Yamada *et al.* integrate ASICs/FPGAs and router memories, and adapt a scalable central architecture in the router, which successfully supports 1Tbps with doubled energy efficiency. Also, some other research focuses on using optical switching architecture to replace electronic based devices [18]. In general, optical switching is much more energy efficient than its electronic counterpart. However, it is not possible to buffer the optical signal, so the optical switch lacks management flexibility. Also, optical switches only support a limited number of ports (less than 100), which limits their application to the large backbone networks.

2.1.2 Power-Aware Network Infrastructure

The emerging low-energy mode of network devices allows network switches, links or parts of the network be put into sleep mode, achieving non-negligible energy savings for each device or collaborative devices. Some work considers further coordinating the network-wide devices to dynamically put a portion of the network into sleep during the low to median utilization, to address the problem of network over-provisioning and redundant design. In this section, we discuss some of the network-wide energy-aware strategies.

Power-Aware Routing and Traffic Engineering

Energy-efficient devices utilize a modified physical layer and link layer protocols to coordinate the transition between sleep and active mode. At the network layer, power-aware routing considers consolidating traffic flows over a subset of links and network devices, allowing more idle interfaces, links, and network components to be put into the sleep state. Achieving a minimum subset of network devices is an optimization problem with the constraints of preserving full network connectivity and satisfying QoS requirements. Theoretically, the power-aware routing problem is an extension of the general capacitated multi-commodity flow problem, which is an NP-complete mixed integer programming problem (MIP).

In the position paper where Gupta and Singh [47] first presented the idea of putting network components into sleeping mode, the possibility of coordinated sleeping mode of multiple routers was described. The challenge is that the routing algorithms (OSPF, for example) will consider the sleeping nodes as having failed and recompute the routes, which incurs extra computing overhead. The paper discussed a possible solution of pre-computing alternative routes.

Chiaraviglio *et al.* [27] is the first paper that formulates an optimization model to find the set of routers and links that must be powered on so that the total power consumption is minimized, subject to flow conservation and maximum link utilization constraints. This network design problem falls into the class of capacitated multi-commodity flow problems, which is NP-complete. Therefore, they propose heuristic greedy approximation algorithms and apply that to the backbone network of ISP networks. They test different node and link selection strategies in responding to the day/night traffic patterns and prove that the total network power consumption can be reduced by switching off links and ports accordingly. Similar work includes [37], which evaluates heuristics algorithms on topology and traffic data from the Abilene backbone network. They prove that the simplest heuristic algorithm can reduce energy consumption by 79% under realistic traffic loads.

More recent work further explores the practical application of energy-aware routing. Coudert [33] formulates a model combining redundancy elimination and energy-aware routing to increase energy efficiency for backbone networks. [62] quantifies the effects of five recently proposed power-aware routing approaches and shows that switching off redundant links affects terminal reliability (TR) and route reliability (RR) significantly. Accordingly, they propose a practical algorithm, called "reliable Green-Routing" to maximally switch-off network cables subject to link utilization as well as TR/RR requirements.

Power-Aware Architecture Design

Power-aware routing aims at dynamically adapting network topology to network usage to address the over-provisioning problem. This approach can be a practical way to improve the energy efficiency of existing networks. Some other work advocates redesigning the network architecture in order to meet energy efficiency goals and QoS guarantees. Several authors propose new architecture design for energy savings [17] [25] [73]. For instance, [17] considers synchronizing the operation of routers and scheduling traffic in advance since traffic comes from predictable services (such as video). [25] proposes power awareness in the design, configuration and management of networks, and in protocol implementations. They conducted a measurement study about the power consumption of various configurations of widely used core and edge routers and created a generic power model for routers. Nevertheless, [73] proposes a planning model that clearly shows the trade-off between energy consumption and network performance and emphasizes the importance of accounting for reliability in energy-efficient network design and analyzed robustness issues in some of the designs. To leverage the high efficiency of optical switching, some research on hybrid network architecture combines optical transport and electronic packet processing. For instance, Baldi *et al.* [17] propose to use a complemental Dense Wavelength Division Multiplexing (DWDM) optical cable for deterministic traffic.

2.1.3 Power-Aware Software Stack

In the operating system and user-space applications, it is possible to implement energy strategies at the transport layer and application layer protocols. For example, Irish *et al.* [59] modify the TCP/IP protocol, putting a `TCP_SLEEP` signal in the TCP header to notify the other party to stop sending data. Application layer protocols can enforce the power-aware configuration as well. For example, Blackburn *et al.* [53] [22] customize Telnet and BitTorrent protocols so that clients send sleep-signal to servers to advertise the energy state and implement a probing mechanism to avoid sending keepalive message. Furthermore, Microsoft scientists developed general tools for application programmers in energy-efficient programming. Kansal *et al.* [55] present automated tools that profile the energy usage of various network resource components used for the guidance of energy-efficient application design. Baek *et al.* [16] provide a framework that enables programmers to approximate expensive functions and loops in a systematic manner while providing statistical QoS guarantees.

2.2 DATACENTER NETWORK ARCHITECTURE

2.2.1 Datacenter Network Topologies

Inside a present-day datacenter, tens of thousands of servers are interconnected using a network of switches, called datacenter networks (DCNs). Many datacenters deploy a multi-layer, multi-rooted tree structure DCN, such as *Clos* [32] and *fat-tree* [11] (also known as *folded-Clos*).

A *Clos* network topology [29][32] has three stages. Each stage is composed of many crossbar switches. An (m, n, r) *Clos* network has m middle-stage switches, r input switches, and r output switches. n is the number of input(output) ports in the input(output) switches. Each input switch is an $n * m$ crossbar and every output switch is an $m*n$ crossbar. The input switches are fully connected with the middle-stage switches, and the middle-stage switches are then fully connected with all output switches. Every switch in the middle-stage has r input links from input stage switches and r output links to output stage switches. Thus, the middle-stage switches are $r*r$ crossbars. An (m, n, r) *Clos* network can have $N = rn$ end nodes. With m middle-stage switches, there are m different paths between each pair of the input and output nodes. Therefore, *Clos* topology has very good path diversity.

Fat-tree [11] is another example of a multi-rooted tree topology that is widely deployed in many datacenters. A *fat-tree* network leverages off-the-shelf Ethernet switches to connect to tens of thousands of nodes. A k-ary *fat-tree* has k *pods*, each containing two layers of $\frac{k}{2}$ switches. Each edge switch in a *pod* is connected to $\frac{k}{2}$

hosts and $\frac{k}{2}$ aggregation switches. There are $\left(\frac{k}{2}\right)^2$ core switches, each of them has k ports connected to k *pods*. The *fat-tree* topology has great scalability. A k-ary *fat-tree* network can support $\frac{k^3}{4}$ hosts. Furthermore, the *fat-tree* topology has identical bandwidth at any bisection and each layer has the same aggregated bandwidth. Therefore, it can achieve full bisection bandwidth with 1:1 oversubscription ratio. Compared with other conventional tree-based topologies, a *fat-tree* network has less bandwidth bottleneck issues and can provide high bandwidth by interconnecting smaller commodity switches.

Another cost-efficient interconnection network topology is called *flattened butterfly* [57]. The *flattened butterfly* structure is derived from the conventional *butterfly* topology by combining the routers in each row into a single router. Channels inside a row are eliminated. All the other channels in the *flattened butterfly* are bidirectional. By combining the routers in the same row, the connection path between pairs of nodes can take any order of dimensions to get through, providing better path diversity than a conventional *butterfly* topology. A k-ary n-flat *flattened butterfly* can support $N = n^k$ end nodes with $n\,(k-1)+1$ links. With high degree of interconnection, *flattened butterfly* scales more effectively than k-ary n-cubes. Also, *flattened butterfly* has smaller network diameter than the *folded-Clos* network. With load-balanced traffic, a *flattened butterfly* is approximately half the cost of a *folded-Clos*.

The *fat-tree*, *Clos*, and *flattened butterfly* are all switch-centric architectures with interconnection intelligence built in switches [45]. Recently, some new server-centric architectures were proposed to rely on servers to make routing decisions and use them as intermediate routers. Examples of server-centric topologies include *DCell* [43], *BCube* [44], *FiConn* [60] and CamCube [9].

DCell [43] is a recursively-defined interconnecting structure. Lower-level *DCell*s are connected to construct a higher-level *DCell*. Therefore, *DCell* scales out doubly exponentially. It can support an enormous number of servers with only a few levels (k) and switch ports (n). For example, a 3-level *DCell* can accommodate around 3.26 million servers [21]. In a *DCell* network, servers are active parts in routing and forwarding process. Each *DCell* can be deemed as a virtual node, and all virtual nodes in the same level are fully connected to each other. The rich physical connectivity and distributed routing protocol provide better fault tolerance. There is no single point of failure in a *DCell*. Compared with other datacenter network topologies, *DCell* has better scalability, fault tolerance, and higher network capacity. However, *DCell* requires higher wiring cost and it has problems of load balancing.

BCube [44] is another server-centric architecture suitable for container-based modular datacenters. The switches only connect with servers and act as a crossbar. Servers communicate through the switches or other relaying servers. Since all the switches in the *BCube* are equally connected, therefore, there is no oversubscription bottleneck and *BCube* can provide high inter-server throughput. However, *BCube* requires the server to have multiple network ports to scale out, which is a barrier for *BCube* to scale to millions of servers.

DCell and *BCube* both require that a server has more network interfaces to scale out, which is a challenge for datacenters currently using commodity servers with at most two network ports each. *FiConn* [60] is a server-centric network, which expands similarly to *DCell*. However, the server in *FiConn* needs only two Ethernet ports to scale out to a large number of servers. *DPillar* [61] is another server-centric network with dual-port servers. *Dpillar*'s structure was inspired by the classic *butterfly* network. It has symmetric structure and eliminates network bottleneck. As a result, *DPillar* can achieve better scalability and network performance.

The server-centric networks we discussed above are all hybrid direct-connect topologies, which use simple dummy mini-switches to connect servers. Abu-Libdeh *et al.* [9] propose a direct-connect topology (*CamCube*) where each server connects directly with a set of other servers, without using any switches or routers. It connects servers in a 3D torus topology, and each server directly connects to six other servers.

Singla *et al.* propose a high-capacity network topology called *Jellyfish* [76]. It adopts a random topology with high flexibility and network capacity. A *Jellyfish* network has small network diameter and supports fine-grain incremental expanding. However, the unstructured design also brings challenges in routing strategy and networking wiring. Similar works include *Scafida* [49] and *Small-World Datacenters (SWDC)* [74].

2.2.2 Datacenter Network Protocols

There is a broad range of applications running in the datacenters today, from web hosting services to file storage services, to other customized applications. With the increasing popularity of cloud computing, large online service providers such as Amazon, Google and Microsoft have launched large-sized cloud datacenters, offering user-facing Internet services such as web services, Instant Messaging, and webmail. Additionally, many large-scale data-intensive applications like MapReduce, Hadoop, and Dryad run in large production datacenters. Benson *et al.* [20] studied the applications and their traffic patterns of ten datacenters.

As the number and variety of applications increases, the network performance of datacenters will significantly influence the QoS. For example, a MapReduce job running in the partition/aggregate pattern distributes small tasks to other worker nodes and collects results afterward, which requires transferring large amounts of data among servers at very high rate, demanding substantial network bandwidth. Secondly, MapReduce is also latency-sensitive which require minimum response time. If a worker node misses the deadline, its result will just be ignored, which will impact the quality of the overall outcome.

To address the performance requirements of applications, datacenter designers have focused their work on routing, flow control and management protocols to optimize the desired metrics of network utilization, latency, and throughput. Current datacenter network protocols used in datacenters originate from those designed for general LAN settings, which has predictable communication patterns and limited paths between end nodes. With the exponential increase in the number of hosts, especially the appearance of distributive cloud computing, many research efforts are being focused on datacenter protocols to address the network management challenges and new application requirements.

Routing and Addressing

The development of large-scale datacenters with an increasing number of servers also imposes a significant challenge on the scalability of the datacenter protocols. In a cloud computing environment, host virtualization allows multiple virtual machines (VMs) running on one physical machine. Each of the virtual machines is assigned a fixed IP address and a MAC address, requiring more scalable address resolution to locate millions of end nodes. Furthermore, virtual machines are migrated to tightly-coupled hosts in order to achieve higher throughput, which imposes challenges on IP address configuration. Some researchers have proposed new addressing schemes for datacenters. For example, Al-Fares [11] described a particular IP addressing over the *fat-tree* topology to provide high bisection bandwidth without using high-cost core switches. The *fat-tree* network enforces a special

IP addressing scheme and a routing protocol customized for its topology. The routing algorithm uses two level look-ups at each node to distribute traffic. The prefix look-up in the primary table routes down to servers and the suffix look-up in secondary table routes up towards core switches. The pod switches forward subsequent packets of the same flow to the same outgoing port. Hence, all the packets going to the same destination will follow the same path without packet reordering.

Datacenter protocols originate from Ethernet and IP-based protocols supporting arbitrary topologies. As a result, many current datacenter protocols have obvious limitations such as inflexibility, high configuration overhead, and limited scalability. Recently, some Ethernet-compatible protocols for datacenters have been proposed to address these problems.

For example, *SEATTLE* [56] is an early proposal of scalable Ethernet-compatible architectures for large-scale datacenters. *SEATTLE* keeps the simplicity of Ethernet by forwarding packets based on layer 2 flat MAC addresses, and employs a broadcast-based link state protocol. Flat addressing treats the datacenter as a unified entity and enables lower administration overhead on handling network configuration and host mobility. To overcome the scalability problem of flat addressing, *SEATTLE* employs a directory service by building a one-hop Distributed Hash Table (DHT). Instead of requiring each switch to maintain a state for every end node, *SEATTLE* employs a link-state protocol, which only keeps switch-level topology and then uses a hash function to map host information to a switch. Switches first run a discovery protocol to find their positions and automatically configure the link-state protocol. Additionally, *SEATTLE* leverages communication locality by letting switches cache the shortest paths of previous queries. Further, location information is memorized during end host ARP queries for later data packet transmission. Compared with other hybrid IP/Ethernet network protocols, *SEATTLE* improves the communication efficiency with minimal management complexity.

Another proposed approach with similar goals is *PortLand* [69], which is a scalable and fault-tolerant layer 2 network protocol implemented over the *fat-tree* topology. It consists of a set of routing, forwarding, and addresses resolution protocols, which applies to any multi-rooted tree topology of current datacenters. Instead of using flat MAC address as seen in *SEATTLE*, *PortLand* leverages the knowledge of the topology by encoding positions into hierarchical addresses and thus achieves scalability and efficiency. It uses a pseudo MAC (PMAC) addresses and embeds the baseline network topology information into the PMAC addresses. Each end node is assigned a unique PMAC address containing hierarchical position information. Switches update the forwarding table through a lightweight Location Discover Protocol (LDP). Compared with the traditional layer 2 flat MAC addresses, the hierarchical structure of the PMAC addresses allows a relatively small forwarding table in each switch and enables more efficient routing and forwarding.

While *PortLand* achieves scalability by using topologically-significant PMAC addresses, the position-related addressing constraints virtual machine migration. To overcome these limitations, Greenberg *et al.* presented the Virtual Layer 2 (*VL*2) [39] network architecture based on layer 3 IP routing in the network infrastructure, and implemented flat addressing at the server level. In *VL*2, all the switches are using location-specific IP addresses (LAs). At the application level, applications are assigned application-specific IP addresses (AAs). The *VL*2 agent at each server encapsulates the packet with the AA address of the application, and LA address of the ToR switch directly connected to the destination server. When the packet arrives at the destination ToR, the switch decapsulates the packet and forwards it to the target server. The AA address represents the name of the server instead of the location, which allows virtual machine migrating with no IP address modification overhead. And the network-level IP protocol assures high-efficiency forwarding and scalability with a small state at each switch. *VL*2 implementation does not require changing the software and API of current switches. Therefore, it is a practical solution for datacenters with commodity switches.

The multi-rooted tree structures of current datacenter networks provide multiple paths between each pair of servers. Many existing forwarding protocols apply Equal Cost Multipath (ECMP) to select a path statically using flow hashing, resulting in collisions and bandwidth losses. To address these challenges, Al-Fares *et al.* proposes *Hedera*, a dynamic flow scheduling system to adaptively allocate flows to paths [12]. *Hedera* is a centralized scheduler with a global view of the flows and network utilization status. Therefore, it can not only appropriately schedule the flows to the core switches with less utilization, but also fully leverages the high degree of parallelism provided by the multi-rooted tree topology and achieves nearly full bisection bandwidth.

To achieve load balancing in multipath datacenter networks, a novel multipath forwarding approach is proposed as Smart Path Assignment in Network (*SPAIN* [68]). While traditional Ethernet protocol uses Spanning Tree Protocol (STP) to generate a single loop-free-tree, *SPAIN* explores a set of redundant paths in a network topology and merges these paths into a set of trees. Every switch installs the information of VLANs, each of which maps to one tree. *SPAIN* supports layer 2 flat addressing and routing, and can deliver higher bandwidth and better fault tolerance than spanning tree.

Similarly, Raiciu *et al.* propose a Multipath TCP (*MPTCP*) [71] as an extension of current TCP protocol. *MPTCP* explores multiple paths simultaneously and utilizes the congestion feedback to choose the effective paths. Compared with single-path TCP, *MPTCP* can find unused capacity more effectively and maximize the utilization of networks in topologies with full bisection bandwidth. Now, *MPTCP* has been deployed in Amazon's latest $EC2$ environment, and experimental tests show that it can improve the throughput by 300%.

The server-centric architectures allow servers to perform routing and forwarding. *DCell* uses a single path routing protocol. Each server is assigned a $(k+1)$-tuple as the address. The *DCellRouting* routing algorithm follows a divide-and-conquer approach to find the path from source to destination. *BCube* adopts a source routing protocol called BSR (*BCube* Source Routing). When a new flow arrives, the source server sends out probe packets through multiple parallel paths and selects the best route after it receives the probe response. BSR can fully utilize the high capacity provided by the *Bcube* topology and realizes automatic load-balancing. In a *FiConn* network, every intermediate server takes a greedy approach for establishing a traffic-aware path hop-by-hop. The source server always selects the outgoing link with higher bandwidth to forward the traffic, thus balancing the traffic.

Flow Control and Resource Management

OpenFlow [66] provides an open protocol to program the flow table in a switch. OpenFlow-enabled switches support fine-grained, flow-level control over Ethernet switching. However, its centralized control and global visibility create scalability issue as well. Curtis *et al.* designed a modified model called *DevoFlow* [31]. It pushes most flow controls to switches and only manages over significant flows and packets. The distributed control mechanism reduces the sizes of flow tables and control messages.

Many studies show the datacenter traffic is characterized by a few large flows and many small flows. Although there are a few long-lived flows, these flows play a critical role in deciding the achievable network bisection bandwidth. Current datacenter commodity switches have limited buffer sizes. Short-length flows may also experience long latencies if the long-length flows occupy the available buffers in the switches. To keep the high throughput for big flows and low latency for short flows, a TCP-like Datacenter TCP protocol (*DCTCP*) [13] is proposed to address this problem. *DCTCP* uses Explicit Congestion Notification (ECN) as feedback of the extent of switch congestion and proportionally resizes the window. A traditional TCP reduces the window by half when it receives ECN feedback, which causes buffer underflow and throughput losses. Experimental results show that *DCTCP* can deliver comparable throughput with 90% less buffer space occupancy compared with conventional TCP.

Another novel traffic management system, *Mahout* [30], was proposed recently to detect elephant flows in datacenter traffic. Instead of polling the switches, Mahout monitors the end hosts socket buffers to detect long-lived flows and signals the central controller. Hence, only the detected long flows will be forwarded through the central controller. As such, Mahout incurs much lower monitoring overhead and switch resources compared to *Hedera*. It is relatively simple to implement Mahout because it only requires a shim layer of software at the end host OS.

Although many resource management proposals consider the sizes of flows in congestion control and flow scheduling mechanisms, few of them are aware of the deadline of the flows and study how meeting the deadline will influence the application throughputs. Wilson *et al.* designed and implemented a Deadline-Driven Delivery (D^3) control protocol [84] to apply explicit rate control to allocate network bandwidth according to the flow deadline. Results from an implemented small testbed show that D^3 can effectively double the peak load supported.

2.2.3 Alternative Datacenter Architectures

The topologies we have considered thus far consist of an enormous amount of electrical switches and links, resulting in high complexity of wiring and deployment of the datacenter networks. Compared with the packet-switching technology, the optical circuit switching technology can provide higher bandwidth at much lower power cost, but it comes at a cost of a slower switching speed. An optical switch requires on the order of one millisecond to establish a new circuit, which is much longer than the transmission time for a single packet. Therefore, optical circuit switching works best with high-speed, high-volume communications. Many proposals present new network architectures to leverage the high bandwidth transmission advantage of optical circuit switching.

Wang *et al.* propose a hybrid packet and circuit switched datacenter network architecture ($HyPaC$) [80] that combines traditional electrical packet-switched network with rack-to-rack circuit-switched optical network. In the implementation of the prototype system, servers buffer traffic and accumulate enough traffic for the links to leverage the high-speed bandwidth. The experiment on an implemented prototype shows that the optical inter-rack switches integrate well with the Ethernet/TCP electrical switches. The case studies on different types of applications suggest that this hybrid architecture benefits applications with bulk data transfer requirement and which are also insensitive to latency.

Modular datacenters have been a new direction in building datacenters in the past few years. Moderate numbers of servers interconnected with non-blocking networks form a module called a pod. High bandwidth inter-pod connections are necessary to prevent the bottleneck of communications. An optical switch is an option for the pod-level aggregated traffic demand. Farrington *et al.* [36] proposed *Helios*, a hybrid electrical/optical switch architecture for modular datacenters. Helios dynamically reconfigures the network topology at run-time according to the communication patterns and traffic demands monitored.

More recently, Chen *et al.* [26] introduced an innovative Optical Switching Architecture (OSA) for datacenter networks. Consisting of all optical switches, OSA achieves high topology flexibility. It can dynamically change topology or link capacity more flexibly to adapt to varying traffic patterns. Compared with the hybrid structures, OSA is characterized by a higher bandwidth, lower energy consumption, and simpler wiring complexity. However, small flows may suffer non-trivial delay due to the reconfiguration delay of OSA.

2.3 DATACENTER NETWORK ENERGY EFFICIENCY

The exponential growth of Internet-scale applications drives the expansion of large-scale, geographically distributed datacenters with fast increasing energy cost. With various types of applications from online services, scientific computations to MapReduce running on datacenters, the communication patterns and network bandwidth requirements have been driving new research on datacenter networks. We explore the literature of recent studies that addresses various aspects of the challenges in designing efficient datacenter networks.

A lot of research efforts are focused on achieving the energy proportionality in datacenter networks. In practice, companies provision their datacenters for peak usage. However, a typical datacenter workload is around $5\% - 25\%$ of the peak. Many researchers propose energy proportionality datacenter networks. For example, Lin *et al.* [63] explored the option of adaptively right-sizing the datacenters by turning off idle servers. Heller *et al.* designed an ElasticTree topology [51] that changes network topology dynamically to adapt to varying traffic load.

Lin's research introduced an online algorithm, Lazy Capacity Provisioning LCP(w), to predict arriving workload in a window size of w. In the case study, some impacting parameters are discussed to analyze the cost-saving of their approach. Especially, the impact of Valley Filling, an alternative approach to right-sizing, is evaluated and compared.

In the *ElasticTree* approach, Heller developed a variety of optimizers to compute a minimal-power subset of network elements according to different traffic patterns. The power control turns off unnecessary switches and links and the routing assigns routes accordingly. Tradeoffs between power, fault tolerance and performance are considered as well in their approaches. Similar work on dynamic topology is called *CARPO* [81], a correlation-aware power optimization algorithm. Different from *ElasticTree*, *CARPO* first consolidates traffic flows by putting negatively-correlated flows onto the same path and positively correlated flows onto different paths, and then feeds the consolidated flows into a smaller set of links before shutting off idle links.

More recently, Adnan and Gupta proposed an online path consolidation algorithm to dynamically right-size the networks [10]. From multiple equal cost paths between each pair of nodes, their algorithm selects the path, which has most overlap with the paths between other pairs of nodes and meets the total flow bandwidth requirement. By combining all the best overlapping paths together, the minimum energy-proportional topology is formed. When there are large amount of flows in the network, their method outperforms the *ElasticTree* approach.

Also, others have explored the ideas of energy-proportional hardware, such as energy-proportional servers or energy-proportional links. For instance, Abts *et al.* proposed building energy proportional datacenter networks by adapting the data rate of individual links to the traffic intensity [8]. Abts then compared the power consumption rate of different topologies and proposed to build the network based on the *flattened butterfly* topology, which is more energy-efficient than a *fat-tree* topology of equivalent size and performance. Compared to the dynamic topology, this approach is a more fine-grained tuning adaptive technology. Additionally, it does not require changing the topology and routing.

An innovative idea of traffic merging was proposed recently for running more energy-efficient datacenters [23]. Given the low utilization of the links, traffic from multiple links is aggregated into fewer uplinks through a simple hardware design of the Merge Network. Unused links can be set to low power mode to save energy. The hardware implementation is simple and has no additional delay and has lower small power consumption.

2.4 COMPARISON AND DISCUSSIONS

The expansion of datacenter infrastructure motivates the research on how to efficiently interconnect a huge number of servers. The high capacity requirement of applications implies that the topologies have high bisection bandwidth. For example, the *fat-tree*, and *Clos* are all hierarchical multi-rooted tree-structured topologies connecting multiple levels of commodity switches. These topologies support high-rate communications between any pair of end hosts and are widely deployed in existing datacenters. Another approach considers server-centric topologies, such as *DCell*, *BCube* and *FiConn*. These topologies leverage the programming capability of servers and conduct routing and forwarding by the servers instead of switches. Network scales out through recursive expansion of lower-level server interconnections. Compared to the tree-based topology, server-centric topology has better scalability and more convenient routing and management mechanism.

Also, there are alternative architectures proposed using optical switches to meet the high bandwidth demands between different racks. Optical switches can form a circuit-switching path and have very high transmission rate.

In our research, we apply merge networks to the switches in different topologies and evaluate the total energy saving under various traffic patterns. For a hierarchical tree-structured *fat-tree* topology, more traffic concentrates in the core layer switches, according to Benson [20]. The link utilization of other layers is much lower. Considering this variation, we can apply merge networks more aggressively in the aggregation/edge layer. For example, we can connect a $2N \times 2N$ merge network to 2 N-interface switches.

We also propose using a switching fabric for the interconnection of servers. Conventional networks use switches as intermediate nodes to forward packets. The forwarding switches increase the network latency and consume large amount of energy. In our research, we will replace switches with simple analog multiplexers, which use minimal electricity. We use servers to make routing decisions and establish a full path from source to destination before data transmission. As we discussed, server-centric architectures also put routing intelligence on the server. However, they need servers to work as the relaying nodes during the transmission, which incurs high latency. Our approach lets the source server set up a particular path. The packets then traverse the path from source to destination without going through intermediate servers. Optical switches also establish a circuit switching path. However, they have a long configuration time, which hinders its application to inter-server data transmissions. Typically, optical switches are used for inter-rack networking. Our proposed switching fabric is simple to configure and has low transmission latency and energy cost.

The increasing concern about datacenter power consumption has attracted many research efforts. Barroso [19] has proposed the concept of energy proportionality, which resorts to keep the energy usage proportional to the offered workload. There are different approaches to achieve energy proportionality. Some proposed dynamically turning off idle servers [63]. Others focus on ensuring the energy use of network infrastructure to be proportional to the network utilization, such as turning off unused switches [51] or adapting the link rate to the workload [8]. More aggressive approaches include consolidating traffic flows [81] or merging transmission paths [10] to find minimal energy proportional topology. Our research on merge network is also focused on combining the traffic and dynamically turning off the idle switches. However, the previous traffic consolidating approach [81] requires statistical analysis of the traffic flow correlations. The path-consolidating approach [10] has to calculate the optimal overlapping path first. The merge network is implemented from simple analog hardware, and it consolidates traffic automatically with almost no software overhead. The switching fabric proposed is a more energy efficient way by replacing all switches with low-energy-cost analog multiplexers.

Chapter 3

MODELING ENERGY USAGE OF DATACENTER NETWORKS

Datacenter networks tend to consume about $10 - 20\%$ of energy in normal usage [51] but account for up to 50% energy [38] during low loads since at those times servers can be put into low power states. It is therefore important to adapt the network energy consumption to actual traffic loads as the servers do. To do this, we need to develop a better understanding of network energy consumption under varying types of loads with the eventual goal of designing more energy efficient datacenter networks.

We consider the *fat-tree* network, which has been a popular choice for many commercial data centers due to its full bisection bandwidth (which minimizes latency and boosts throughput). Unfortunately, the energy consumption of this or any other network is very dependent on the type of traffic, the type of load and on the selected routing algorithm. For instance, even at high loads if most of the traffic is between servers located in the same pod (see Figure 3.1) then core switches are never used resulting in significant energy savings. On the other hand, at light loads if most of the traffic is between servers located in different pods, then savings are small since more switches in the network will need to be utilized for routing. Routing also plays an important part in the potential for energy savings. Thus, routing algorithms that seek to minimize only latency will distribute flows over unused paths when possible, ensuring that a majority of switches are kept busy (albeit at very low loads). Alternatively, if paths can be consolidated into a few, then there is the potential to save energy at the idle switches.

In this chapter, we provide a systematic analysis of the energy efficiency of a *fat-tree* network using modeling and detailed simulations. The key question we ask is *how does energy usage scale with total load as well as with different types of loading.* To answer this question, we build a detailed analytical model that gives the *lower bound* on the *fraction of active switches* required for a given load and type of load. We show that *fat-tree*s have a minimal cost of about $40 - 50\%$ (i.e., about half the switches need to remain active at all times) but beyond that, the lower bound scales almost linearly with total offered load. The lower bound computation is based on aggregating traffic into few routes. We next conducted a detailed simulation of a *fat-tree* network where we used different types of load (staggered and stride) [11] and different amounts of total load. We compute routing tables empirically every second for the next second and compute the fraction of needed active switches. The simulation shows that the model we develop is accurate in predicting switch activity and the simulation demonstrates that by modifying the routing algorithms we can potentially save significant amounts of energy in real networks.

3.1 MODELING ENERGY CONSUMPTION

The structure of a fat-tree is shown in Figure 3.1. The tree is made up of $2k$ "pods" which are connected to k^2 core switches. Within each pod we have k aggregation switches and k edge switches. Each edge switch is in turn connected to k servers. Therefore, each pod has k^2 servers and the DCN has a total of $2k^3$ servers. Each core switch has one link to each of the $2k$ pods. The ith port of a core switch is connected to an aggregation switch in pod i. The left-most k core switches are connected to the leftmost aggregation switch of each of the $2k$ pods. The next set of k core switches are connected to the second aggregation switch of each of the

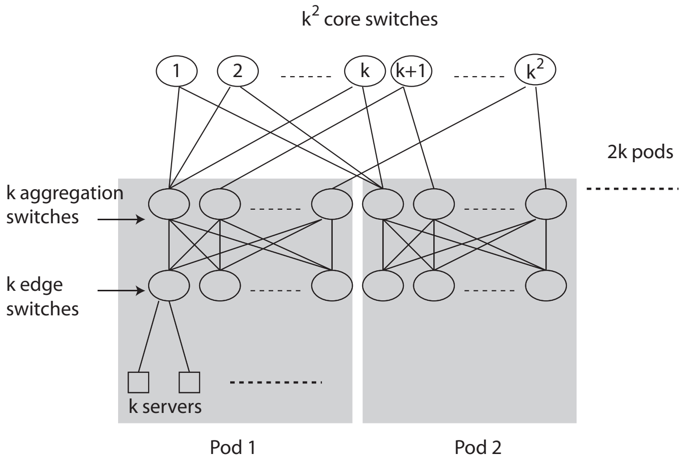


Figure 3.1. *Fat-tree* network model.

pods, and so on.

Our goal here is to derive analytical expressions for minimal energy consumption of *fat-trees* for different type of loading. The metric we use for energy consumption is *fraction of active switches*. To model different types of loading we use three parameters. A packet from a server goes to another server connected to the same edge switch with probability p_1, it goes to a server in the same pod but another edge switch with probability p_2 and with probability $p_3 = 1 - p_1 - p_2$ it goes to a server in a different pod. Thus p_1 of the traffic is never seen by either the core or the aggregation switches while p_2 of the traffic is not seen by the core switches. It is easy to see that by varying p_1 and p_2 we can model very different types of traffic. Finally, we model external traffic (i.e., traffic going to/from the Internet) as the fraction q (also see discussion in first part of section 3.1.2).

Let λ denote the average *internal* load offered by each server expressed as a fraction of link speed (which we normalize to 1). This load refers to packets that will stay within the datacenter. Thus, the *total offered load per server* is $\lambda + q$. For simplicity, we assume that $\lambda + q$ is the same for all the servers in the datacenter. Thus, the total load in the datacenter is $2k^3(\lambda + q)$. We have the following equalities for total traffic at the level of edge switches, pod aggregation switches and core switches:

$$\text{traffic per edge switch } = (\lambda + q)k$$

$$\text{traffic for all aggregation switches in a pod } =$$
$$((1 - p_1)\lambda + q)k^2$$

$$\text{traffic for all core switches } =$$
$$((1 - p_1 - p_2)\lambda + q)k^2 \times 2k$$

Note that traffic flow is symmetric and the numbers above correspond to both, traffic into and out of a switch or switches.

We perform our analysis below in three stages:

- In the *basic model* we assume that $q = 0$ and thus all traffic is internal only. This analysis gives us a good starting point for generalization to the other two models.

- The *extended model* allows $q > 0$ but assumes that every core switch has a connection to the Internet. This model is valid for small datacenters.

- In the *asymmetric model* we only allow a small subset of core switches to be connected to the Internet and these switches are equipped with much higher rate links. This model is representative of a large number of datacenters today.

3.1.1 Basic Model

Assume that $q = 0$. The first observation we can make is that *all the edge switches need to remain active at all loads* to ensure servers have network connectivity. This gives us $2k^2$ active switches at this level. Within each pod we have total traffic equal to $(1 - p_1)\lambda k^2$ going into/from the k aggregation switches from/to the edge switches. Given each link has a normalized capacity of 1, and that there are k interfaces per aggregation switch connected to the edge switches, we require at least $\lceil \frac{(1-p_1)\lambda k^2}{k} \rceil$ active aggregation switches per pod. Observe that in the *fat-tree* each edge switch is connected to each aggregation switch. Therefore, we can force all traffic from the edge switches to go to the fewest number of aggregation switches. This fact is represented in the expression for the total number of active aggregation switches above. Since there are $2k$ pods, the total number of active aggregation switches becomes $2k\lceil(1-p_1)\lambda k\rceil$. Finally, since every core switch i is connected to aggregation switch i from each of the $2k$ pods, the number of active core switches we require is simply, $\lceil \frac{(1-p_1-p_2)\lambda \times 2k^3}{2k} \rceil$ where we divide the total traffic passing through the core switches by the number of links per switch and round up. Therefore, the total number of active switches can be written as:

$$\begin{aligned} \text{Active}_{\text{Basic}} = \ & 2k^2 + 2k\lceil(1-p_1)\lambda k\rceil + \\ & \left\lceil \frac{(1-p_1-p_2)\lambda \times 2k^3}{2k} \right\rceil \end{aligned} \tag{3.1}$$

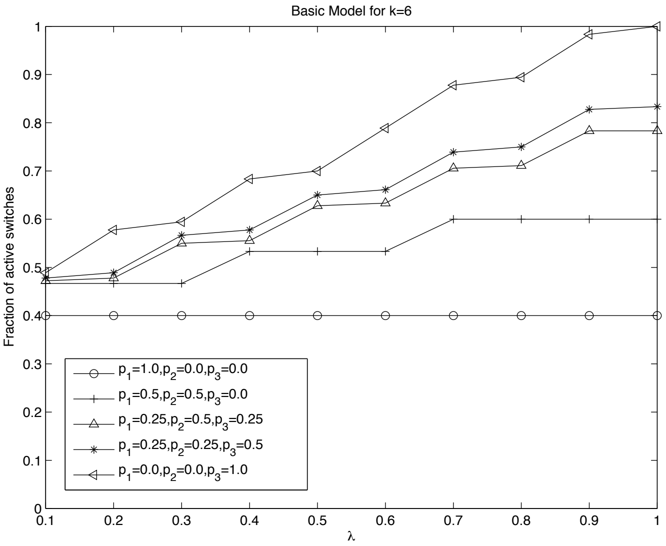


Figure 3.2. Active switches for the basic model.

Figure 3.2 plots the fraction of active switches as a function of load λ for five different scenarios when $k = 6$. The plot with the labels *-o* corresponds to the case when all traffic is between the servers connected to an edge switch. In other words, no traffic needs to flow to the aggregation switches or to the core switches. As expected, the graph stays flat. However, *what is relevant here is that even at light loads of 0.1 all the edge switches are fully active.* At this load value, each server generates 1/10th of the uplink capacity of traffic (similarly for downlink) but the energy consumed is the same as when the link is fully loaded. The total combined traffic from all the $k = 6$ servers is 0.6 which is less than the capacity of a single link. It is clear that significant energy savings can be accomplished here by redesigning the edge switches or the topology at the edge. We return to a discussion of this point later.

The plot with the labels $-\triangleleft$ corresponds to the extreme case when all the traffic is destined for servers in a different pod. Hence the core and aggregation switches will be utilized. Contrasting this with the case discussed above, we observe that *energy scales approximately linearly with load*, if we discount the edge switches. This is the desired behavior for energy-proportional networking.

3.1.2 Extended Model

Let us now extend the above discussion to the more realistic case when the datacenter sees external traffic from the Internet. Let us assume that traffic coming into the datacenter is $2k^3q_{\text{in}}$ equally distributed among all the servers and traffic going out is $2k^3q_{\text{out}}$ also equally generated by each server. It is easy to see that $\lambda + q_{\text{in}} \leq 1$ and $\lambda + q_{\text{out}} \leq 1$ since the normalized capacity of the link connecting each server to the edge switch is 1. Before proceeding with the derivations below, note that a switch interface is typically bi-directional. Therefore, even if there is no traffic in one direction, the entire interface is functioning and running link layer protocols to maintain connectivity. Therefore, instead of considering q_{in} and q_{out} separately, we need to only consider the maximum of the two. Let,

$$q = \max\{q_{\text{in}}, q_{\text{out}}\}$$

In order to handle external traffic, *let us assume that each of the core switches is equipped with additional interfaces with a total normalized capacity of* Q and connected to a border switch or router. Therefore the network can handle a total external load of Qk^2. Observe that $Q \leq 2k$.

In order to compute the number of aggregation switches and core switches that are active, it is convenient to begin at the core layer. The total external traffic is $2k^3q$ therefore the minimum number of core switches required to handle this traffic is,

$$n = \frac{2qk^3}{Q}$$

It is possible that n is a fraction or is greater than the total number of core switches. Therefore, we obtain,

$$n_{\text{core}}^{\text{ext}} = \min\left\{k^2, \lceil n \rceil\right\}$$

since each core switch can only handle Q external traffic. The reason we take a minimum above is to account for the case when the external traffic exceeds the total capacity of the network to handle it.

Each of the $2k$ interfaces of the core switches (facing towards the servers) has a normalized capacity of 1. For the core switches serving external traffic, $Q/2k$ *of each link's capacity* is used for external traffic coming/going from/to the connected pods leaving a capacity of $(1 - Q/2k)$ for internal traffic between pods. The reason for this is two-fold. First, each of the pods is assumed to be identical to the other pods and generate an equal amount of external traffic. And second, in the computation of the number of core switches needed to support the external traffic, we assume that *all the external traffic is put into as few core switches as possible rather than spreading it out among all the core switches.* This design is more energy efficient since we can minimize the number of active switches.

We may require additional core switches to handle inter-pod internal traffic. To compute this additional number of core switches we first determine how much internal traffic can be carried by the $n_{\text{core}}^{\text{ext}}$ switches. The remaining internal traffic can then be carried by free core switches. To compute the first value, note that of the $n_{\text{core}}^{\text{ext}}$ switches, the first $n_{\text{core}}^{\text{ext}} - 1$ will be running their external links at full capacity of Q each while the last switch may be running at lesser capacity. Thus, the residual capacity of these active switches can be written as,

$$(2k - Q)\lfloor n \rfloor + (2k - (2qk^3 - \lfloor n \rfloor Q))$$

Recall from our discussion of the basic model that the total internal traffic reaching the core switches from the pods (i.e., traffic sent between pods) is $2(1-p_1-p_2)\lambda k^3$. Therefore, the number of additional core switches we require to handle internal traffic is,

$$n_{\text{core}}^{\text{addl}} = \left\lceil \frac{2(1-p_1-p_2)\lambda k^3 - \big((2k-Q)\lfloor n \rfloor + (2k - (2qk^3 - \lfloor n \rfloor Q))\big)}{2k} \right\rceil \tag{3.2}$$

We divide by $2k$ because that is the capacity of each free core switch ($2k$ links of capacity 1 each). Of course, it is possible that the total traffic will exceed the capacity of the core switches. Therefore, we obtain the final value for the number of required core switches as,

$$n_{\text{core}}^{\text{total}} = \min \left\{ k^2, n_{\text{core}}^{\text{ext}} + n_{\text{core}}^{\text{addl}} \right\}$$

Finally, let us compute the number of aggregation switches required. In each pod, each server sends $(1-p_1)\lambda$ amount of internal traffic to the aggregation layer. An additional q amount of external traffic is also sent. Therefore, the total amount of traffic within each pod that reaches its aggregation layer is $((1-p_1)\lambda + q)k^2$. Each aggregation switch has k interfaces connected to edge switches. Therefore, the number of aggregation switches required per pod is,

$$\left\lceil \frac{((1-p_1)\lambda + q)k^2}{k} \right\rceil$$

Yielding the total number of required aggregation switches in the network as,

$$n_{\text{aggr}}^{\text{total}} = 2k \min\left\{k, \lceil ((1-p_1)\lambda + q)k \rceil \right\}$$

Combining all the derived values, we obtain the total number of active switches in the extended model as,

$$\text{Active}_{\text{Extended}} = 2k^2 + n_{\text{aggr}}^{\text{total}} + n_{\text{core}}^{\text{total}} \tag{3.3}$$

In Figure 3.3 we plot the fraction of active switches as a function of $\lambda + q$ for the case when $q = 0.25$, $k = 6$ and $Q = 12$. In this case, the external capacity of each core switch is more than sufficient to handle all external traffic. In the figure, point A denotes the case when there is no internal traffic at all and all traffic is external. Here, all the edge switches are active and in addition, 9 core switches are required for external traffic. Each of these 9 core switches have *zero* available capacity to handle internal traffic. Therefore, as we now start increasing internal load λ we require additional core switches to become active when $(1 - p_1 - p_2) > 0$. Observe that the case when $p_1 = p_2 = 0.5$ only requires additional aggregation switches after λ exceeds 0.2. Prior to that we can get away with using the aggregation switches that are already active for external traffic.

It is instructive to contrast the above figure with Figure 3.4 where we now have $Q = 6$ and $q = 0.5$. This represents a case where the external traffic accounts for 50% of all traffic handled by the network and the number of external interfaces is smaller. Point A again denotes the case when there is no internal traffic. The *-o*

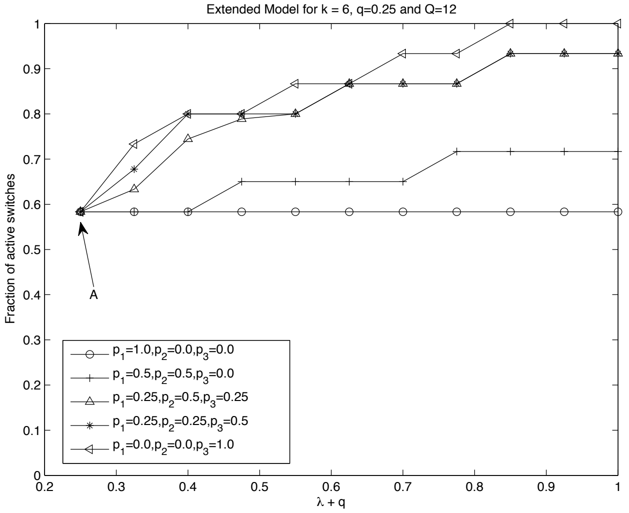


Figure 3.3. Extended model with high external connectivity.

data set corresponds to the case when all internal traffic confined to edge switches only. As we increase the traffic within the pod but between edge switches (data shown by -+) we see an increase in aggregation switches used but no change in core switches. The remaining three plots correspond to the case where we slowly increase the amount of inter-pod internal traffic. This causes an increase in number of core switches required until the point where the internal inter-pod traffic plus the external traffic exceeds the capacity of the network. To understand this further, let us derive the expressions for traffic loss.

First note that there are no traffic losses in the basic model without external traffic since the *fat-tree* has full bisection bandwidth. In the extended model, traffic losses will not occur within the pod if $\lambda + q < 1$. However, traffic losses occur in the core if external capacity Q is unable to handle the external load.

Traffic losses at the core can potentially be divided into two types. The first type corresponds to losses to the external traffic and happens if,

$$Qk^2 < 2qk^3$$

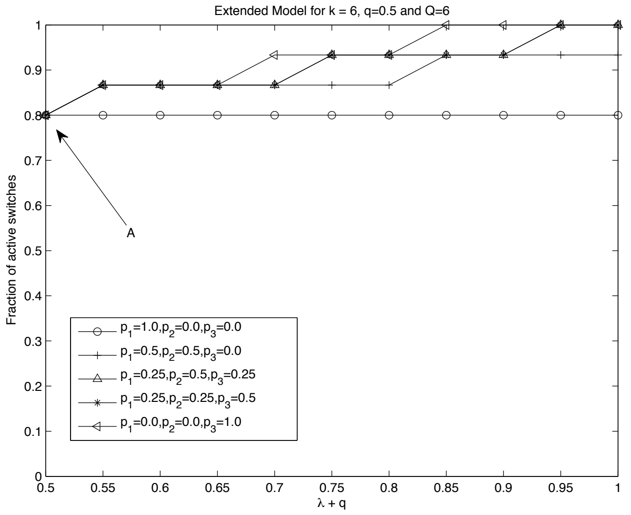


Figure 3.4. Extended model with reduced external connectivity and high external load.

This yields a loss of external traffic of,

$$\text{Loss}_{\text{ext}} = \max\{0, (2qk^3 - Qk^2)\}$$

The second source of losses is from internal traffic if the residual capacity of the core switches (after handling external traffic) is insufficient for internal traffic. Recall that the total internal traffic coming to the core layer is $2(1-p_1-p_2)\lambda k^3$. Equation 3.2 gives us the number of additional core switches required to handle this traffic. Therefore the amount of internal traffic loss is,

$$\begin{aligned}\text{Loss}_{\text{int}} &= \max\{0, 2(1-p_1-p_2)\lambda k^3 - \\ &((2k-Q)\lfloor n \rfloor + (2k - (2qk^3 - \lfloor n \rfloor Q))) - 2k(k^2 - n_{\text{core}}^{\text{ext}})\}\end{aligned}$$

Theorem 1: In the extended model $\text{Loss}_{\text{int}} = 0$.

Proof sketch: (We have not included the formal proof here for space reasons) The intuition behind this result is relatively simple. Consider traffic going up to the core first. Since $\lambda + q < 1$ there will be no losses seen by any of the traffic either in the pods or in the inputs to the core switches. Traffic heading out of the core

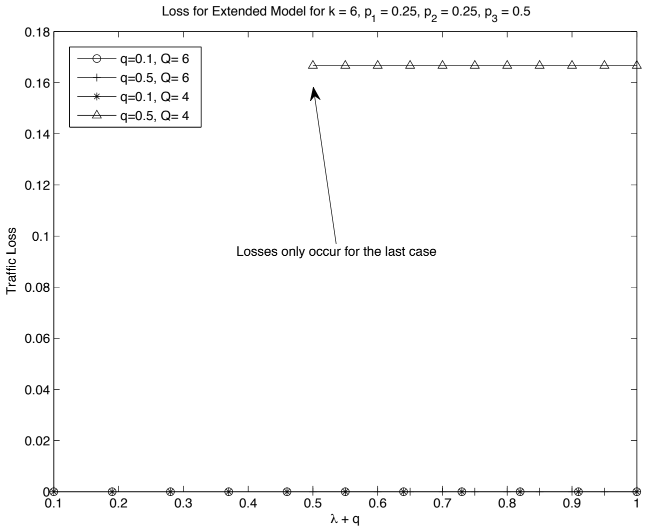


Figure 3.5. Traffic loss corresponding to Figure 3.4.

to the Internet is limited by Q and hence we may see packet drops if $2kq > Q$. Consider now traffic coming into the network from the Internet as well as inter-pod and intra-pod traffic. At the core layer, this traffic will be $2(1-p_1-p_2)\lambda k^3 + 2k^3 q'$. The first term is the inter-pod traffic and the second term is the amount of external traffic that *was not lost* due to the limitation on Q. Clearly, $q' \leq q$ and hence the total traffic flowing into the servers is below the link capacity of one and there will again be no losses. Therefore, we can write the total loss in the extended model as,

$$\text{Loss}^{\text{extended}} = \text{Loss}_{\text{ext}}$$

Figure 3.5 plots the fraction of traffic loss (total traffic lost divided by the number of servers) versus $\lambda + q$ for various cases. Note that there are no losses when $q = 0.1$ for $Q = 4, 6$. The only case we see losses is when $q = 0.5, Q = 4$ since there is not sufficient external capacity.

3.1.3 Asymmetric Model

The extended model above assumes that every core switch has a link to a border gateway for connectivity to the Internet. This assumption may be reasonable for smaller datacenter networks but for larger ones, the more likely scenario is one where only a few of the core switches have external links. Let us assume that of the k^2 core switches, $k \geq C \geq 1$ have external connectivity via links of capacity Q. Assume further that these C switches are connected to the aggregation switches using links of capacity $l \geq 1$. All remaining links in the network have a capacity of 1. Clearly, $Q \leq 2kl$ and $l \leq k$. The latter inequality makes sense since an aggregation switch is connected to k edge switches with capacity one links and thus there is little point in connecting it to a core switch by a link of capacity greater than k. Without loss of generality, assume that the C core switches are $1, 1+k, 1+2k, \cdots, 1+(C-1)k$. Thus aggregation switches $1, \cdots, C$ in each pod are connected with a link of capacity l to these special core switches.

As before, assume that the total external traffic load is $2qk^3$ and the traffic is uniformly distributed among all the servers. The total number of externally connected core switches we need to be active is thus given by,

$$m = \frac{2qk^3}{Q}$$

Since m may be greater than C or have a fractional part, we obtain,

$$m_{\text{core}}^{\text{ext}} = \min\{C, \lceil m \rceil\}$$

If the external traffic exceeds the capacity of the network to handle it, then we can compute the loss as,

$$\text{Loss}_{\text{ext}}^{\text{Asym}} = \{0, 2qk^3 - CQ\}$$

We proceed as in the previous section to compute the number of additional core switches needed to support *internal* traffic. Recall that the externally connected core switches may not be using all of their link capacity and thus they can be used for routing internal traffic as well.

Each of the $2k$ interfaces of the active externally connected core switches has a capacity of l. If $m_{\text{core}}^{\text{ext}} = \lceil m \rceil$ then $\lfloor m \rfloor$ of these switches are using their full external capacity of Q leaving $(2kl - Q)\lfloor m \rfloor$ free capacity. For example, a switch may have $2k = 12$ one gigabit links and one $Q = 10$ gigabit link connected externally. Thus this switch has 2 gigabits of free capacity that can be used for routing internal traffic.

One additional externally connected core switch will be using less capacity for external traffic leaving $(2kl - (2qk^3 - \lfloor m \rfloor Q))$ free capacity. The total *free* capacity is thus $f = (2kl - Q)\lfloor m \rfloor + (2kl - (2qk^3 - \lfloor m \rfloor Q))$. If $C < \lceil m \rceil$, however, then all the $m_{\text{core}}^{\text{ext}}$ switches are using their full external capacity Q leaving only $f = (2kl - Q)C$ free capacity. The total internal traffic that needs to be forwarded by core switches is $2(1 - p_1 - p_2)\lambda k^3$. Therefore, the number of additional core switches we need is,

$$m_{\text{core}}^{\text{addl}} = \begin{cases} 0, \ \text{if } 2(1 - p_1 - p_2)\lambda k^3 \leq f \\ \left\lceil \frac{2(1-p_1-p_2)\lambda k^3 - f}{2k} \right\rceil, \ \text{otherwise} \end{cases}$$

We divide the second term above by $2k$ since that is the degree of the additional core switches used. Since it is possible that the above number exceeds the available number of free core switches, we can write the final answer as,

$$m_{\text{core}}^{\text{total}} = \min \left\{ k^2, m_{\text{core}}^{\text{ext}} + m_{\text{core}}^{\text{addl}} \right\}$$

Let us next compute the number of aggregation switches required per pod. Within each pod, the total external traffic is qk^2 and this is forwarded to/from the externally connected core switches using $m_{\text{core}}^{\text{ext}}$ aggregation switches. This is the case because of the way we are performing the minimization forces traffic from/to each pod to be identically routed. The total internal traffic that needs to be handled by the aggregation switches is $(1 - p_1)\lambda k^2$.

Consider the aggregation switches in a pod that are connected to the active externally connected core switches. Say the high-capacity link (of capacity l) carries traffic a. This traffic is evenly distributed over the k capacity 1 links connecting this aggregation switch to edge switches. In other words, each of the edge switches can send up to $(1 - a/k)$ internal traffic to this aggregation switch. In all, the k connected edge switches can send $(k - a)$ total internal traffic to this aggregation switch. Consider next the k links from this switch connected to the core switches. One of the links is capacity l while the others are capacity 1 each. Thus, the total available capacity of these links is $(l-a)+(k-1) = (k-a)+(l-1)$. Since $k - a < (k - a) + (l - 1)$ the total internal capacity that can be handled by this aggregation switch is $k - a$.

In a pod, if $m_{\text{core}}^{\text{ext}} = \lceil m \rceil$ then there are $\lfloor m \rfloor$ aggregation switches where $a = Q/2k$ (corresponding to the $\lfloor m \rfloor$ core switches that run their external links at full capacity) and at most one switch $(\lceil m \rceil - \lfloor m \rfloor)$ where $a = (2qk^3 - Q\lfloor m \rfloor)/2k$. These $\lceil m \rceil$ aggregation switches handle internal traffic equal to,

$$t_{\text{aggr}} = \lfloor m \rfloor (k - \frac{Q}{2k}) + (\lceil m \rceil - \lfloor m \rfloor)(k - (2qk^3 - Q\lfloor m \rfloor)/2k$$

leaving $(1-p_1)\lambda k^2 - t_{\text{aggr}}$ to be handled by other aggregation switches. If, however, $C < \lceil m \rceil$ then $t_{\text{aggr}} = (k - Q/2k)C$. In all, in the entire network, the total number of aggregation switches needed is then given by,

$$m_{\text{aggr}}^{\text{total}} = 2k \min \left\{ k, m_{\text{core}}^{\text{ext}} + \left\lceil \frac{(1-p_1)\lambda k^2 - t_{\text{aggr}}}{k} \right\rceil \right\}$$

Putting all these values together we have,

$$\text{Active}_{\text{Assymetric}} = 2k^2 + m_{\text{core}}^{\text{total}} + m_{\text{core}}^{\text{addl}} \tag{3.4}$$

Recall that $C \leq k$ in the asymmetric model therefore, $q \leq QC/2k^3$. Let us assume that $C = k = 6$. In Figure 3.6 we plot the fraction of active switches versus

$\lambda + q$ when $q = 0.25, Q = 12, C = 6, k = 6, l = 1$. We observe that even when $\lambda = 0$ we are using over 80% of the switches. The reason for this has to do with the fact that when $m_{\text{core}}^{\text{ext}}$ switches are active in the core, *an equal number of aggregation switches are forced to be active in each pod* since they have a high capacity link to these externally connected core switches. In this particular example, therefore all $C = 6$ externally connected core switches are active. No additional switches are needed since there is no internal traffic. In each of the $2k = 12$ pods, six aggregation switches are also active and all the edge switches are active. Therefore we get 6+72+72 = 150 active switches or 83%. The switches are essentially idle but due to the external traffic and the topology of the network we are paying this high cost.

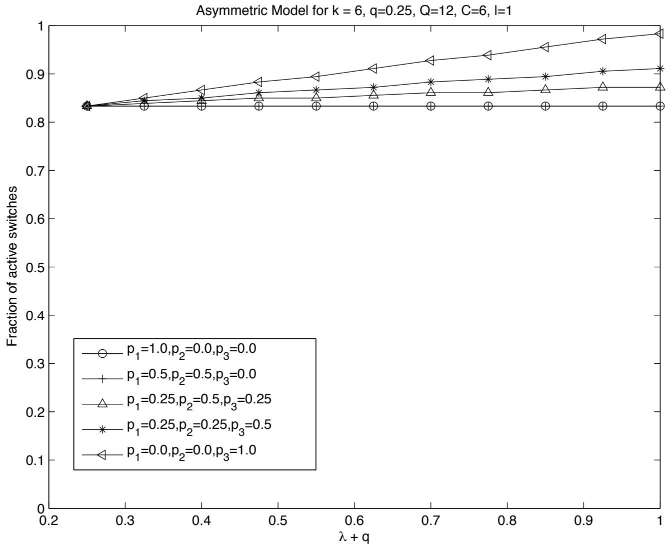


Figure 3.6. Asymmetric model with high external traffic.

Let us consider a more realistic case where the external links are $Q = 40$ gbps and $l = 10$ gbps for the $C = 6$ externally connected core switches. Figure 3.7 plots the fraction of active switches and we note that in this case only 3 of the $C = 6$ core switches was used resulting in an overall reduction in number of aggregation

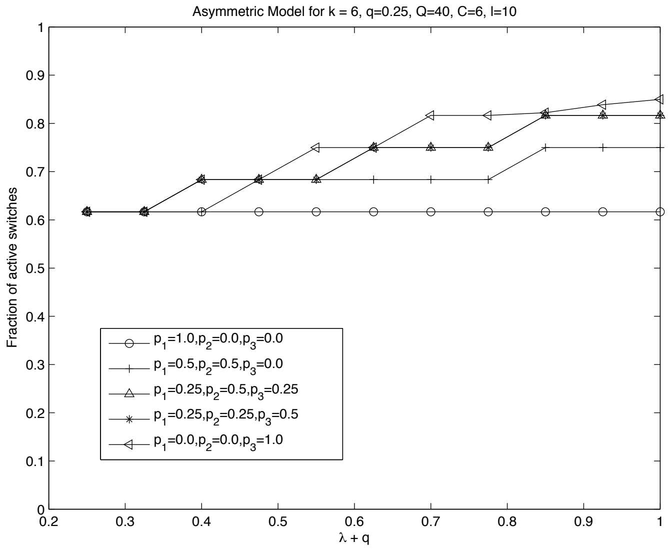


Figure 3.7. A more realistic asymmetric model with high capacity external links.

switches needed. Also, since $2kl = 120$ gbps for each of these C switches, there is $120 - 40 = 80$gbps excess capacity that can be used to forward internal traffic. Unfortunately, as in the previous example, each pod has at least $m_{\text{core}}^{\text{ext}}$ active aggregation switches and thus even when $\lambda = 0$ we use 62% of switches.

Finally, let us consider traffic loss for the asymmetric model. As in the case of the extended model, we can state that the only traffic loss will occur at the core switches when $\lambda + q < 1$. Thus,

$$\text{Loss}^{\text{Asym}} = \text{Loss}_{\text{ext}}^{\text{Asym}}$$

Figure 3.8 plots the traffic lost (total traffic lost divided by number of servers) for the case corresponding to Figure 3.6. Regardless of internal traffic patterns the traffic loss is the same. This makes sense since only external traffic $q = 0.25$ will be lost due to insufficient external bandwidth.

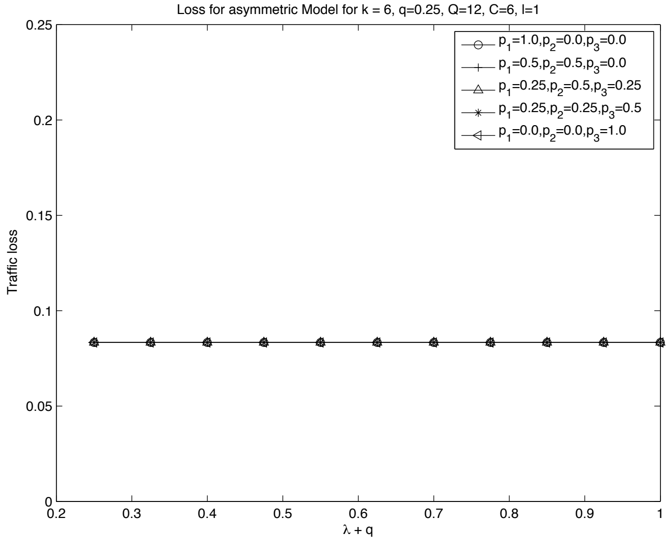


Figure 3.8. Traffic loss for the model in Figure 3.6.

3.1.4 Summary

The derivations above point to a few significant areas where energy is being squandered. The first is the cost of running edge switches all the time even when there is minimal load. This contributes a large constant to the overall energy cost. Second, external traffic to the datacenter can cost a lot of energy as shown in the asymmetric case. In the examples described, we end up using more aggregation switches than necessary due to the topology.

In order to reduce energy consumption and make it linear for low loads, we need to modify the edge topology of these networks without sacrificing the full throughput to support high loads. To deal with the challenge of supporting external traffic while not using unnecessary aggregation switches, we need to consider minimizing C. For example, by making $C = 1$ but boosting its link speeds dramatically, we can ensure that only one aggregation switch per pod needs to be active for a wider range of loads. This fact is illustrated in Figure 3.9 where $C = 1$ but the external capacity is 120 gbps with each of its 12 links running at 10 gbps. This one core

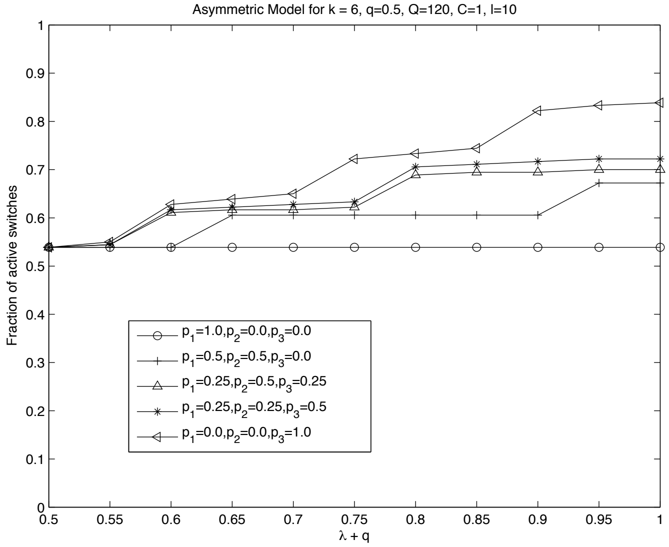


Figure 3.9. Using only one externally connected switch.

switch now carries a significant amount of internal traffic as well and thus even a $\lambda + q = 1$ we see less than 100% active switches.

3.2 SIMULATIONS

We built a simulator for a *fat-tree* network with $k = 6$ and 1 gbps link capacity. We also use $C = 1$ and designate the leftmost core switch as the externally connected core switch. In the simulator, we read trace files generated externally and forward packets based on routing tables computed every second of simulated time. The routing algorithm is a modified version of Dijkstra's algorithm where we force flows to use routes that are already in use, thus packing flows together. In the algorithm we assign weights to edges as well as nodes. Edge weights are constant but node weights can be 0 or 1. If a node has been used for forwarding a flow, its weight changes from 1 to 0. Thus, flows are encouraged to reuse the same subset of nodes (or switches). Of course, if adding a new flow over a link will exceed the link's capacity we eliminate that link from further consideration in that round of routing computation.

We used the traffic models developed in [11] to analyze our algorithm. The two models that are most relevant for datacenters are the *stride* and the *staggered* models. Imagine numbering the $2k^3$ servers consecutively starting from 1. In stride(s), packets are send from server i to server $i + s \mod 2k^3$. Thus, stride(1) tends to mainly send traffic between servers connected to the same edge switch (high p_1 in our analytical model) while stride(6) generates mainly traffic within a pod but between edge switches (high p_2) and stride(36) is mainly inter-pod traffic. Staggered traffic is very similar to the traffic model we used for our analysis and is specified using the same probabilities p_1 and p_2. The seven different traffic models we used are as follows:

1. stride(1), stride(6), stride(36), stride(216)
2. staggered(1) $p_1 = 1.0$, $p_2 = 0.0$
   staggered(2) $p_1 = 0.5$, $p_2 = 0.3$
   staggered(3) $p_1 = 0.2$, $p_2 = 0.3$

We assume that external traffic q is 10% in all cases.

In Figure 3.10 we plot the fraction of active switches versus total load using simulation for the staggered data and in Figure 3.11 we plot the same metric using the asymmetric model from the previous section (but using the staggered packet trace parameters). It is easy to see that our model is a very close match to the simulations. *The implication of this is that the lower bound of energy efficiency can be achieved in practice by utilizing the simple routing algorithm described above.*

Figure 3.12 plots the fraction of active switches versus load for the stride case using simulation. Figure 3.13 plots the same data using our asymmetric analytical model. Again note the closeness of the two models. The 5% error between the

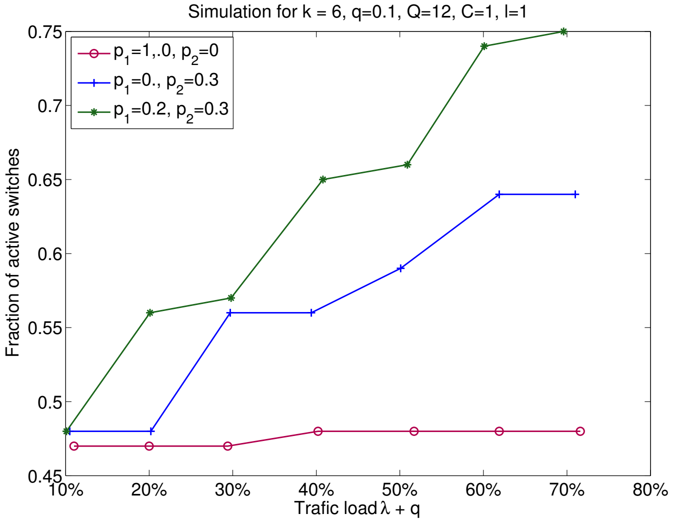


Figure 3.10. Fraction of active switches for the staggered model.

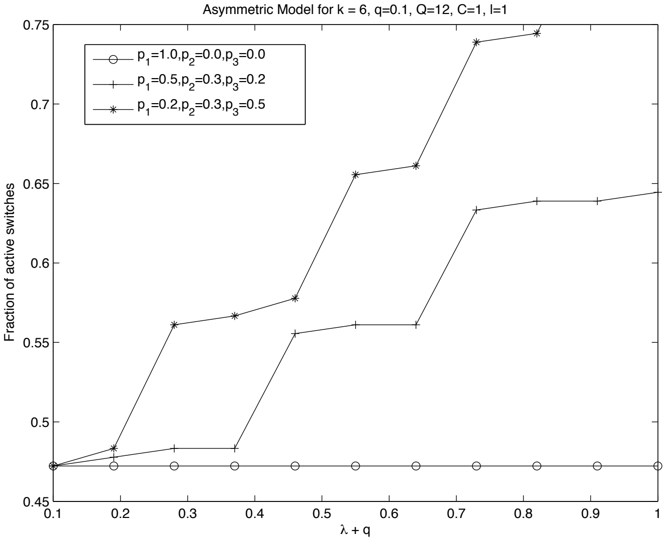


Figure 3.11. Fraction of active switches for the analytical model (staggered cases).

simulations and analysis is due to the fact that we estimated the values of p_1 and p_2 from the simulations and then used them in the analysis. The estimated values for these probabilities are listed in the legend of Figure 3.13. One noteworthy feature of the stride model is that there is no difference between stride(36) and stride(216). This makes sense because in both cases packets are inter-pod. The difference between stride(6) and stride(36) is that most packets in stride(6) remain within one pod and thus only one core switch is used.

When we examine Figures 3.10 to 3.13, we observe that the type of loading has a significant impact on energy consumption. While stride(1) and staggered(1) may be impractical for many distributed applications, we see that stride(6) and staggered(2) are better choices than stride(36) and staggered(3). This means that when it comes to allocating tasks to servers, the task manager should be mindful of the type of traffic that will be generated since we can obtain significant energy savings by careful scheduling.

3.3 SUMMARY

In this chapter, we provide a systematic analysis of the energy efficiency of a *fat-tree* network topology using modeling and detailed simulations. The key question we ask is *how does energy usage scale with total load as well as with different types of loading.* To answer this question, we build a detailed analytical model that gives the *lower bound* on the *fraction of active switches* required for a given load and type of load. We show that *fat-trees* have a minimal cost of about 40-50% (i.e., about half the switches need to remain active at all times) but beyond that, the lower bound scales almost linearly with total offered load. The lower bound computation is based on aggregating traffic into few routes. We conducted a detailed simulation of a *fat-tree* network where we used different types of load (staggered and stride) [11] and different amounts of total load. We compute routing tables empirically every second for the next second and compute the fraction of needed active switches. The simulation shows that the model we develop is accurate in predicting switch activity and the simulation demonstrates that by modifying the routing algorithms we can potentially save significant amounts of energy in real networks. By developing analytical models for energy consumption, datacenter researchers are able to study fat-tree DCNs theoretically. A practical application of our work would be to jointly optimize task scheduling and flow assignment so as to maximize the traffic consolidation for given job loads. In this research, we first provide a systematic analysis of the energy efficiency of a *fat-tree* network topology using modeling and detailed simulations. The key question we ask is *how does energy usage scale with total load as well as with different types of loading.* To answer this question, we build a detailed analytical model that gives the *lower bound* on the *fraction of active switches* required for a given load and type of load. We show that fat-trees have a minimal cost of about 40-50% (i.e., about half the switches need to remain active at all times) but beyond that, the lower bound scales almost linearly with total offered load. The lower bound computation is based on aggregating traffic into few routes. We conducted a detailed simulation of a *fat-tree* network where we used different types of load (staggered and stride) [11] and different amounts of total load. We compute routing tables empirically every second for the next second and compute the fraction of needed active switches. The simulation shows that the model we develop is accurate in predicting switch activity and the simulation demonstrates that by modifying the routing algorithms we can potentially save significant amounts of energy in real networks.

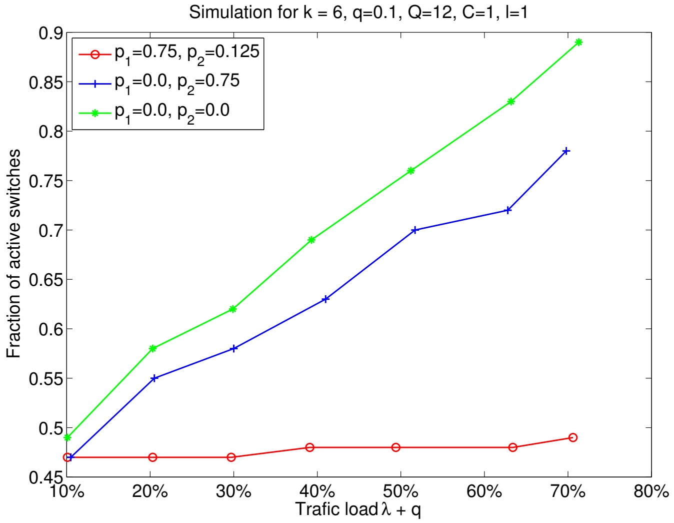


Figure 3.12. Fraction of active switches for the stride model.

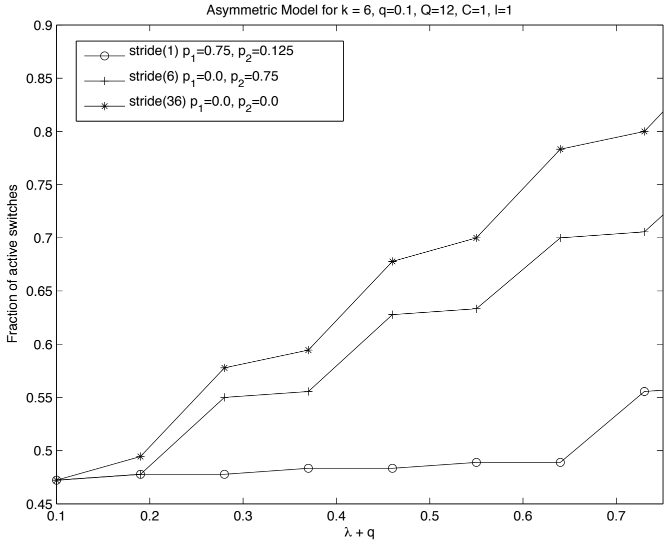


Figure 3.13. Fraction of active switches for the analytical model (stride cases).

Chapter 4

ANALYTICAL OPTIMIZATION MODEL

To compute the minimal power required by a datacenter network, we will compute the minimum subset of network elements of network infrastructure. For a given traffic load, we need to find optimal route assignments of traffic flows that involves minimum number of switches and links. This optimization problem is in general a mixed-integer programming problem (MIP), and can be integrated into a capacitated minimum cost multi-commodity network flow problem (MCMCF).

In this chapter, we examine the power optimization model with the goal of minimizing energy consumption of datacenter networks. We implement the power model using commercial optimization software. For a more scalable implementation, we propose a heuristic algorithm that finds a near-optimal subset of network switches and links that satisfies a given traffic load and consumes minimal power. We demonstrate that this simple routing algorithm can approximate the optimization model very closely and it can be applied to large-scale datacenter networks to achieve optimal subset of networks with minimum overhead.

4.1 MINIMIZING ENERGY CONSUMPTION

For a datacenter network, we formulate a power model for all network elements including switches and links. A network $G(V, E)$ is given, where V is the set of nodes in the network and E is the set of links. We consider both the end hosts and the switches as network nodes and thus we have $V = V_1 + V_2$, where V_1 is the set of end hosts and V_2 is the set of switches. Link $(u, v) \in E$ connects node u and node v $(u, v \in V)$. Assuming each switch consumes power P_s and each link consumes power P_l, the total power consumed by the entire network can be expressed as

$$P_{total} = \frac{1}{2} \sum_{u \in V_2} k_u \times P_l + n \times P_s + \frac{\epsilon}{2} \times \sum_{u \in V, w \in V_u} f_{u,w} \tag{4.1}$$

where n is the number of active switches and k_u is the number of active interfaces of switch u. V_u is the set of nodes connecting to node u. ϵ is the dynamic energy consumption factor representing the power consumption per unit data transmitted through a link. $f_{u,v}$ is amount of traffic flow assigned to link (u, v). We use binary variables y_u and $x_{u,v}$ to represent the power state of node u and link (u, v), respectively. For instance, if $x_{u,v} = 1$, link (u, v) is active; if it is 0, link (u, v) is idle and can be powered off. Therefore, k_u and n can be written as

$$n = \sum_{u \in V_2} y_u \tag{4.2}$$

$$\forall u \in V_2, \quad k_u = \sum_{w \in V_u} x_{u,w} \tag{4.3}$$

4.1.1 Optimization Model

Based on the power model defined above, we define an optimization problem in order to find the optimal flow assignment that involves a minimum subset of active network elements, (n, k_u), with the minimal total power consumption P_{total} for a given network topology and a traffic load. This optimization problem is an extension to the capacitated minimum-cost multi-Commodity Flow problem (MCMCF). A classical MCMCF problem is subject to three constraints - *capacity constraint* (4.4), *flow conservation constraint* (4.5) and *demand satisfaction constraint* (4.6) written as follows

$$\forall (u, v) \in E, \quad f_{u,v} \leq c x_{u,v} \tag{4.4}$$

$$\forall u,\ u \notin S \ and \ \ u \notin D,\ \ \sum_{w \in V_u} f_{u,w} - \sum_{w \in V_u} f_{w,u} = 0 \tag{4.5}$$

$$\begin{cases} \forall s \in S,\ \sum\limits_{w \in V_s} g^i_{s,w} - \sum\limits_{w \in V_s} g^i_{w,s} = t^i_{s,d} \\ \forall d \in D,\ \sum\limits_{w \in V_d} g^i_{w,d} - \sum\limits_{w \in V_d} g^i_{d,w} = t^i_{s,d} \end{cases} \tag{4.6}$$

where c is the capacity for each link. S is the set of source nodes and D is the set of destination nodes. V_s and V_d is the set of switches that connect to source node s and sink node d, respectively. $f_{u,w}$ is the total flow assigned on link (u, w) and $f_{u,w} = \sum\limits_{i} g^i_{u,w}$, where $g^i_{u,v}$ represents the flow of the ith traffic demand $t^i_{s,d}$ routed through link (u, v).

Capacity constraint (4.4) takes account of maximum link utilization and ensures that the total traffic flow assigned to a link does not surpass the link capacity. The *capacity constraint* also forces flows to go through active links only. For example, inactive link (u, v) has $x_{u,v} = 0$, which causes $f_{u,v} = 0$ meaning no traffic flow is assigned to this link. *Flow conservation* (4.5) ensures that traffic entering an intermediate node equals to traffic exiting from it. *Demand satisfaction* (4.6) describes that the overall traffic departing a source node or entering a destination node equals to the traffic demand.

Besides these three constraints, the *bidirectional link* rule ensures that both directions of a link are powered on if there is a flow assigned to either direction of the link. The *bidirectional link* constraint is expressed as

$$\forall (u, v) \in E, \quad x_{u,v} = x_{v,u} \tag{4.7}$$

Additionally, we include constraints that correlate the power states of switches and links. For each node u and the connected links (u, w) and (w, u), we have

$$\forall u \in V,\ \forall w \in V_u,\ x_{u,w} \le y_u \ \ and \ \ x_{w,u} \le y_u \tag{4.8}$$

$$\forall u \in V, \;\; y_u \leq \sum_{w \in V_u} (x_{u,w} + x_{w,u}) \tag{4.9}$$

Constraint (4.8) makes sure that a switch is powered off only when all its connected links are powered off, and constraint (4.9) ensures that a switch be powered off when all its connected links are powered off. Optionally, we can include a *non-splitting* constraint as follows to prevent flow splitting:

$$\forall i, \forall (u,v) \in E, \;\; g^i_{u,v} = t^i \times r^i_{u,v} \tag{4.10}$$

where $r^i_{u,v}$ is a binary decision variable that indicates whether the traffic demand t_i is assigned to link (u,v). Constraint (4.10) ensures that $g^i_{u,v}$, the flow assignment to link (u,v), is either equal to the ith traffic demand t_i or equal to zero.

Furthermore, we define heuristic constraints to reduce the problem size. For example, since a k-ary fat-tree network has $5k^2/4$ switches and each switch has at most k active links, we explicitly apply an upper bound and a lower bound to k_u and n as $0 \leq k_u \leq k$ and $0 \leq n \leq \frac{5}{4}k^2$, which can greatly improve convergence time for the problem.

We implement the power optimization model using CPLEX, which is an optimization solver for integer programming problems. For a given traffic matrix, the optimization model outputs the numbers of active switches and links, and the flow assignment to each link corresponding to every traffic flow demand. Our model is implemented with both flow-splitting and non-flow-splitting options.

4.2 GREEDY FLOW ASSIGNMENT

Through the formal power optimization model, we can find the optimal flow assignment for a given network topology and traffic loading. However, noticing that mixed integer programming is known to be strongly NP-hard, a MCMCF problem for a large-sized datacenter network cannot be solved within a reasonable time frame. To address this problem, we propose a heuristic greedy algorithm to find a near-optimal flow assignment.

4.2.1 Heuristic Algorithm

Our greedy flow assignment algorithm is based on the Dijkstra's algorithm that solves the shortest path problem. For each traffic flow, the algorithm finds a route with sufficient bandwidth between the source node and the destination node with the lowest cost. The cost function is defined as the sum of the cost of switches and links along the route. By carefully defining the cost value of each node and each link, our greedy algorithm finds the lowest-cost route for each flow incrementally, and ultimately obtains the optimal routing for all the flows. The greedy algorithm is described as in Algorithm 1.

Each link and each node has a *fixed capacity.* We only assign a flow to a link when there is available bandwidth at that link and also at the source and destination node. Once a flow is assigned, the corresponding amount of traffic demand is subtracted from the bandwidth of the link and the nodes on both ends. Link cost $cost(u, v)$ is defined as a constant value of 2 for all links while node cost $cost(v)$ is initialized as 1. Each link is counted in the cost of the route and we are ensured to find the shortest route. *Once node v is used in a route once, $cost(v)$ is updated to* 0. This makes sure that a switch that has been used in a previous route will have higher priority to be reused. As a result, we can achieve the minimum overall number of active switches. We use higher link cost than node cost in order to avoid detour routes between switches.

The greedy algorithm is not optimal, but we verified that the results produced by the algorithm are very close to those from the CPLEX optimization solver in Section 3 for all traffic types and loads. However, the optimization model can only

Algorithm 1 Flow Assignment algorithm

```
function FLOWASSIGN($source, sink, demand$)
    for each vertex $v$ in $Graph$ do
        $dist[v] \leftarrow Infinity$
    $dist[source] \leftarrow 0$
    insert $(source, dist[source])$ to $Q$
    while $Q$ is not empty do
        $u \leftarrow$ first pair in $Q$
        if $u == sink$ then
            break
        for each neighbor $v$ of $u$ do
            if $(capacity(u, v)! = 0)$ and
             $(capacity(u)! = 0)$ then
                $alt \leftarrow dist[u] + cost(v) + cost(u, v)$
            else
                $alt \leftarrow Infinity$
            if $alt < dist[v]$ then
                $dist[v] \leftarrow alt$
                $previous[v] \leftarrow u$
                update $(v, dist[v])$ in $Q$
    for $v = sink; v! = -1; v = previous[v]$ do
        insert $v$ to $route$
    return $route$
```

scale to $k = 4$ fat-tree networks. In the next part of this paper, we use this greedy algorithm to simulate larger scale fat-tree networks.

4.2.2 Validation of Greedy Algorithm

The greedy algorithm is not optimal but, as we show below, the routes produced by the algorithm are very close to those produced by solving the optimization formulation using CPLEX optimization solver in section 4.1. We use fat-tree topology with $k = 4$ and generate a number of packet traces following certain datacenter network traffic patterns [20]. The packet traces in each one-second interval are organized as a traffic matrix and is fed into the CPLEX optimization model and the simulated greedy algorithm. We obtain the number of active switches and active interfaces for the eight traffic patterns and seven traffic loads shown in Table 4.1.

Table 4.1. Number of active switches and active interfaces from optimization model vs. simulation with greedy algorithm.

load	Random				Staggered(1)			
	active switches		active interfaces		active switches		active interfaces	
	opt	greedy	opt	greedy	opt	greedy	opt	greedy
10%	13	13	40	40	8	8	16	16
20%	13	13	40	40	8	8	16	16
30%	13	14	40	44	8	8	16	16
40%	14	14	48	48	8	8	16	16
50%	14	14	48	48	8	8	16	16
60%	18	19	64	72	8	8	16	16
70%	19	19	72	72	8	8	16	16

load	Staggered(2)				Staggered(3)			
	active switches		active interfaces		active switches		active interfaces	
	opt	greedy	opt	greedy	opt	greedy	opt	greedy

10%	13	13	40	40	13	13	40	40
20%	13	13	40	40	13	13	40	40
30%	13	13	40	40	13	13	40	40
40%	13	13	40	40	13	13	40	40
50%	13	13	40	40	14	14	48	47.2
60%	13	13	40	40	14	14	48	53.4
70%	13	13	40	40	18	19	64	72

load	Stride(1)				Stride(2)			
	active switches		active interfaces		active switches		active interfaces	
	opt	greedy	opt	greedy	opt	greedy	opt	greedy
10%	13	13	40	40	13	13	40	40
20%	13	13	40	40	13	13	40	40
30%	13	13	40	40	13	13	40	40
40%	13	13	40	40	13	13	40	40
50%	13	13	40	40	17	17	58	56.2
60%	13	13	40	40	18	18	64	64
70%	13	13	40	40	19	18	66	64

load	Stride(4)				Stride(8)			
	active switches		active interfaces		active switches		active interfaces	
	opt	greedy	opt	greedy	opt	greedy	opt	greedy
10%	13	13	40	40	13	13	40	40
20%	13	13	40	40	13	13	40	40
30%	14	14	48	48	14	14	48	48
40%	14	14	48	48	14	14	48	48
50%	17	17	60	60.8	17	17	60	63.6
60%	19	19	72	72	19	20	72	75.2
70%	19	19	72	72	19	20	72	75.6

The results we get from the simulated greedy algorithm are very close to those get from the CPLEX optimization model, especially for the lighter loads. Since the optimization model can only scale to a fat-tree datacenter network with $k = 6$, we use the greedy algorithm to simulate the optimization of a large-scale fat-tree network in the following chapters of this paper.

4.3 SUMMARY

Inspired by the earlier work of Gupta *et al.* [48], many researchers propose energy-proportional datacenter network topologies through topology-aware heuristics to find optimal subset and power off idle interfaces or devices. For example, ElasticTree *et al.* [51] leverages the regularity of hierarchical datacenter networks and uses left-most heuristics to find the smallest topology. *CARPO* [81] examines the dynamic topology by consolidating timely-negative-correlated flows into a smaller set of links and shutting off unused ones. Instead, we propose a universal greedy flow assignment algorithm to find the optimal network subset. Our algorithm can find near-optimal flow assignments comparable to solutions achieved from MIP model solver, for not just hierarchical network topologies, but also random or irregular datacenter network topologies. In addition, our approach is proved to be able to achieve energy conservation based on real-time traffic load.

Chapter 5

USAGE-BASED DATACENTER NETWORK TOPOLOGY

In this chapter, we construct a datacenter network (DCN) topology that supports the expected loading for different application domains but incurs a lower energy cost. Specifically,

1. We first begin with fat-tree network and examine the sub-graphs of these networks that are used for loadings as high as 70% for different types of applications (educational, cloud, and private data centers) when using left-most routing as in [51]. The results indicate that we can indeed reduce the number of switches by 50% at the aggregation and core layers of the network without incurring any loss or increased latency.

2. Next we consider the possibility of moving flows to fewer servers, particularly for low loads. This approach is interesting since it can inform job schedulers about how and where to place jobs in order to minimize network energy cost. Consolidating flows further reduces the needed switches in the network by up to 10%.

3. Our analysis shows that edge switches (i.e., switches connected to servers) account for a high energy cost as they are always powered on, even at tiny loads. Given that a significant energy cost of a switch is static (in the chassis, power supply, processor, interconnect fabric), by using high cardinality

switches (and thus fewer switches) we can save significant amount of energy even if they are always on.

Putting all these studies together we obtain a new DCN in which edge switches have high port density and where the other switches are connected in a left-skewed topology which is a subgraph of the fat-tree. This type of topology has a lower capital cost and lower operational cost as well.

5.1 SUB-TREES FOR DIFFERENT TRAFFIC CHARACTERISTICS

A typical fat-tree DCN consists of three layers of switches: edge layer, aggregation layer and core layer. For a k-ary fat-tree network, there are $\frac{k^2}{4}$ switches in the core layer. The aggregation layer and edge layer are divided into k pods, each of which has $\frac{k}{2}$ edge switches and $\frac{k}{2}$ aggregation switches. In total, the network is composed of $\frac{5k^2}{4}$ switches and each switch has k ports. Every edge switch is connected to $\frac{k}{2}$ end hosts, thus $\frac{k^3}{4}$ end hosts in total can be interconnected through the fat-tree network.

5.1.1 Traffic Model

Benson *et al.* [20] analyzed network traffic characteristics of ten Datacenters, including three university data centers (EDU), two private enterprise Datacenters (PRV) and five commercial cloud Datacenters (CLD). EDU Datacenters serve students and staff on campus. The main applications in EDU data centers include distributed file systems and Web services. PRV Datacenters mainly serve corporate users and developers. Besides hosting traditional Web services, these data centers also run customized applications. CLD Datacenters are purposely-built to support specific applications and serve external users. For example, two of the CLD Datacenters primarily run MapReduce style jobs and the other three are mainly for Internet-facing applications, including Messaging, Webmail, Web portal and searching.

By observation, a significant part of the traffic in the EDU data centers is distributed file system traffic across the entire network. On average, about 30% of the traffic in these three EDU datacenters is within the same rack. The applications in the PRV datacenters have shown a degree of emerging patterns of consolidation and virtualization and around 45% of the traffic is within the same rack. The MapReduce job in the CLD Datacenters is scheduled to be packed into the same rack to reduce core interconnection and nearly 75% of the traffic is confined in the same rack.

For our topology study, we need traffic traces that not only follow different patterns in the EDU, PRV and CLD Datacenters but also have different loads. Therefore, we created a traffic generator to generate traffic traces following the traffic patterns of these Datacenters with varies loadings and fed them to the fat-tree simulator we implemented. Traffic in a fat-tree can be characterized by two probabilities: p_1 and p_2. p_1 denotes the probability that the source and destination of a packet are connected to the same edge switch. Of the other packets, there is a probability p_2 that their destination is within the same pod and thus will need to traverse an aggregation switch. The rest of the packets are destinated to servers in other pods and thus need to pass through a core switch. By varying the probabilities p_1 and p_2, we can simulate different types of traffic patterns. Based on the results of Benson *et al* [20]), we generate synthetic traffic traces using an On/Off process with the On and Off periods following the lognormal distributions ($\sigma_{off} = [2.6, 3], \mu_{off} = [12.4, 13.8], \sigma_{on} = [2.6, 2.9], \mu_{on} = [12.2, 14.1]$). The packet interarrival time is also a lognormal process ($\sigma = [3.8, 4.1], \mu = [20.1, 22.1]$). The source and destination nodes are chosen *uniformly* from servers in each pod. The specific parameters we used are: EDU($p_1 = 0.3$, $p_2 = 0.2$), EDU1 ($p_1 = 0.3$, $p_2 = 0.5$), PRV ($p_1 = 0.45$, $p_2 = 0.2$) and CLD ($p_1 = 0.75$, $p_2 = 0.2$).

In a second study, we examine the benefits of consolidating jobs into fewer servers by distributing different traffic loads to each pod, which is called the *non-uniform* case. For example, say we have 12 pods. We generate 70% pod load for the first five pods and 10% pod load for the sixth pod and leave all other pods with zero traffic with the overall load be 10% for the entire network. For each of the traffic patterns we generate loads from 10% to 70% of network capacity.

We simulate a $k = 12$ fat-tree DCN that supports 432 end hosts connected through 180 *12-port* switches. These switches are grouped in 12 pods and each pod contains 6 edge switches and 6 aggregation switches. The core layer consists of 36 core switches, which connect the 12 pods together.

5.1.2 Active Sub-Trees

We feed a 10-second synthetic packet trace to the simulated fat-tree and use left-most routing. We demonstrate the sub-trees for all traffic patterns with load of 10% in Figure 5.1. It shows that a minimum spanning tree is sufficient for CLD Datacenters because only around 25% of their traffic leaves the rack. We observe that even the load increases to 70%, there are still a significant number of switches and links in idle state. If we can pack the communicating jobs into fewer number of pods like in the non-uniform scenario, then the number of edge switches and aggregation switches required will be further reduced.

The fraction of active switches is shown in Figure 5.2. It shows that when the traffic load is less than 70%, 20% of switches are never used. For light traffic at 10%, less than 50% of switches are needed. Since the CLD Datacenters are usually designed with meticulous job placement in order to decrease cross-pod traffic, the fraction of unused switches for CLD is the greatest. For non-uniform traffic, fewer edge switches and aggregation switches are required since jobs are consolidated into fewer pods. However, the non-uniform case requires more core switches than uniform cases because the number of active core switches is dependent on the pod with the heaviest traffic going to the core layer. So when we pack jobs into fewer servers, it is better to balance the load among the active pods. For example, the 10% CLD nonuniform traffic load is split into 70% of pod traffic in pod 1 and 50% pod load in pod 2. The maximum core number required (= 12) is determined by pod 1 since more traffic from pod 1 is going to core layer. If we split the 10% overall load into 60% in pod 1 and 60% in pod 2, the maximum number of core switches required is reduced to 6.

5.1.3 Analytical Model of Sub-Tree Size

The simulations above clearly show that significant part of a fat-tree can be dispensed without affecting performance. However, the simulation results were conducted for relatively small DCNs. In this section, we provide a theoretical model that can be generalized to arbitrary sized DCNs. Let us assume that traffic load generated by k pods is $\lambda_1, \lambda_2, ..., \lambda_k$ (represented as a fraction of full load). We use parameters p_1 and p_2 to represent the probability of traffic travelling between servers connected to the same edge switch, and traffic travelling within the same pod but different subnets, respectively. Therefore, $(1 - p_1 - p_2)$ of traffic goes to other pods. We assume that p_1 and p_2 for each of the k pods are written as $p_1^1, p_1^2, ..., p_1^k$ and $p_2^1, p_2^2, ..., p_2^k$. For pod i, λ_i of traffic load is generated of which $\lambda_i(1-p_1^i)$ goes up to the aggregation layer and $\lambda_i(1-p_1^i-p_2^i)$ arrives the core layer switches.

The edge switches are constantly active since they are connected to servers.

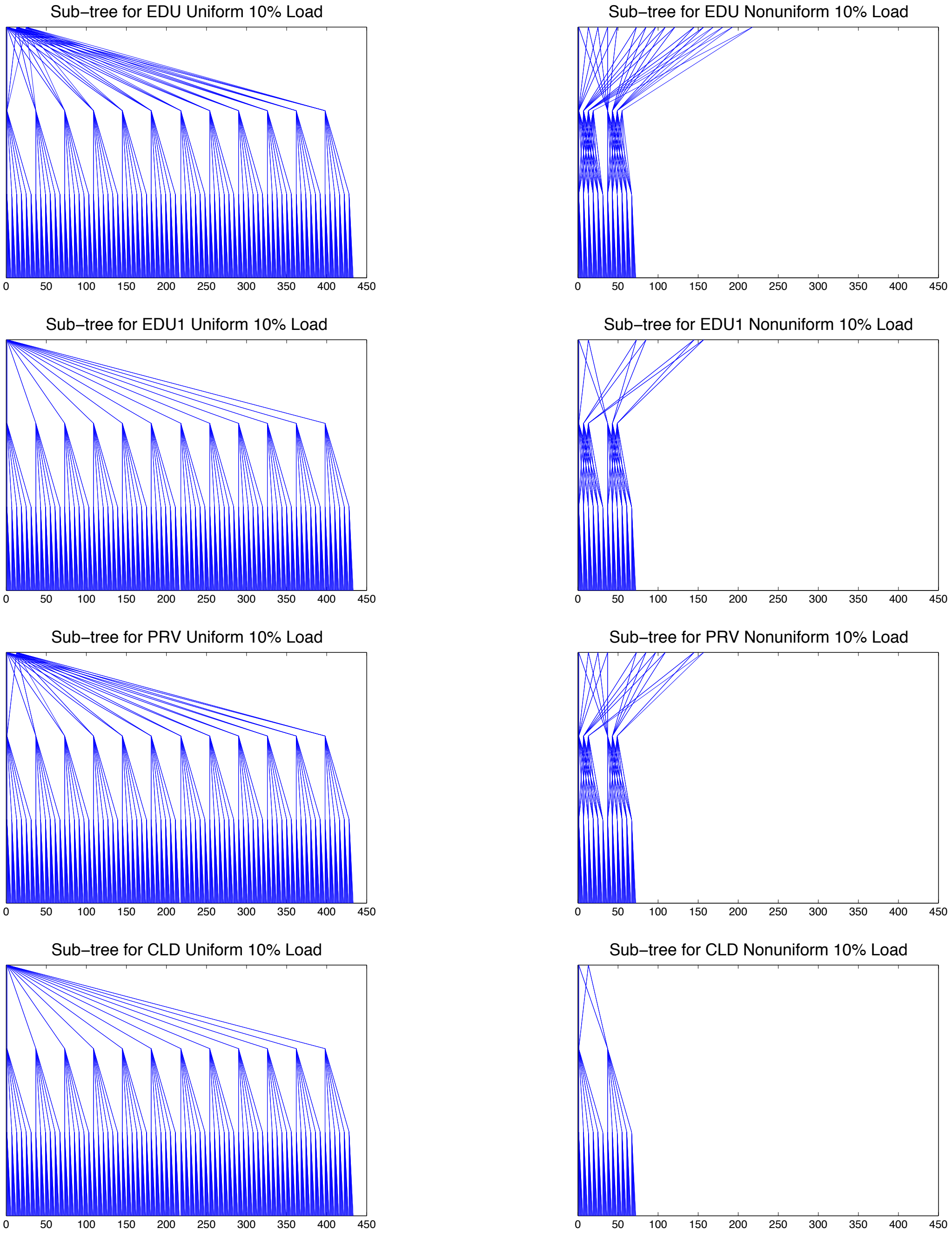


Figure 5.1. Minimal *fat-trees* with uniform and non-uniform traffic of load 10%.

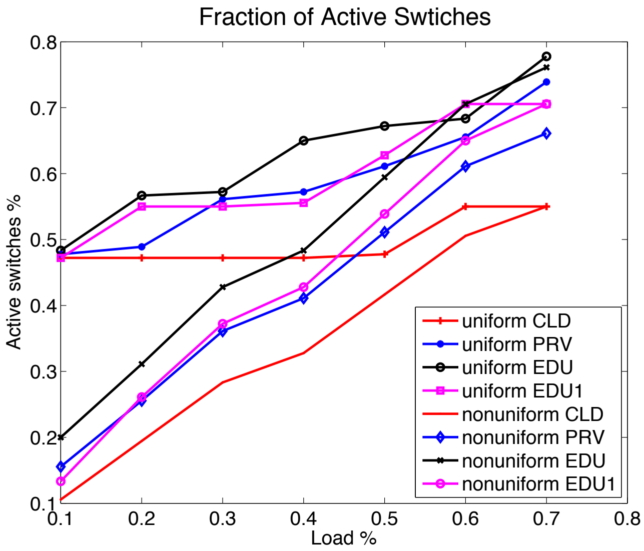


Figure 5.2. Fraction of switches required for uniform and non-uniform traffic.

Thus for each pod i, the number of edge switches that are powered on is $e_i = k/2$. Since the traffic from the edge switches takes the left-most available aggregation switches first, and the total capacity of each aggregation switch is $\frac{1}{\frac{k}{2}}$ of the pod load, the total number of aggregation switches in pod i that handle traffic load $\lambda_i(1 - p_1^i)$ is:

$$a_i = \lceil \frac{\lambda_i(1 - p_1^i)k}{2} \rceil \tag{5.1}$$

For the core layer, we consider two scenarios. The first scenario is when $p_2 = 0$, all the traffic arriving in the aggregation switches is going up to the core layer. The number of core switches is determined by the maximum load of the k pods. Suppose pod j has the maximum load, $\lambda_j(1 - p_1^j)$, going to core layer. Since each core switch can handle a fraction of $\frac{1}{\frac{k^2}{4}}$ pod load, the total number of core switches for the entire network is computed as:

$$c = \lceil \frac{\lambda_j(1 - p_1^j)k^2}{4} \rceil \tag{5.2}$$

When $p_2 \neq 0$, the number of core switches varies with the traffic load going to core layer. We can compute the range of the number of core switches needed. If the traffic going to core layer is distributed on the left-most aggregation switches for all the pods, then minimum number of core switches is needed and is calculated from the maximum core load of the k pods. Similarly, we suppose the maximum load going to the core layer is from pod j with the load $\lambda_j(1 - p_1^j - p_2^j)$ and the number of core switches is computed as:

$$c_{min} = \lceil \frac{\lambda_j(1 - p_1^j - p_2^j)k^2}{4} \rceil \tag{5.3}$$

If the load going to core layer is distributed randomly on active aggregation switches of each pod, the maximum number of active core switches is dependent on the maximum number of active aggregation switches. For example, suppose pod j has the most active aggregation switches and the number of active aggregation switches is $a_j = \lceil \frac{\lambda_j(1-p_1^j)k}{2} \rceil$. Each of these aggregation switches can send traffic to $k/2$ core switches connected to it. Therefore, we can estimate that the upper bound of the number of core switches as:

$$c_{max} = \lceil \frac{\lambda_j(1 - p_1^j)k}{2} \rceil \times \frac{k}{2} \tag{5.4}$$

Using above formulations, we can compute the number of active switches in each layer of a fat-tree network. The significant conclusions we can draw are as follows:

- Even at 70% loading, in both the uniform and non-uniform traffic cases, no more than 50% of aggregation switches are used.

- For CLD Datacenters, only a *third* of the aggregation switches are used because of the job placement policies.

- At 70% loading, no more than 50% of core switches are used while for CLD this percentage is even smaller at 33%.

Based on above results, we can reduce the number of aggregation switches and core switches by 50% from current fat-tree networks. One approach may be to keep each pod symmetric (for the uniform traffic model) and discard the rightmost 50% of aggregation switches from each pod. For the non-uniform traffic case, the leftmost pods would not be modified but the rightmost pods would have only a few aggregation switches as computed in eqn. 5.1. Similarly, we can discard rightmost core switches based on eqn. 5.3 and 5.4.

5.2 RIGHT SIZING THE EDGE SWITCHES

A regular fat-tree uses switches of the same size over the entire network. While this is a useful feature when purchasing switches in bulk from OEMs, we show that it is not the best approach from an energy efficiency standpoint. Consider the benefits of increasing the degree of edge switches, the immediate impact is that it increases p_1 and decreases p_2, thus we need fewer aggregation switches. The second benefit comes about in energy cost of the edge switches. The energy cost of a switch can be viewed as the cost of the chassis, switching fabric, linecards and ports [14]. As we show in Section 5.3.1, we can increase the port density of switches by adding new linecards which has the net effect of scaling the energy cost sub-linearly with the number of ports. Thus using a single switch with twice as many ports is more energy efficient as compared to using two switches with half as many ports each.

As we analyze in Section 5.1, the number of switches and links required in any pod for a given traffic is dependent on the traffic load λ, and traffic pattern parameters of p_1 and p_2. If we increase the size of edge switches, more servers will directly connect to it and, by definition, p_1 will be greater, and thus more traffic is transferred directly through the edge switches. As a result, the number of required aggregation layer switches will decrease to

$$a'_i = \lceil \frac{\lambda_i(1 - p_1^{i'})k}{2} \rceil \tag{5.5}$$

If we keep the size of the pod unchanged, then fewer edge switches is required when we replace small-sized edge switches with larger-sized switches. At the same time, the amount of inter-pod traffic, $1 - p_1 - p_2$, remains the same. Therefore, the lower bound of required core switches is still c_{min}. However, since a smaller number of aggregation switches are used, the inter-pod traffic is moved to the left side of the aggregation switches, so the maximum number of required core switches will decrease to

$$c'_{max} = \lceil \frac{\lambda_j(1 - p_1^{j'})k}{2} \rceil \times \frac{k}{2} \tag{5.6}$$

In particular, when p_1 keeps increasing and p_2 decreases to zero, all the traffic reaching aggregation layer is directed to the core switches. Under this circumstances, all traffic in core layer are consolidated to the leftmost core switches.

We simulate fat-trees using different sizes edge switches. The original $k = 12$ fat-tree has 6 *12-port* edge switches in each pod. In our experiment, we use 3 *24-port* switches, or 2 *36-port* switches, or 1 *72-port* switch in the edge layer of each pod. For all cases, edge switches still use half of the ports connected to servers and the other half connected to aggregation switches. For example, the *24-port* edge switch connects to 12 servers and connects to 6 aggregation switches with *two links* connecting each edge switch and aggregation switch. We note that changing the radix of edge switches will not change the total traffic between servers and edge switches or between aggregation and core layers. The only change is in traffic between the edge and aggregation layers, as shown in Figure 5.3. Indeed, as the radix of edge switches increases, less traffic goes from the edge switches to the aggregation switches.

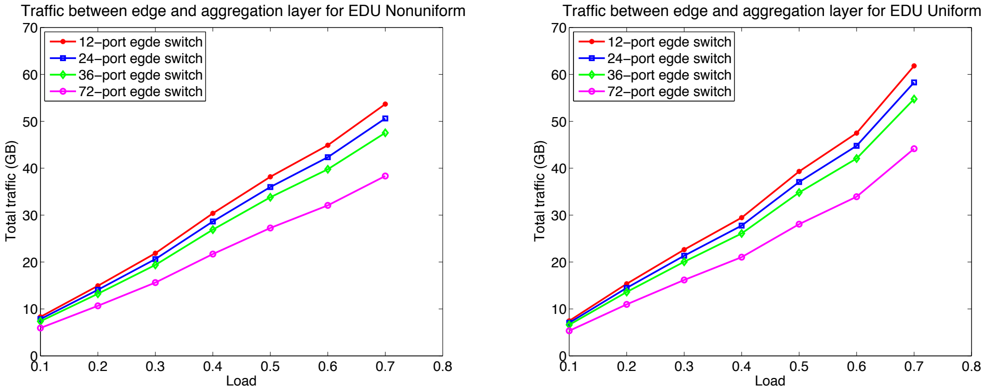


Figure 5.3. Traffic between edge layer and aggregation layer is less when the size of edge switches increases. (Figures shown above are for uniform and nonuniform traffic patterns in EDU Datacenters. CLD, PRV and EDU1 Datacenters also have the same properties.)

The fraction of active switches is illustrated in Figure 5.4. We can conclude that as the size of edge switches increases, the fraction of the total number of required switches decreases. The reduction in aggregation switches needed comes about since p_2 decreases (less intra-pod traffic). Although the inter-pod traffic remain unchanged, the fraction of core switches may reduce slightly because the traffic going to the core layer can be further consolidated to the left core switches. Figure 5.5 shows examples of the resulting sub-trees when we use either *12-port* or *72-port* edge switches for an EDU Datacenter in the $k = 12$ fat-tree. We can conclude that *using the highest port-density switches for the edge layer minimizes the overall number of aggregation and core layer switches required.*

5.3 ENERGY SAVINGS OF LARGER-SIZED EDGE SWITCHES

Let us next consider the energy benefits of using high port density switches at the edge. Datacenter switches are chassis-based modular switches designed for reliability and performance. The number of ports can be expanded by inserting more

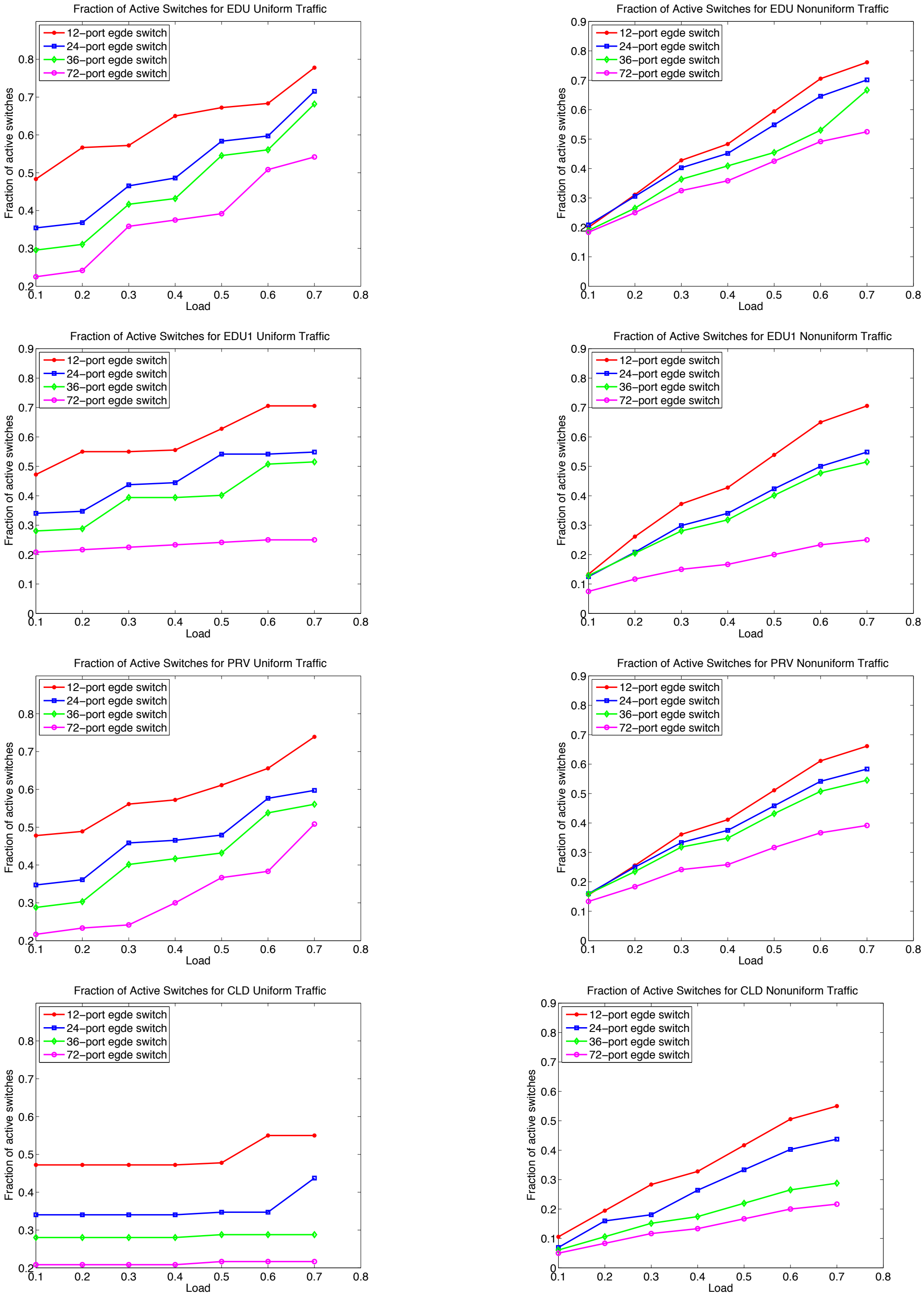


Figure 5.4. Fraction of active switches with larger-sized edge switches.

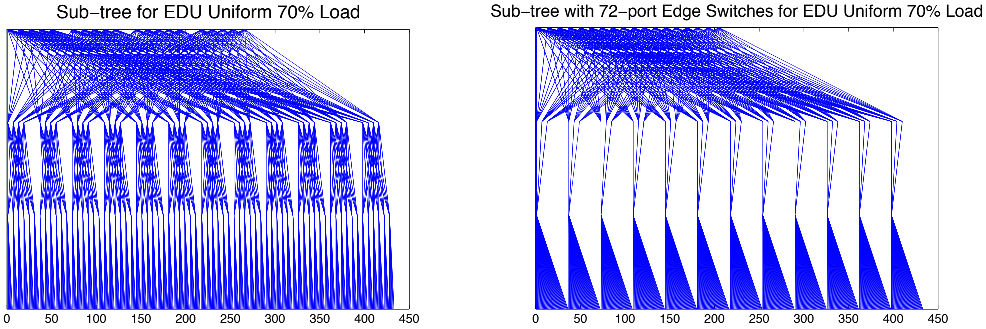


Figure 5.5. EDU DCNs with *12-port* and *72-port* edge switches, 70% load.

linecards in the switch chassis. In this section, we formulate the power consumption model of the DCN and use actual power consumption data from different Cisco switches to make the case for utilizing high port density edge switches.

5.3.1 Static Cost

Consider a fat-tree network constructed with *12-port* switches supporting 432 end hosts. We compare its energy consumption with the cases using edge switches of 24, 36 and 72 ports. Cisco Catalyst 4503-E is a modular datacenter switch with two linecard slots. From Cisco Power Calculator [2], we find the power consumption data of Catalyst 4503-E switches (shown in Table 5.1). The fixed power cost of a switch includes the power cost by the chassis, supervisor engines and linecards. The Catalyst-4503-E switch works with three supervisor engine models: 6LE,7E/7LE and 8E. Model 6LE supports 46XX series 1GE *12-port* and *24-port* linecards and model 7E/7LE/8E supports 47XX series linecards up to 48 ports. Combining different choices of linecards, we can create switches with port number from 12 to 72. We calculate the total power consumption of a $k = 12$ fat-tree DCN when using different sizes of edge switches. The results is shown in Figure 5.6 (left) and we see around 30% power savings for the entire network when replacing *12-port*

Table 5.1. Power Consumption of Datacenter Modular Switch - Cisco Catalyst 4503-E.

Component	Model	Power Cost
chassis		48W
Supervisor Engine	6LE	168W
	7E/7LE	223.68W
	8E	319.97W
Linecard	4612-SFP-E	24W
	4624-SFP-E	40.32W
	4712-SFP-E	19.97W
	4724-SFP-E	31.97W
	4748-SFP-E	73.63W

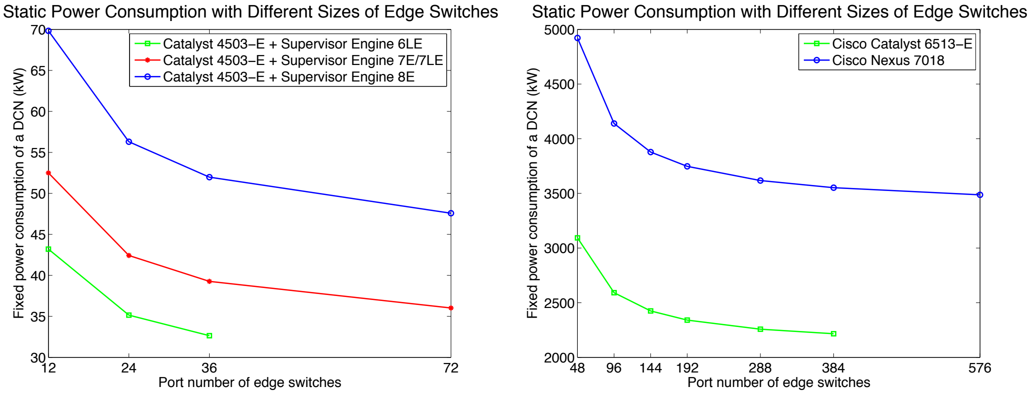


Figure 5.6. Static power consumption of a $k = 12$ and a $k = 48$ *fat-tree* DCN with different sizes of edge switches.

switches with *72-port* switches in the edge.

Large cloud Datacenters have tens of thousands of servers and require high-port-density switches. For example, we need a *192-port* switches to merge 4 *48-port* edge switches together. A Catalyst 6513-E switch chassis has 11 linecard slots and supports 528 1GE ports in total. A Cisco Nexus 7018 switch has 16 linecard slots, providing 768 1GE ports. We choose switch configurations that consume the least power per port from the two switches and calculate the DCN power consumption shown in Figure 5.6 (right).

5.3.2 Dynamic Cost

The power consumption of chassis, supervisor engine and linecards is fixed when the switch is deployed. However, a port consumes more power when it is active. Besides, port capacity setting, port utilization and switch firmware version also affect the power consumption of a switch [64]. For simplicity, We only consider the static power consumption and the power cost by ports in this work and we represent the power model of a switch as:

$$\begin{aligned} P_{switch} = {} & P_{chassis} + P_{supervisor-engine} \\ & + numCard \times P_{linecard} \\ & + numActPort \times P_{actPort} \\ & + numIdlePort \times P_{idlePort} \end{aligned}$$

where $numCard$ is the number of linecards supported by the switch. $numActPort$ is the number of active ports and $numIdlePort$ is the number of inactive port. We compute the overall DCN power consumption as the sum of power cost of all switches:

$$P_{total} = \sum P_{switch}$$

Using the data in Table 5.1, the switch 4503-E chassis with 12, 24, 36 and 72 ports has a fixed cost of at least 240W, 264W, 280.32W and 377.28W, respectively. We also learn from [14] that each port consumes 3W when active and 0.1W when idle. Thus we can formulate the power model for estimating the switch power

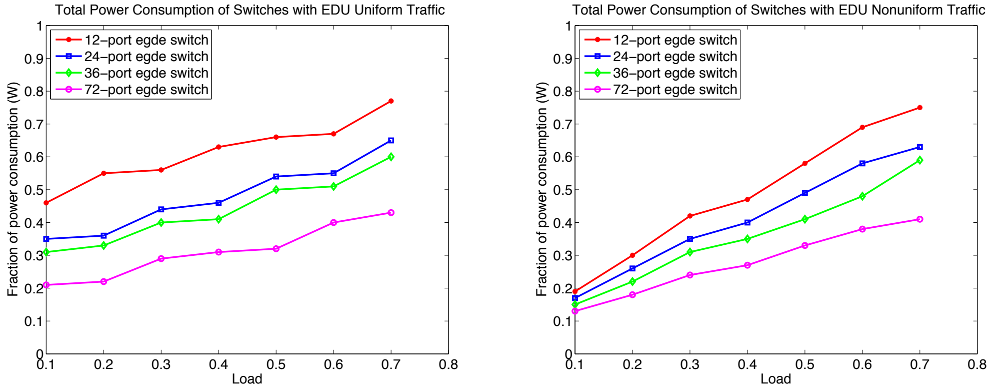


Figure 5.7. Fraction of total power consumption of network switches with larger-sized edge switches for different traffic load and patterns in EDU Datacenters.

consumption as:

$$P_{switch} = \begin{cases} 240 + (12 - x) * 0.1 + 3x & \textit{12-port} \text{ switch} \\ 264 + (24 - x) * 0.1 + 3x & \textit{24-port} \text{ switch} \\ 280.32 + (36 - x) * 0.1 + 3x & \textit{36-port} \text{ switch} \\ 377.28 + (72 - x) * 0.1 + 3x & \textit{72-port} \text{ switch} \end{cases}$$

where x is the number of active ports. The resulting power consumption of the subtrees of EDU Datacenters is compared in Figure 5.7. With the uniform traffic load, the homogeneous fat-tree can achieve more than 50% power savings through leftmost routing. By replacing the *12-port* edge switches with larger-sized switches, traffic flows can be further consolidated at the edge and core layer, and thus achieving a skinner sub-tree with more energy savings. *Thus we can conclude that using higher port density edge switches saves energy by requiring fewer aggregation switches and by reducing the energy needed by the edge switches themselves.*

5.4 SUMMARY

Many approaches are studied to find the minimum subset of DCN topology for an offered traffic load without changing network interconnection or the network devices. Recently, Widiaja *et al.* [82] compare the energy savings of deploying different sizes of switches in a fat-tree network. They find it is more energy efficient to use smaller-sized switches when the traffic is highly localized. Chabarek *et al.* [24] propose to build energy proportional DCN using low-power low-radix switches for matched traffic patterns.

This chapter explores the approaches to find the minimum network topology for a given loads and traffic patterns. We derive a traffic-driven model to calculate the number of switches required for each layer in a fat-tree DCN. We use the left-most heuristic flow assignment algorithm to simulate the traffic consolidation process and validate the model correctness. Based on the model, we propose to use high-radix edge switches when the traffic load within the same pod is high, which significantly reduces the number of switches in aggregation layer. Furthermore, using high-radix edge switches can affect the routing of inter-pod traffic and consolidate it to the left-side core switches. As a result, fewer core switches are used and the active core switches are aligned to the left of the core layer, resulting in a smaller sub-tree. Using this principle, datacenter operators can easily determine which core switches can be powered off. We survey the power consumption of commodity modular switches and conduct an evaluation of the power savings when using larger-sized edge switches in different types of datacenters. We find that the overall power consumption is reduced by using high-radix edge switches in fat-tree DCNs.

Chapter 6

TRAFFIC CONSOLIDATION USING MERGE NETWORKS

In previous chapters, we investigated the usage-based network optimization. We also formulated analytical models and proposed heuristic routing algorithm to find optimal power consumption of datacenter networks. However, in practice, a large number of switches still need to remain active even for very light loadings, resulting in sub-optimal energy savings.

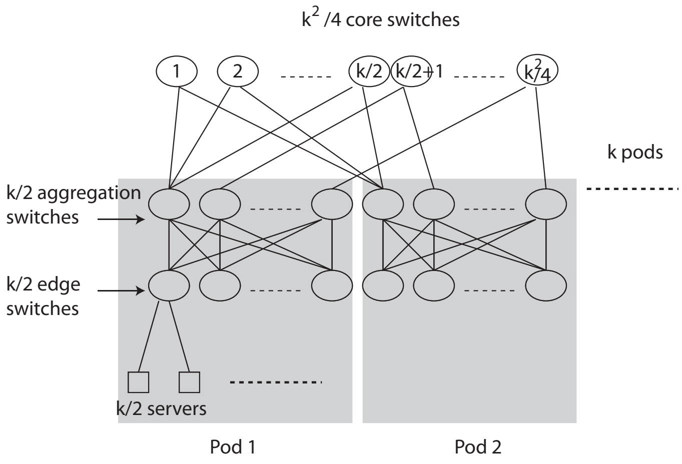


Figure 6.1. *Fat-tree* model.

We use a k-ary *fat-tree* shown in Figure 6.1 as an example to illustrate the situation and explain the motivation for our work. The k-ary *fat-tree* interconnects $k^3/4$ servers using three layers of switches. The edge-layer switches and aggregation-layer switches are organized in k "pods". Each pod includes $k/2$ aggregation switches and $k/2$ edge switches and each edge switch connects to $k/2$

servers. There are $k^2/4$ core switches and each core switch has k links connecting to k pods.

In datacenters, all the edge switches are always powered on as they are connected to servers and have to remain active to be able to forward traffic upward and downward at all times. Even if a server has very little traffic going to the connecting edge switch, the switch will be fully powered on although very lightly loaded. *The contribution of our approach is to enable powering off more switches and links by consolidating traffic at the edge and aggregation layer.*

6.1 OUR APPROACH: MERGING

Consider the case of an edge switch connected to $k/2$ servers. Assuming each server offers a load of λ (expressed as a fraction of link rate). The total traffic to this switch from the servers is $k\lambda/2$. Normally the $k/2$ switch interfaces remain active even when λ is small. If the traffic can be consolidated, we need at most $\lceil k\lambda/2 \rceil$ interfaces to be active. When the load λ is small, more switch interfaces of the switch can be powered off. In other words, if there was a way to *merge* the traffic from the $k/2$ servers, we could potentially power off more switch interfaces and save power. Previous papers [75][85] provided a hardware design of a device called *merge network*. Rather than repeating that discussion here, we provide a *functional* model of what such a network does, and then use it in the remainder of this chapter.

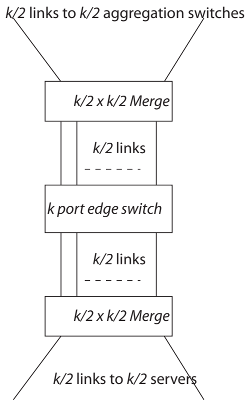


Figure 6.2. Merge networks applied to a switch.

We illustrate a $\frac{k}{2} \times \frac{k}{2}$ merge network in Figure 6.2. The merge network has $k/2$ connections to the $k/2$ servers and $k/2$ connections to the $k/2$ switch interfaces. A merge network has the property that it pushes all traffic from servers to the *leftmost interface* of the switch. If that interface is busy, then the traffic is forced to the next interface, and so on. This traffic merging behavior ensures that several switch interfaces can be put to low power mode without compromising connectivity. Of course, because of the fact that we are breaking the 1-1 association of a switch interface to a server interface, several layer 2 protocols will break. Some additional observations about the merge network are as follows:

1. The merge network is a fully analog device with no transceivers and, as a result, its power consumption is below one watt. The merge network can be visualized as a train switching station where trains are re-routed by switching the tracks (rather than store-and-forward).

2. Consider the uplink from the servers to the merge network. All traffic coming into the merge network is output on the *leftmost* $m \leq k/2$ links connected to the m leftmost interfaces of the switch, where $m = \lceil k\lambda/2 \rceil$ (assuming a normalized unit capacity for links). This is accomplished internally by sensing packets on links and automatically redirecting them to the leftmost output from the merge network that is free.

3. On the downlink to the servers, traffic from the switch to the $k/2$ servers is sent out along the leftmost $m \leq k/2$ switch interfaces to the merge network. The packets are then sent out along the $k/2$ links attached to the servers from the output of the merge network. The manner in which this is accomplished is described in [75] (note that the challenge is to correctly route the packets flowing through the merge network to the appropriate destinations).

We apply two $\frac{k}{2} \times \frac{k}{2}$ merge networks to each edge switch as shown in Figure 6.2. The connections are similar for each aggregation switch. For the core switches, we connect a $k \times k$ merge network.

Alternatively, we can connect a merge network to multiple switches. In this scenario, traffic is merged to interfaces of the leftmost switches, and the right-end switches with all idle interfaces can be powered off, achieving more energy savings. In this work, we apply merge networks at two locations within a pod of a fat-tree network – one location is between the servers and the edge switches and the other location is between the edge switches and the aggregation switches. Figure 6.3 shows a single pod of a *fat-tree* after applying merge networks. As shown, we utilize one $k^2/4 \times k^2/4$ merge network to connect all the servers in a pod to the interfaces of edge switches and apply another merge network to connect edge switches to aggregation switches.

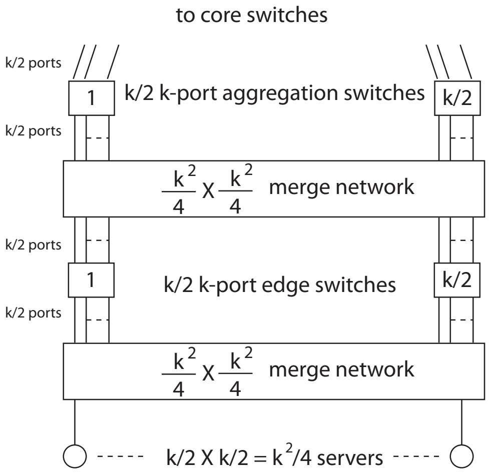

to core switches
k/2 ports
1
k/2 k-port aggregation switches
k/2
k/2 ports
$\frac{k^2}{4}$ X $\frac{k^2}{4}$ merge network
k/2 ports
1
k/2 k-port edge switches
k/2
k/2 ports
$\frac{k^2}{4}$ X $\frac{k^2}{4}$ merge network
k/2 X k/2 = k^2/4 servers

Figure 6.3. Merge network applied to pod in a *fat-tree*.

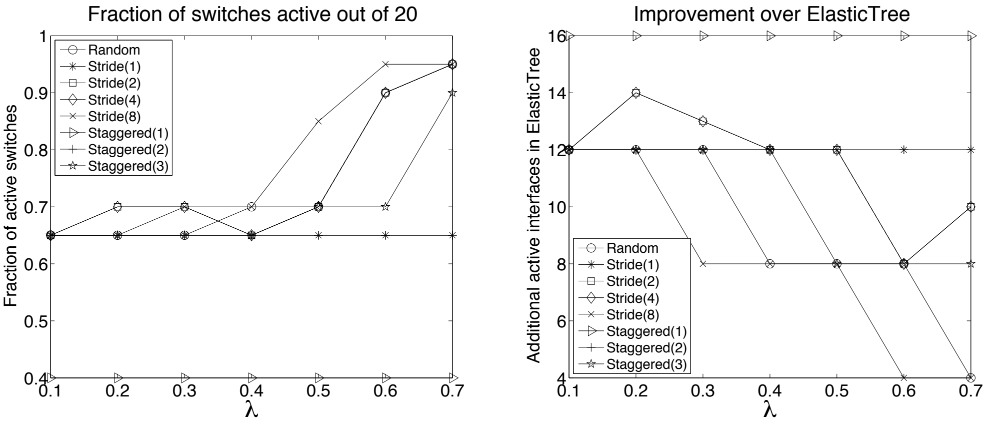
To apply merge networks to a *fat-tree* network, we add two $\frac{k}{2} \times \frac{k}{2}$ merge networks to each edge switch as shown in Figure 6.2. The connections are similar for each aggregation switch. For the core switches, we connect a $k \times k$ merge network.

6.2 ENERGY SAVINGS DUE TO TRAFFIC MERGING

To illustrate the additional energy savings achieved by merge networks when compared with approaches such as ElasticTree, we quantify this benefit by running the optimization problem on several different types of network loadings for a small *fat-tree* topology of size $k = 4$. In this topology, there are 8 edge switches, 8 aggregation switches and 4 core switches. For each edge switch, there are 2 servers connected for a total of 16 servers in 4 pods. We assume that there is a 2×2 merge network connected to either side of each edge and aggregation switch and there is a 4×4 merge network connected to each core switch.

6.3 TRAFFIC PATTERNS

Traffic patterns in data centers can vary greatly, and to ensure our results are widely applicable, we run the optimization algorithm on the following types of traffic: *Random*, $Stride(n)$, $Staggered(n)$ [11]. In *Random*, the source and destination are randomly selected from among the servers. For $Stride(n)$, the destination of a flow from server i is server $[(i + n) \mod 16]$, where servers are numbered left to right as $0, 1, \cdots, 15$. For example, in a $k = 4$ *fat-tree* network, $Stride(1)$ has almost half of the traffic goes between servers connected to the same edge switch and the other half traffic goes to aggregation and core switches. On the other hand, $Stride(4)$ sends all traffic between pods, resulting in a larger number of switches to participate in forwarding traffic. The *Staggered* traffic model assigns a probability p_1 for traffic going to a server in the same subnet (i.e., connected to the same edge switch), a probability p_2 for traffic going to a server in the same pod but different subnet, and a probability $1 - p_1 - p_2$ where the flow is destined to a server in a different pod. By varying these probabilities, we can generate a large number of



(a) Number of active switches

(b) Total number of active interfaces

Figure 6.4. Difference in number of active switches and active interfaces network-wide.

different loading patterns.

6.4 TRAFFIC MERGING WITHIN A SWITCH

6.4.1 Number of Active Interfaces

Figure 6.4a plots the percentage of active switches for our approach as well as for ElasticTree for different loading patterns and different loads. As we have expected, the number of active switches for $Stride(1)$ does not vary with λ. This is because almost all the traffic goes to the server in the same subnet or in the same pod and therefore, the active switches required are always the eight edge switches, one aggregation switch per pod and one core switch. $Stride(8)$ shows the highest number of active switches because all the traffic is inter-pod traffic and hence more core switches are used.

In order to illustrate the potential benefits of traffic merging, we take a difference between the total number of active interfaces when using ElasticTree and using traffic merging with the above optimization. The results, shown in Figure

6.4b, clearly illustrate the benefits of merging. In the case of $Stride(1)$, ElasticTree uses 12 more interfaces than merging. The reason is that one aggregation switch is active per pod. In ElasticTree, all the four interfaces to this switch are active (albeit with very low traffic). In our approach, in contrast, we merge the traffic using a merge network and use only a single interface of the switch.

6.4.2 Energy Savings

The overall energy cost of a switch can be roughly partitioned into the cost of the chassis and the cost of the interfaces. As described in [82, 34], a reasonable approximation to the cost of a switch is

$$\text{Switch Cost } = C + m \log m + m$$

where m is the number of active switch ports. The constant C accounts for static costs of a switch such as fan, etc. The second term corresponds to the cost of the interconnection fabric within the switch, which is a significant contributor to energy consumption (typically 30% $\sim$ 40%). This cost scales as $m \log m$ for a switch with m active ports. The last term is the cost contribution from the active interfaces. This term folds into itself the cost of the line cards that the interfaces are on. For the purpose of comparing the *overall cost reduction* of traffic merging relative to ElasticTree, we set C to 50% of the maximum switch cost and express it as

$$C = m_{\max} \log m_{\max} + m_{\max}$$

where $m_{\max}$ is the number of switch ports. If the traffic load fraction going to a switch is λ, the merge network will switch the traffic to the leftmost $k = \lceil \lambda m \rceil$ ports. Thus, the cost of a switch with merge networks is written as

$$\text{Traffic Merging Switch Cost } = C + k \log k + k$$

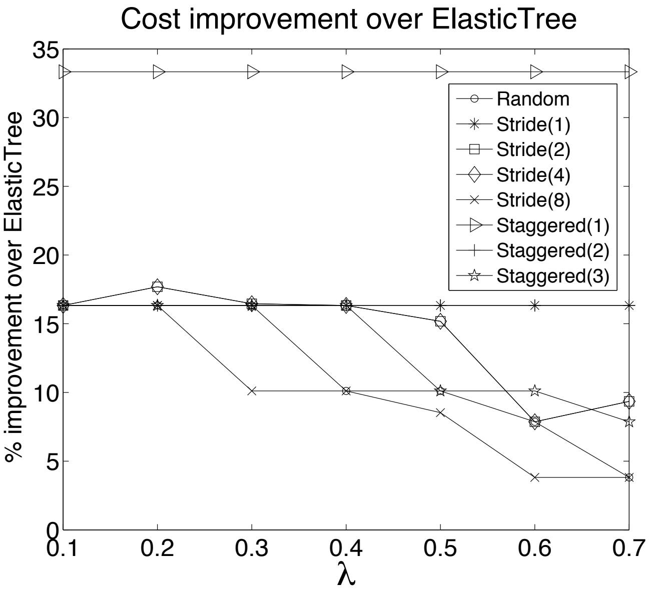


Figure 6.5. Reduction in total cost when using traffic merging.

Therefore, the fraction of cost savings of traffic merging over ElasticTree is calculated as

$$\text{Cost Savings} = \frac{m\log m - k\log k + m - k}{C + m\log m + m}$$

Figure 6.5 plots the fraction of reduction of network cost using traffic merging over ElasticTree. It is noteworthy that, for all traffic patterns and across all loads, the traffic merging reduces the overall energy cost even for a small-sized network consisting of 20 switches. These savings are more substantial when we consider realistic DCNs as we do later in this paper.

6.5 TRAFFIC MERGING WITHIN A POD

In this section, we compute the minimum number of active switches required when we apply merge networks to all switches at each layer within a datacenter pod. The merge networks force traffic to the left so that more aggregation and core switches can be powered off. Specially, traffic merging at the edge layer enables idle edge switches, which can be powered off.

6.5.1 Lower Bound on Energy Consumption

We derived the analytical expressions for minimal energy consumption of *fat-trees* for different type of loadings in Chapter 3. We assume *fraction of active switches* as the metric for energy consumption and assume an underlying routing protocols that pushes traffic to the left. Using parameters p_1, p_2 and p_3 to model traffic loading types (A packet from a server goes to another server connected to the same edge switch with probability p_1, which goes to a server in the same pod but another edge switch with probability p_2, and with probability $p_3 = 1 - p_1 - p_2$ it goes to a server in a different pod.) and λ to denote the average load offered by each server (λ as a fraction of link speed which we normalize to 1), the total traffic at the level of edge switches, pod aggregation switches and core switches is represented as follows:

$$\begin{aligned}
&\text{Traffic per edge switch} = \tfrac{\lambda k}{2} \\
&\text{Traffic for all aggregation switches in a pod} \\
&\qquad\qquad\qquad = \tfrac{(1-p_1)\lambda k^2}{4} \\
&\text{Traffic for all core switches} = k \times \tfrac{(1-p_1-p_2)\lambda k^2}{4}
\end{aligned}$$

Note that traffic flow is symmetric and the numbers above correspond to both, traffic into and out of a switch or switches.

We observe that *all the edge switches need to remain active at all levels of loads* to ensure servers have network connectivity. This gives us $k^2/2$ active edge switches. Within each pod we have total traffic equal to $(1 - p_1)\lambda k^2/4$ going into/from the $k/2$ aggregation switches from/to the edge switches. Given each link has a normalized capacity of 1, and that there are $k/2$ interfaces per aggregation switch connected to the edge switches, we require at least $\lceil\frac{(1-p_1)\lambda k^2/4}{k/2}\rceil$ active aggregation switches per pod. Since there are k pods, the total number of active aggregation switches becomes $k\lceil\frac{(1-p_1)\lambda k}{2}\rceil$. Finally, since every core switch is connected to an aggregation switch from each of the k pods, the number of active core switches we require is simply, $\lceil\frac{(1-p_1-p_2)\lambda k^3/4}{k}\rceil$ where we divide the total traffic passing through the core switches by the number of links per switch and round up. Therefore, the total number of active switches can be written as:

$$\text{Active Switches} \quad = \frac{k^2}{2} + k\left\lceil\frac{(1-p_1)\lambda k}{2}\right\rceil + \left\lceil\frac{(1-p_1-p_2)\lambda k^2}{4}\right\rceil$$

Figure 6.6 plots the fraction of active switches as a function of load λ for five different scenarios when $k = 12$. The plot with the labels -*o* corresponds to the case when all traffic is between the servers connected to an edge switch. In other words, no traffic needs to flow to the aggregation switches or to the core switches. As expected, the graph stays flat. However, *what is relevant here is that even at light loads of 0.1, all the edge switches are fully active.* At this load value, each server generates 1/10th of the uplink capacity of traffic (similarly for downlink) but the energy consumed is the same as when the link is fully loaded. The total combined traffic from all the 6 servers is 0.6, which is less than the capacity of a single link.

The plot with the labels $-\triangleleft$ corresponds to the extreme case when all the traffic is destined for servers in a different pod. Hence the core and aggregation switches will be utilized. Contrasting this with the case discussed above, we observe that *energy scales approximately linearly with load*, if we discount the edge switches. This is the desired behavior for energy-proportional networking.

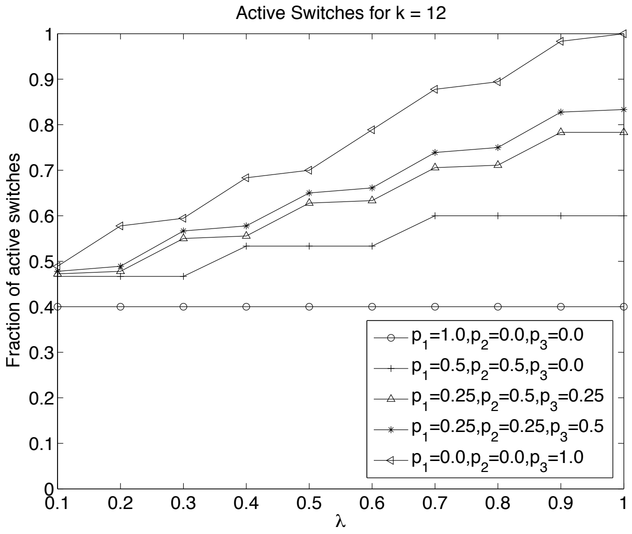


Figure 6.6. Active switches for the model in Section 3.1.1.

6.5.2 Energy Savings Due to Traffic Merging

To save energy used by the edge switches, we apply a $\frac{k^2}{4} \times \frac{k^2}{4}$ merge network between the $k^2/4$ servers and the $k/2$ edge switches. There are two consequences after applying the merge networks. First, the λp_2 traffic that goes to other subnets within the same pod has no necessity to go through the aggregation level. Instead, it is transferred directly to the destination servers. The traffic loading parameters are changed to p_1' and p_2' and $p_1' = p_1 + p_2$ and $p_2' = 0$. Accordingly, the number of active aggregation switches required in each pod is $\lceil \frac{(1-p_1')\lambda k}{2} \rceil = \lceil \frac{(1-p_1-p_2)\lambda k}{2} \rceil$. Second, traffic from servers is now sent to a merge network and consolidated to the leftmost edge switch and the idle edge switches can be put to low power mode to save energy. Therefore, The active edge switches in one pod can be calculated as $\lceil (\frac{\lambda k^2}{4})/(\frac{k}{2}) \rceil = \lceil \frac{\lambda k}{2} \rceil$, which changes with the traffic load λ. The total number of active switches after applying the merge networks can be written as:

$$\begin{aligned}\text{Active Switches} &= k\lceil \tfrac{\lambda k}{2} \rceil + k\lceil \tfrac{(1-p_1')\lambda k}{2} \rceil + \\ &\left\lceil \tfrac{(1-p_1'-p_2')\lambda k^2}{4} \right\rceil \\ &= k\lceil \tfrac{\lambda k}{2} \rceil + k\lceil \tfrac{(1-p_1-p_2)\lambda k}{2} \rceil + \\ &\left\lceil \tfrac{(1-p_1-p_2)\lambda k^2}{4} \right\rceil\end{aligned}$$

Figure 6.7 shows the fraction of active switches in a $k = 12$ *fat-tree* with merge networks for the same traffic models as Figure 6.6. It is very illustrative of the benefits of traffic merging. The number of active switches is reduced more when the traffic load is lower. It is noticeable that the flat line in Figure 6.6 becomes more linear with the load after the traffic merging.

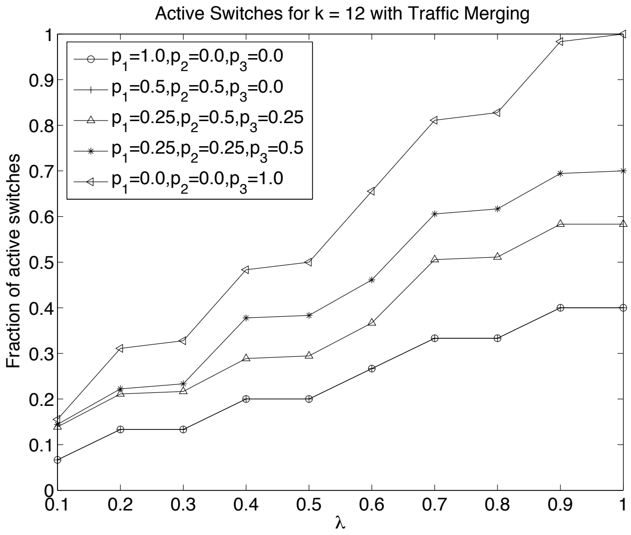


Figure 6.7. Active switches for the model with traffic merging.

6.6 SIMULATION RESULTS

We build a simulator for a *fat-tree* network with $k = 6$ and 1 Gbps link capacity. Each server generates traffic based on a two-state On/Off process in which the length of the On and Off periods follows a lognormal distribution. In the On state, packet inter-arrival times are also from a lognormal process [20]. The parameters selected for the lognormal processes are based on different types of traffic patterns as well as different loading patterns. For each packet, the destination is selected uniformly randomly from the set of all nodes based on probabilities p_1 and p_2. In the simulator, we read these trace files which are generated externally and forward packets based on routing tables computed *every second of simulated time.* In our simulation, we test several sets of traffic loads with different p_1 and p_2 and obtain similar results. Due to space limitations, we only publish the results from the first set of traffic with p_1 and p_2 as follows:

1. $p_1 = 0.75, p_2 = 0.125$;
2. $p_1 = 0, p_2 = 0.75$;
3. $p_1 = 0, p_2 = 0$;

We assume that external traffic q is 10% in all three cases. The total traffic load is from $10\% - 70\%$ of the full bandwidth.

The routing algorithm is a modified version of Dijkstra's algorithm where we force flows to use routes that are already in use, thus packing flows together. In the algorithm we assign weights to edges as well as nodes. Edge weights are constant of 2, but node weights can be 0 or 1. If a node has been used for forwarding a flow, its weight changes from 1 to 0. Thus, flows are encouraged to reuse the same subset of nodes (or switches). We eliminate link with zero available capacity from further consideration in that round of routing computation.

We use $C = 1$ and designate the leftmost core switch as the externally connected core switch. For 10% external traffic, the link capacity l of the externally connected core switch has to be greater than 4. Therefore, we use $l = 4$ and let $Q = 2kl = 48$ to avoid traffic loss.

In Figure 6.8, we plot the fraction of active switches versus total load using simulation without merge networks and with merge networks. Figure 6.9 shows the same metric from the analytical models described in Chapter 3 and Chapter 6. It is easy to see that, our model is a very close match to the simulations. The minor difference between the simulations and model is due to the fact that we estimated the values of p_1 and p_2 from the synthetic traffic and then used them in the analysis. The estimated values for these probabilities are listed in the legend of Figure 6.9. *The implication of this is that the lower bound of energy efficiency can be achieved in practice by utilizing the simple routing algorithm described above.*

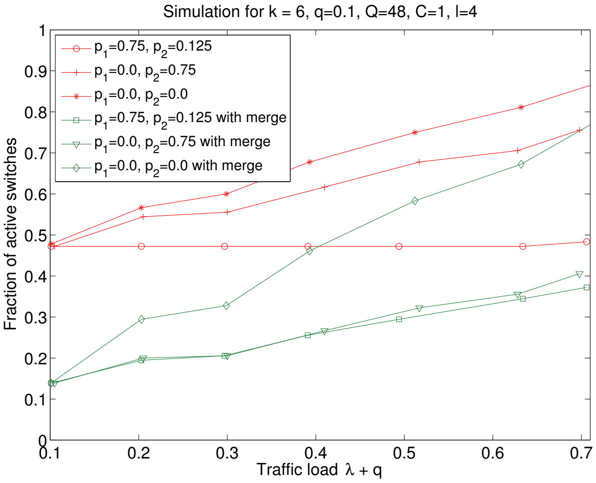


Figure 6.8. Simulation results of active switches for near and far traffic.

When we examine Figure 6.8, we observe that the type of loading has a significant impact on energy consumption. When it comes to allocating tasks to servers, the task manager should be mindful of the type of traffic that will be generated since we can obtain significant energy savings by careful scheduling.

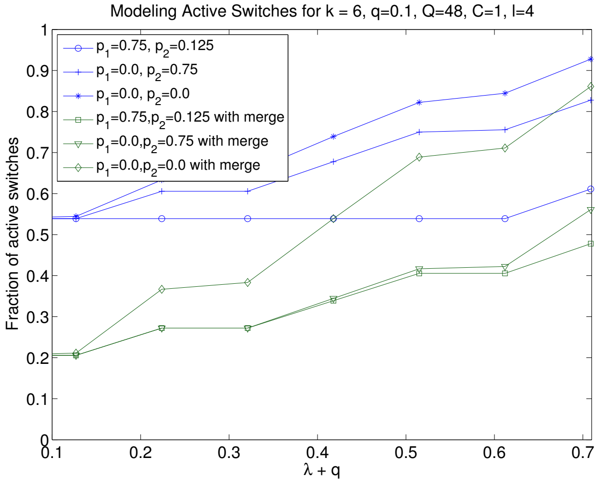


Figure 6.9. Modeling active switches for near and far traffic.

6.7 SUMMARY

Many approaches proposed on reducing energy consumption by right-sizing the datacenter networks using different approaches. Lin *et al.* [63] propose to turn off idle servers. *CARPO* consolidates negatively-correlated traffic flows to keep a smaller subset of active links. Adnan and Gupta [10] proposed an online path consolidation algorithm to dynamically right-size the networks. These approaches perform well when there are lots of flows in the network. ElasticTree [51] propose to force traffic in a network to the leftmost switches in a *fat-tree* topology to power off unused switches. It is the most energy-efficient of all approaches, but is still sub-optimal because, as we show, for many light loading patterns, a large number of switches still need to remain active.

In this chapter, we present a concept of a merge network to be applied to switches to consolidate traffic. Our merging approach enables powering off more switches and links by merging traffic at edge and aggregation layer and scale network energy cost to the number of busy interfaces of each switch. By applying merge networks to each switch, we further reduce power consumption of active switches. With very light load, our approach saves 20% $\sim$ 40% energy cost compared with ElasticTree, depending on the traffic types. Traffic with small number of inter-pod and inter-subnet flows can benefit even more from traffic merging. We show that with traffic merging at switches, datacenter networks can achieve energy proportionality without changing the network topology and devices.

Chapter 7

SIMULATION RESULTS WITH MERGE NETWORKS

We simulate a $k = 12$ *fat-tree* network which supports 432 servers and 180 12-port switches. In this network, there are 12 pods and each of which has six edge switches and six aggregation switches. We assign 1Gbps capacity to each link and assume that each of the core switches has extra ports to be connected to external Internet through border routers. We experiment with synthetic traffic data from a traffic generator and real packet traces from a university datacenter. *Since flow splitting will incur packet reordering cost, which is not a desirable practice in real datacenters, we implement our simulation using non-splitting flow assignment.*

7.1 TRAFFIC DATA

7.1.1 Synthetic Traffic Data

The experimental traffic traces are generated following the On/Off pattern derived from production datacenters [79][20]. The duration of the On/Off period and the packet interarrival time follow the lognormal distribution. We generate different traffic type including *Random*, *Stride(n)*, and *Staggered(n)*, each of which has different patterns of near and far traffic. For instance, a flow in *Stride(n)* goes from node i and to node $[(i + n) \mod N]$ (N is the total number of servers). Source and destination nodes in *Random* type are uniformly distributed. *Staggered(n)* is staggered probability traffic and assigns fixed probabilities for traffic going to the

Table 7.1. Probabilities of flow going to the same subnet (p_1), to the same pod (p_2) and to different pods ($1 - p_1 - p_2$) for all traffic suites studied

Traffic Suite	p_1	p_2	$1 - p_1 - p_2$
Random	1.2%	7%	91.8%
Stride(1)	83.3%	13.9%	2.8%
Stride(6)	0%	83.3%	16.7%
Stride(36)	0%	0%	100%
Stride(216)	0%	0%	100%
Staggered(1)	100%	0%	0%
Staggered(2)	50%	30%	20%
Staggered(3)	20%	30%	50%

same subnet and to the same pod.

We generate 8 traffic suites with parameters p_1, p_2 and $1 - p_1 - p_2$ showed in Table 7.1. Flows in *Stride(1)* always go to the next server. In a $k = 12$ *fat-tree*, each edge switch connects to 6 servers and forms a subnet. Flows from the first 5 servers go to the same subnet. While flow from the 6th server travels to the next subnet or the next pod. Therefore, 5/6 of the traffic goes to the same subnet. Flows in *Stride(6)* always travel cross subnets. *Stride(36)* have all flows traveling to other pods. The load fraction λ offered by each server varied from 0.1 to 0.7.

7.1.2 Empirical Traffic Data

We use packet traces from a university datacenter published by Benson *et al.* [20]. This university datacenter has about 500 servers providing services for campus users. 60% of the traffic is for Web services and the rest is for other applications such as file sharing services. Traffic traces are captured by a sniffer installed at a

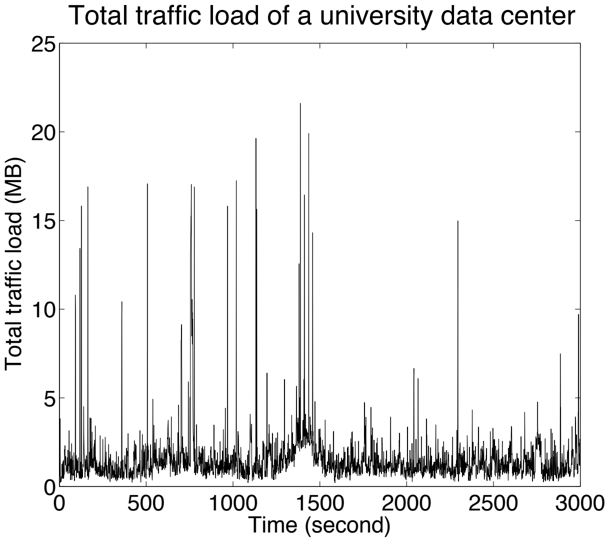


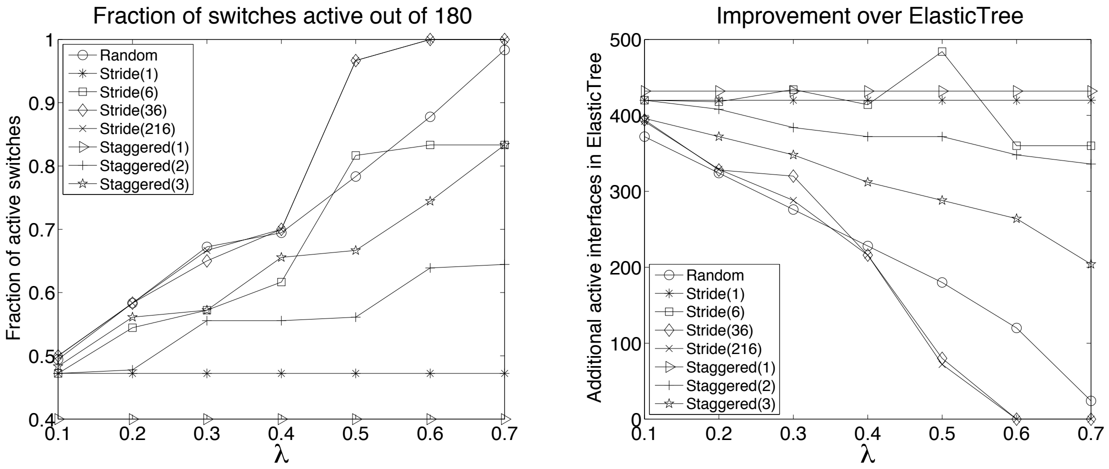
Figure 7.1. Traffic load of a university datacenter.

randomly selected switch in the datacenter. Figure 7.1 illustrates the total load of the packet traces within 50 minutes. The overall load is very small for a high-bandwidth *fat-tree* topology.

7.2 APPLYING MERGE NETWORK WITHIN A SWITCH

Our simulation outputs the number of active switches (Figure 7.2a) and the number of active interfaces of each switch with varies traffic loads and patterns. In general, the number of active switches increases with the traffic load. However, both *Stride*(1) and *Staggered*(1) have constant number of active switches and active interfaces. This is because, for *Stride*(1), all loads can be satisfied by using a minimum spanning tree. For *Staggered*(1), only edge switches are used since all the traffic flows are local traffic within the same subnet.

Figure 7.2b illustrates the difference of total numbers of active interfaces of a DCN using merge networks versus ElasticTree. It shows that more interfaces of the active switches become idle when the traffic is light, which demonstrates that



(a) Number of active switches.

(b) Total number of active interfaces.

Figure 7.2. Number of active switches and active interfaces network-wide for a $k = 12$ *fat-tree* network.

traffic merging can save more energy with lighter traffic (Figure 7.3). *Stride*(1) achieves the most energy savings over ElasticTree (around 42%) because, for each active edge switch, the energy consumed by the five idle interfaces is wasted. *Staggered*(1) saves 30% energy consumption since for the entire network, only half of the interfaces (facing the severs) of the edge switches are used.

ElasticTree provides an energy-efficient solution for DCNs. However, the drawback of ElasticTree is that, a DCN still consumes a large amount of power with light load [51]. In contrast, our approach reduces energy consumption when the network is lightly loaded, which demonstrates that traffic merging achieves better energy proportionality than ElasticTree.

We observe that power cost decreases from 30% to 17% when applying merge networks compared with ElasticTree (Figure 7.4).

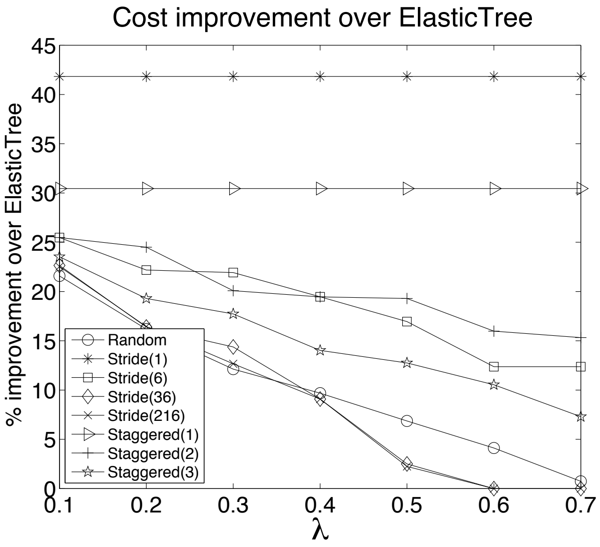


Figure 7.3. Reduction in total cost when using traffic merging.

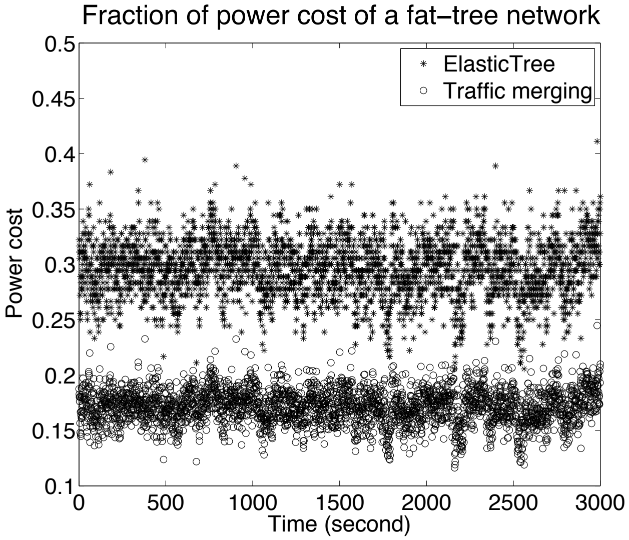


Figure 7.4. Energy savings when using traffic merging.

7.3 APPLYING MERGE NETWORK WITHIN A POD

The $k = 12$ *fat-tree* network consits of 180 12-port switches, organized in 12 pods and each of which has six edge switches and six aggregation switches. Every switch in the edge layer and aggregation layer, there are six uplink ports and six downlink ports. So within each pod, there are total 36 servers connected with six edge switches, and the number of links between edge layer and aggregation layer is 36. In this experiment, within each pod, we apply one 36×36 merge network between servers and edge switches, and another 36×36 merge network between edge switches and aggregation switches. The merge network switches traffic flows to the left switches. Flow splitting is allowed for simplicity.

7.3.1 Number of active switches

Figure 7.5 compares the number of active switches at each level before and after applying traffic merging. As we can find, for all traffic suites, the number of active switches at edge level reduces significantly after applying merge networks. For aggregation-level switches, we observe obvious reduction for *Stride(6)*, *Staggered(2)* and *Staggered(3)*. We notice from table 7.1 that these three traffic suites have higher p_2, and as we discussed in Section 6.2, a amount of λp_2 of traffic is switched away from aggregation level after traffic merging, which means a substantial part of the traffic originally going to aggregation layer has been cut short to be transferred directly through edge switches with merge networks.

Figure 7.6 illustrates the fraction of total active switches of the DCNs before and after using merge networks. The fraction of active switches reduced from $40\% - 60\%$ to $10\% - 35\%$ for light traffic loadings $(0.1 - 0.2)$.

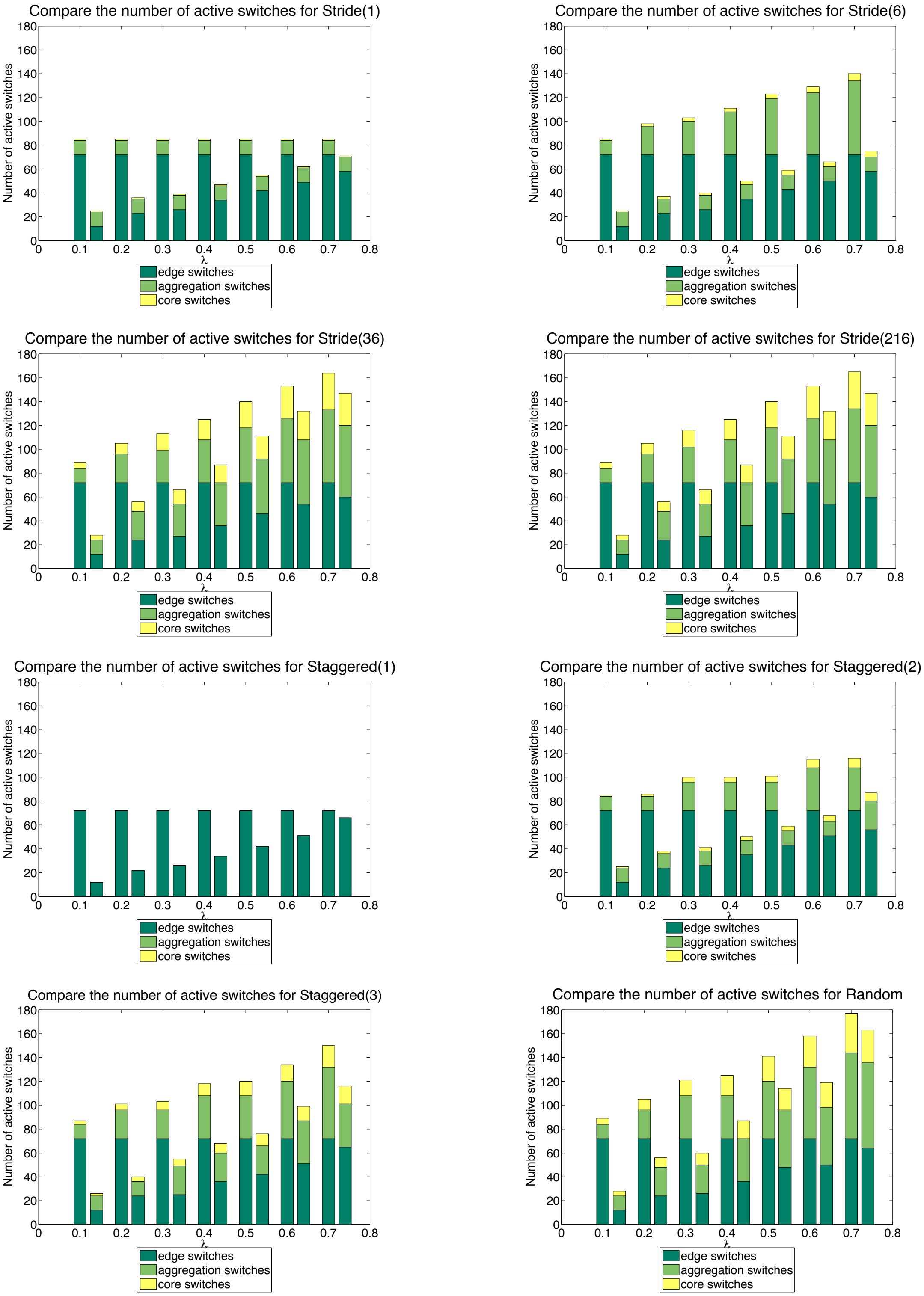


Figure 7.5. Compare number of active switches with vs. without traffic merging.

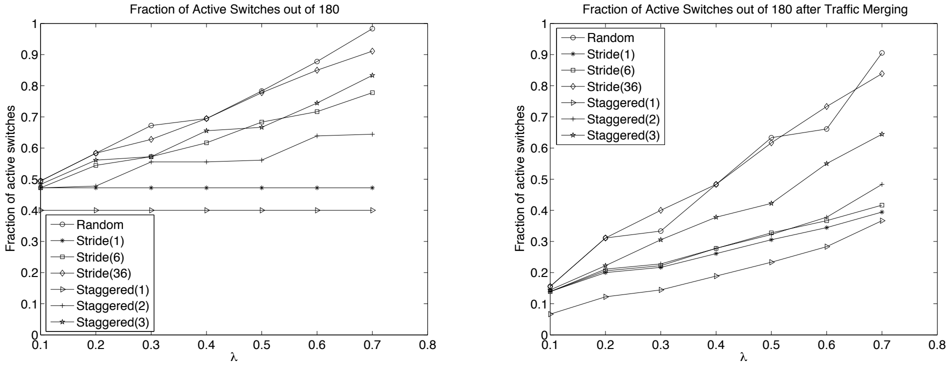


Figure 7.6. Fraction of active switches before using traffic merging (left) and after using traffic merging (right).

7.3.2 Energy cost

The above discussions focused on reducing the number of active switches. The overall energy cost a DCN consists of the cost incurred by switches and links. However, cost incurred by links is negligible and can be incorporated within the cost of interfaces of switches. Generally speaking, the energy cost of a switch can be roughly partitioned into the cost of chassis and the interfaces. As described in [82][34], a reasonable approximation to the cost of a k-port switch is:

$$\text{Switch Cost } = C + k\log k + k$$

The constant C accounts for static costs of the switch such as fan etc. The second term corresponds to the cost of the interconnection fabric within the switch, which is a significant contributor to energy consumption (typically $30\% - 40\%$). This cost scales as $k\log k$ for a k-port switch. The last term is the contribution to the cost from the active interfaces. This term folds into itself the cost of the linecards that the interfaces are on. For the purposes of comparing the *overall cost reduction* of traffic merging, we set C to 50% of the maximum switch cost. That is:

$$C = k\log k + k$$

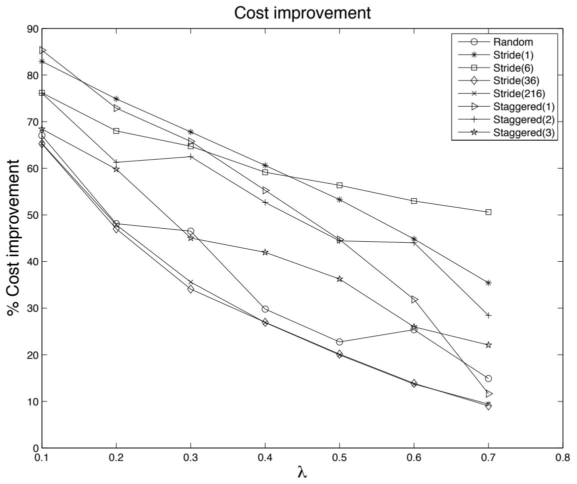


Figure 7.7. Reduction in total cost after using traffic merging.

If the traffic load fraction going to a switch is λ, the merge network will switch the traffic to the leftmost $m = \lceil \lambda k \rceil$ interfaces. The cost of a switch with merge networks is thus:

$$\text{Switch Cost } = C + m \log m + m$$

Figure 7.7 shows the overall cost improvement over approaches without merge networks. It demonstrates that our traffic merging method can save up to 90% of energy cost when the traffic load is low. Figure 7.8 shows that the traffic merging can achieve better energy efficiency that is closer to the ideal energy proportionality.

7.4 SUMMARY

We examine the approach of merging traffic by simulating a large-size *fat-tree* datacenter network and applying merge network at each switch and at a whole pod, and finding the lower bound of the minimum number of network switches and links that satisfies a variety of traffic patterns and loads similar to those from

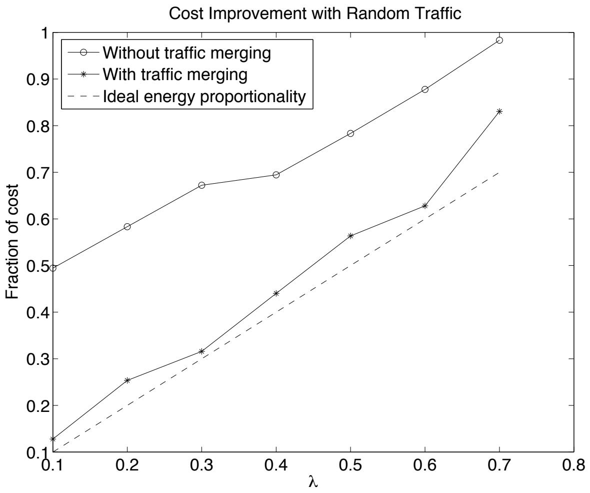


Figure 7.8. Fraction of total cost without traffic merging vs. using traffic merging.

actual datacenters.

Simulation results show that our approach can substantially reduce the number of active switches and lower the energy consumption of a *fat-tree* datacenter networks when the load is light. We show that our solution achieves up to $70\% - 90\%$ total energy savings through traffic merging and achieves almost perfect energy proportionality.

Chapter 8

PROTOTYPE OF MERGE NETWORKS

In previous chapters, we have studied the idea of merging traffic at a switch to consolidate traffic flows to minimize the number of active switch ports. In this chapter, we describe the implementation of a prototype of a simple 2 × 2 merge network built for optical switches. We use optical networks and devices for two reasons: 1) fiber optic cables are the fastest-growing transmission medium used in data centers today since they provide high bandwidth communication and reliable high-speed data transmission, and 2) passive optical networks (PON) can be used in the enterprise as point-to-multipoint solutions because passive optical splitters can distribute data, voice, and video signals throughout a network with greatly improved cost efficiency than Ethernet switches. Thus our work on merging traffic is related to the enterprise networks as well.

8.1 2×2 MERGE NETWORK ARCHITECTURE DESIGN

We utilize three Linux workstations to build a test-bed for a 2 × 2 merge network. Figure 8.1 shows the architecture of the test environment. We configure the top Linux machine named PACLAB11 as a virtual Linux bridge to simulate an L3 network switch with two interfaces with IP addresses, 192.168.0.11 and 192.168.0.14. The bottom two Linux machines, PACLAB12 and PACLAB13, work as two servers connected to the simulated network switch, PACLAB11. The network addresses of these two hosts are 192.168.0.12 and 192.168.0.13, respectively. We build a merge network to merge traffic to the left port of the network switch. In other words, the hosts always choose to connect to port 192.168.0.14 if it is available. Otherwise, the right-side port (192.168.0.11) will be used.

We implement the 2×2 merge network using two 2×2 optical switches, shown as the two beige boxes in Figure 8.1. To communicate with the optical switches, we install two Gigabit multimode SC fiber optic network adapters in PACLAB11, to act as two ports of the network switch. Each SC fiber optic network adapter provides a Transmitter (Tx) and a Receiver (Rx). We install one multimode SC fiber optic network adapter each in host PABLAB12 and PACLAB13. The left-side optical switch is the uplink switch, which connects the Transmitters of the two servers to the Receivers of the network switch. The optical switch on the right is the downlink switch, connecting the Receivers of the two hosts to the Transmitters of the network switch.

Figure 8.2 is the picture of the two 2 optomechanical optical switches used in our prototype [3]. A 2×2 optical switch has two states: Inserted State (A) and Bypass State (B) (Figure 8.3). The state of the optical switch is controlled by electric signals applied to the latches on the outer side the switch. For instance, when the uplink switch and downlink switch are both configured as in Bypass state, host PACLAC12 is connected to the switch port 192.168.0.11 and host PACLAB13 is connected to port 192.168.0.14. If both optical switches are in Insert state, PACLAB12 and PACLAB13 are connected to 192.168.0.14 and 192.168.0.11, respectively.

To implement priority on the left port, the controller has to decide which port is the left-most available port. Since it is not possible to detect whether the network switch port is busy or not when the host is disconnected from the ports of network switch, we use a variable named STATE to keep the status of the optical switches

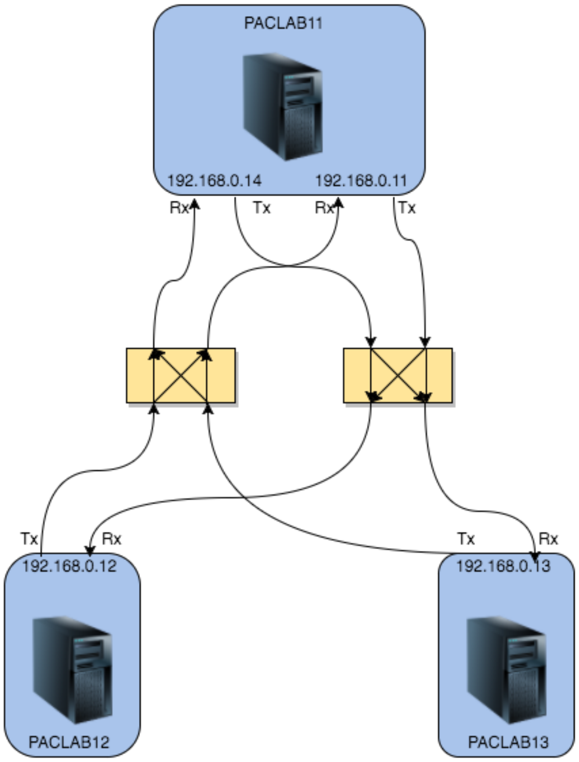


Figure 8.1. A 2 × 2 merge network implemented with two 2 × 2 optical switches.

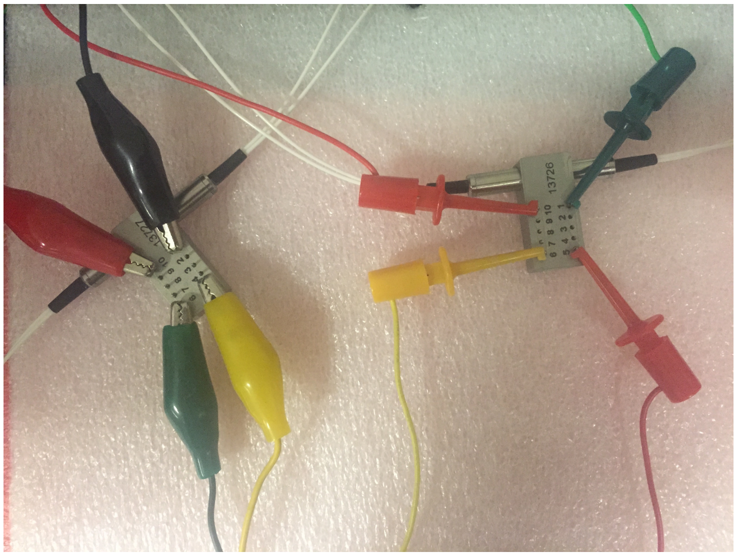


Figure 8.2. Optical switches

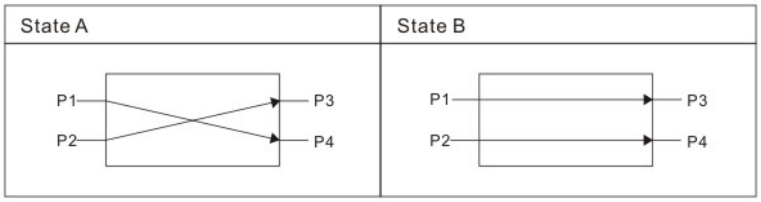


Figure 8.3. Two states of the optical switch.

and store it on the Arduino board. In our 2 × 2 merge network, the STATE corresponding to different scenarios is listed as follows:

1. STATE = 0: idle state - no port is in use
2. STATE = 1: both uplink and downlink switches are in Bypass state
3. STATE = 2: both uplink and downlink switches are in Insert state
4. STATE = 8: conflict state - two hosts send contradictory states

We implemente a communication protocol between the hosts and Arduino to negotiate the merge network state. Before a host starts to transfer data, it reads the current STATE from Arduino. If the Arduino is already set to one of the active states (STATE = 1 or STATE = 2), the host will update its own state variable, myState, as the same value of the STATE of merge network and starts to send data using the current setting. If STATE = 0, the host will set mySTATE to a value that can make the merge network connect the host to the left port of the network switch. The host sends myState, as a STATE update request, to the Arduino. The host will check STATE again and start to send data if STATE is set to 1 or 2. If STATE = 8, it means that the other host was trying to set up merge network to the opposite state at the same time, in other words, competing for the left port. In this case, the host will back off for some random time and set myState to 0 and restart.

Figure 8.4 shows the flowchart of the logic to determine myState implemented on host PACLAB12. When current STATE is 0, there is no active data transmission. Host PACLAB12 needs to set the merge network to STATE = 1 in order to be connected to the left-side port. If the current STATE is 2, which means host PACLAB13 is connected to the left port, PACLAB12 can only use the right port and keep STATE = 2. For host PACLAB13, its preferred STATE is 2 and will set up myState to 2 when the board STATE is 0. We implement the state-determine logic in the socket connection function at each host as part of the application-level protocol.

We use a 5V Arduino Uno board as the controller to negotiate STATE with hosts and to send control signals to optical switches and to coordinate them to provide connection channels between the hosts and network switch (Figure 8.5). Arduino Uno consumes only 232mW power when it is active. The two optical switches are passive optical devices with no power requirement. As such, the power consumption of the 2×2 merge network is negligible. In addition, the optical switch requires minimum management and it is highly reliable. It supports multimode optic fiber operating at a wavelength from 650nm to 1310 nm.

The architecture design of the merge network is illustrated in Figure 8.6. The Arduino board works as a micro-controller, receiving state inquiries and state updating requests from hosts, and controls optical switches by outputting control signals to the latches of optical switches. The Arduino board communicates with the hosts through serial ports. Given there is only one hardware serial port in Arduino Uno, we use two pins on Arduino to work as Tx and Rx of a serial port and simulate a software serial port to communicate with the second host.

The Arduino reads myState values from the two hosts and sets up new STATE, which is sent to the optical switches to change the switching states. The new value

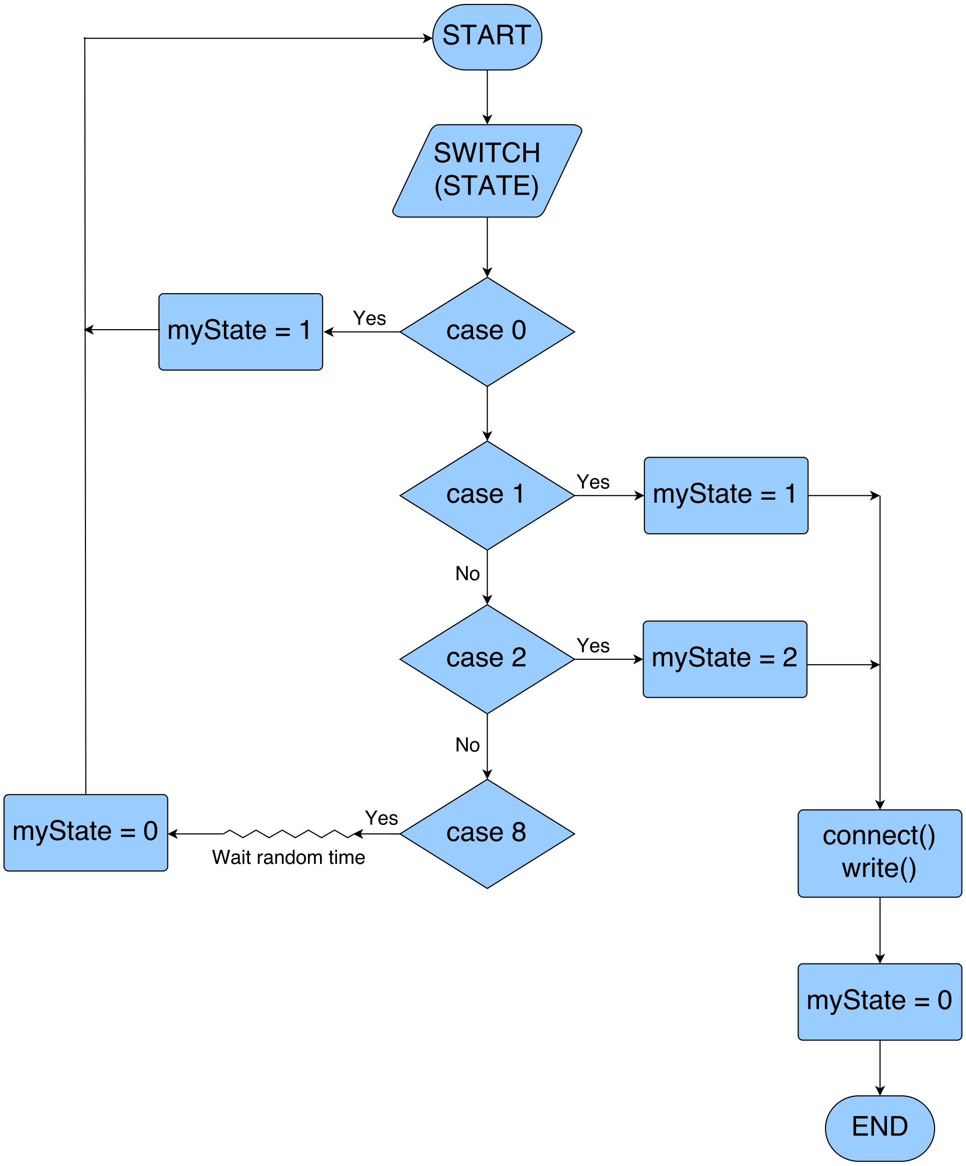


Figure 8.4. Controlling the state of merge networks: state-transferring logic implemented at PACLAB12.

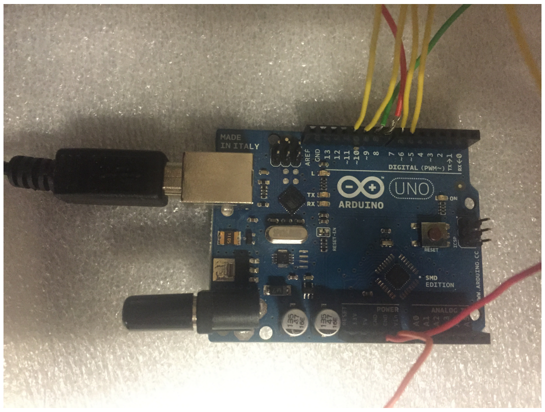


Figure 8.5. Arduino board to control the states of the two optical switches.

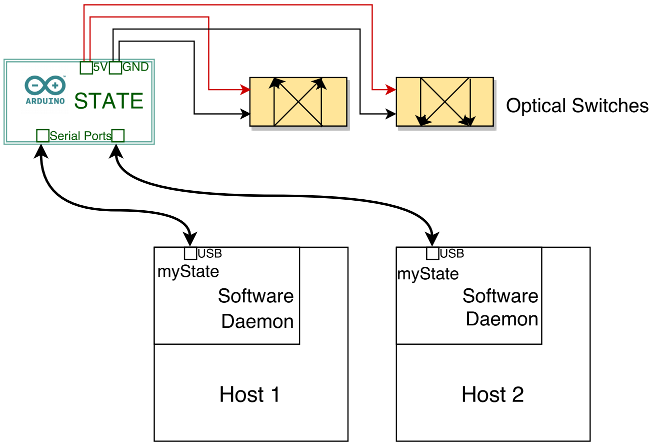


Figure 8.6. Architecture design of a 2 × 2 merge network.

Table 8.1. Arduino board STATE values

myState2 myState1	0	1	2
0	0	1	2
1	1	1	8
2	2	8	2

of the STATE variable is determined according to the values of the state request variables received from the two hosts, myState1 and myState2. The logic is shown in Table 8.1. If the new STATE is 1 or 2, the Arduino board will send control signals to the uplink and downlink optical switches to turn them into Bypass or Insert state. If the new STATE is 0, Arduino will turn both optical switches off. It is possible for the two hosts to send contradictory myState values (e.g. one sends 1 and the other sends 2). This situation happens when the current STATE is 0 and both hosts have data to transfer at the same time, and each of the two hosts considers itself the only sender and can use the left port to transfer data. As a result, one host sets myState to 1 and the other host sets myState to 2. When Arduino receives two different myState, the new STATE will be set to 8. When the hosts detect this situation, they will reset myState to 0, wait some random time, and try again.

8.2 MEASUREMENT RESULTS

We test the utilization of the two ports in a switch (implemented at workstation PACLAB11). We use Iperf to send packets from host1 and host2 and customize the traffic flows with log-normal distribution of flow length and inter-arrival time to generate traffic with different loadings. We measure the active time periods of the two ports and compare the throughputs with that of a switch without a merge network.

We use the Iperf application to send traffic packet flows to the switch, with overall loadings from about 10% to 75%. The two ports of the switch are named Port 1 (left) and Port 2 (right) for simplicity. When there is no merge network used, Port 1 and Port 2 are connected to Host A and Host B respectively, and they are always in an active mode. With the merge network at the switch, the available left-most port is chosen first. That means, in our test case with the switch having two ports, the left port (Port 1) always has higher priority than Port 2 (on the right).

We use green color to represent the traffic flows going to the left port (Port 1) and red color for the flows to Port 2. Figure 8.7 illustrates the ten-hour record of the traffic flows from Host A and Host B. Host A sends most of the traffic flows to Port 1. Host B also uses Port 1 when loading is low, and uses Port 2 more when loading increases to 75%.

In our experiment, the two hosts' software settings are exactly the same. The reason that Host A can seize more time of the Port 1 is that we use Arduino UNO as the state control circuit of the merge network. Each of the two hosts sends its state control signal to the Arduino through a serial port. Since Arduino UNO has only one built-in hardware serial port, we use a software serial port to simulate a hardware serial port and let Arduino receive the signal from the second host. Although we set the two serial ports with the same transfer baud rate, it appears that the host using the hardware serial port can always get connected to Port 1 faster and more frequently.

In this first experiment, Host A is connected to the hardware serial port and

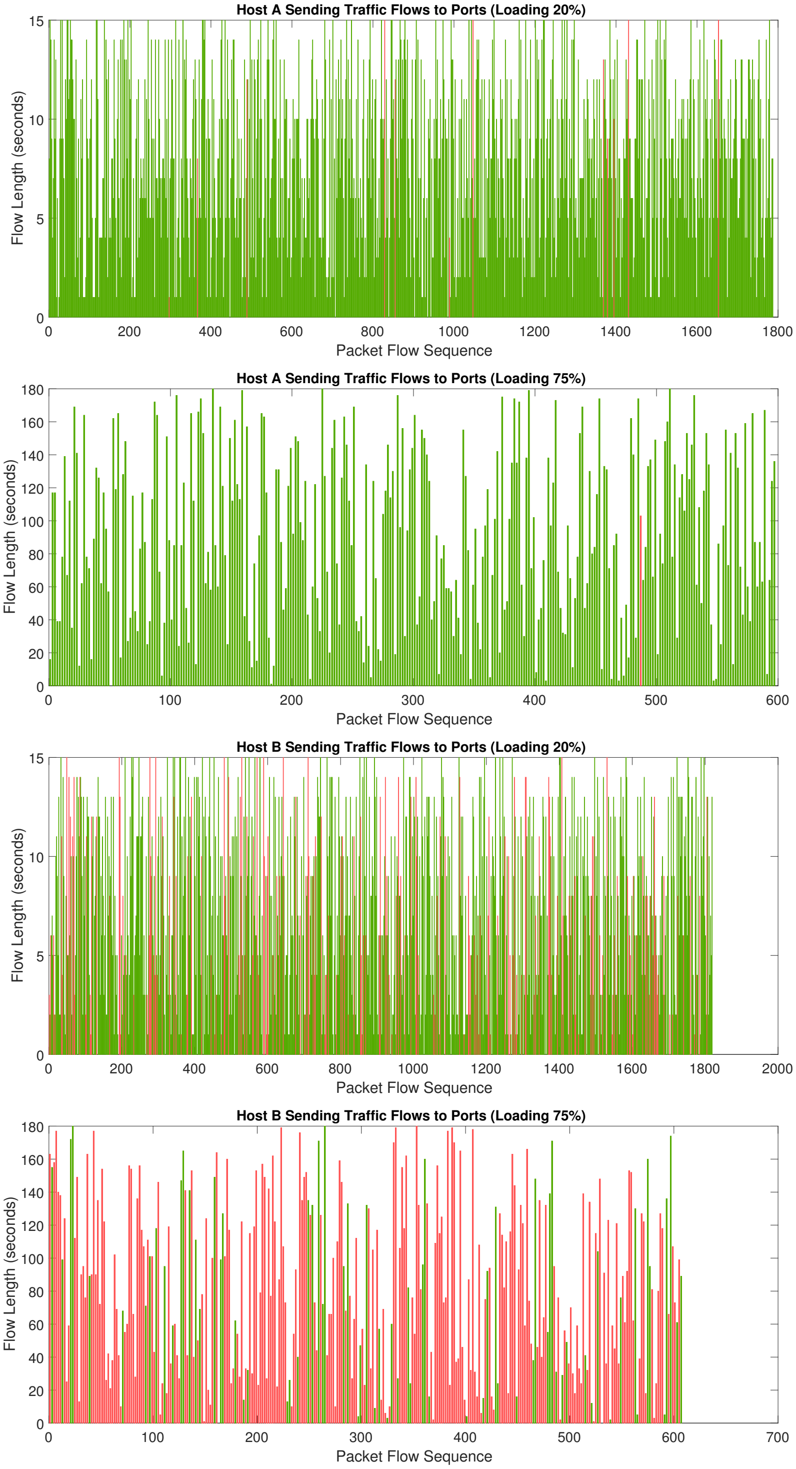


Figure 8.7. Traffic flows and port usage of Host A and Host B.

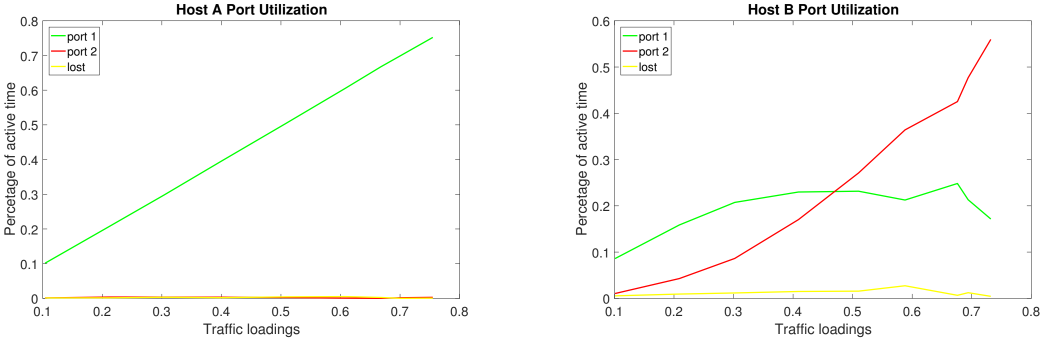


Figure 8.8. Total port usage of Host A and Host B.

Host B is linked to the software serial port. The assumption above explains why Host A sends most of the traffic flows using Port 1. Figure 8.8 shows the total percentage of traffic flows that is sent to each of the ports from Host A and Host B, with loadings increase from 10% to 75%. In this figure, the green line represents the usage of Port 1, and the red line represents the usage of Port 2. It is obvious that Port 1 is the dominant port used by Host A. For Host B, when loading is approximately below 45%, the traffic flows from Host B can still squeeze into Port 1. As a result, more traffic is going to Port 1 than going to Port 2. When loading is above 45%, the total traffic from the two hosts will approach the capacity of each port. As a result, Host A occupies Port 1 most of the time, and Host B increasingly sends traffic to Port 2.

To verify our assumption that the different performance results of Host A and B are caused by the difference of the Arduino hardware serial port and software serial port, we switch the hardware and software serial ports between Host A and B in the second set of experiments. Figure 8.9 shows the reverse results from that, as shown in Figure 8.7, which verify that the host connected with hardware serial port of the Arduino (i.e. Host B) uses Port 1 more. The total port utilization shown in Figure 8.10 also proves that Host B sends significantly more traffic to

Port 1 after we switch the serial port connections.

To show the results without the influence of Arduino serial port, we sum the two experiment results and calculate the average. The results shown in Figure 8.11 illustrate very similar performance of Host A and Host B, and indicates that more traffic goes to Port 1 for both Host A and Host B. It is also possible to achieve this result by using a control circuit with two hardware ports. For example, Arduino Mega provides 3 extra built-in serial ports on board.

We calculate the overall utilization of the two ports by adding the traffic from Host A and Host B. The result is shown in Figure 8.12. The x-axis represents loading increase. The y-axis represents the percentage of the active time of the port. We can see that Port 1 is used much more than Port 2, especially when the loading is smaller. When the loading is about 75%, Port 1 is used close to the line capacity and Port 2 is active half of the time. This proves that the merge network perfectly consolidates the traffic to the left port, which is Port 1.

We extract the state switching data from the Arduino board in order to understand how frequently the merge network changes its switching state and illustrate the result in Figure 8.13. The x-axis is the workload. The y-axis is the number of times that the state switching command is received from each host. We find that the host connected with the software serial port of the Arduino board switches state more. The host connected with the hardware serial port maintains a much lower and more stable number of state switching.

We add the state switching number of Host A and Host B and compare the total numbers of state switching in experiments 1 and 2, we find very similar curves of the two, Figure 8.14. Both curves start to climb from abut 300 first when loading increases, and they peak when loading reaches about 30%. After that, the curves start to descend, totaling lower than 100 when the loading increases to 75%.

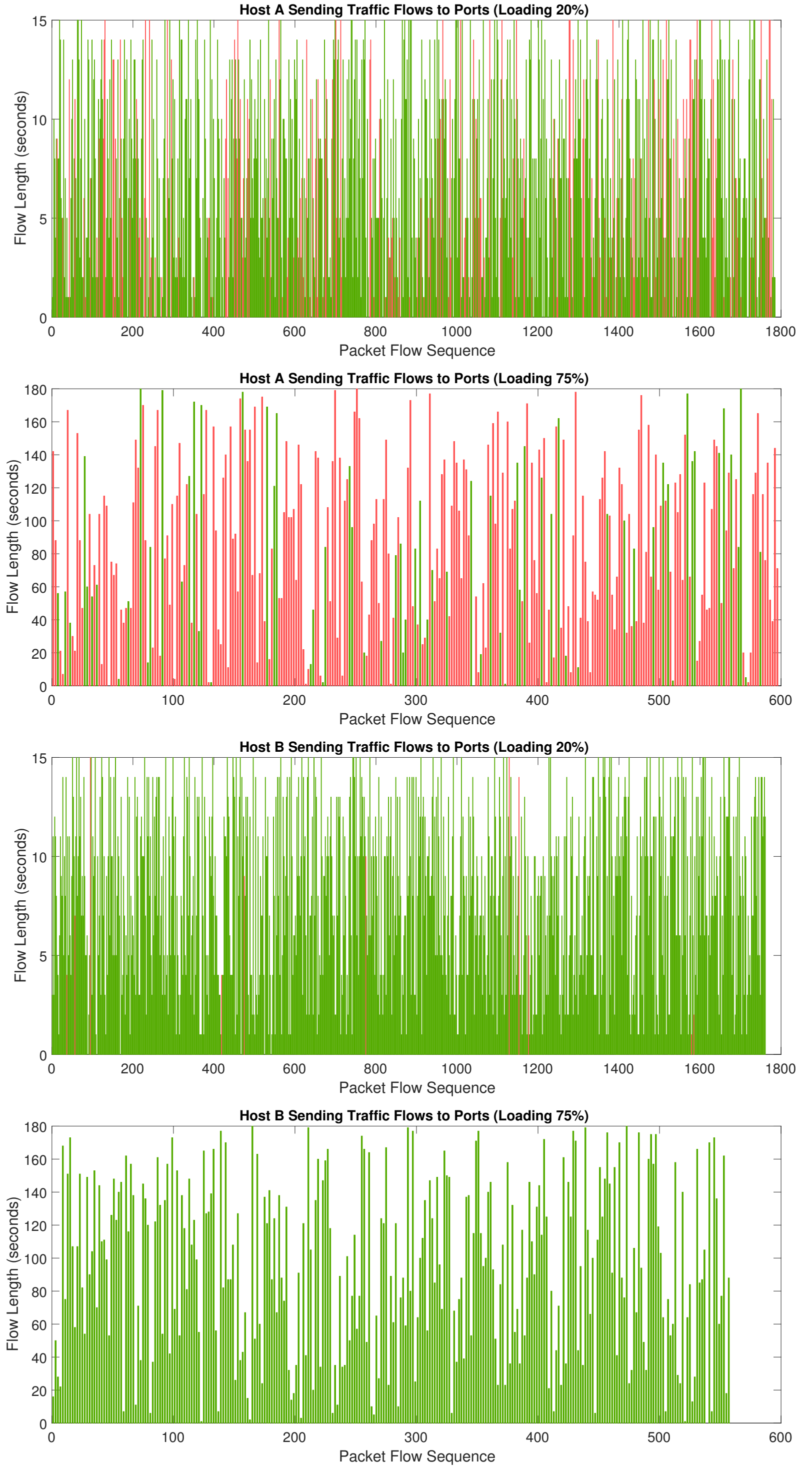


Figure 8.9. Traffic flows and port usage of Host A and Host B after switching serial ports.

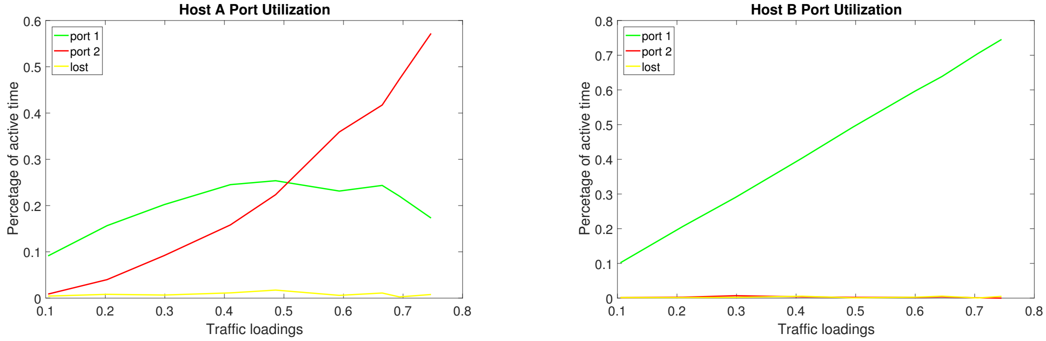


Figure 8.10. Total port usage of Host A and Host B after switching serial ports.

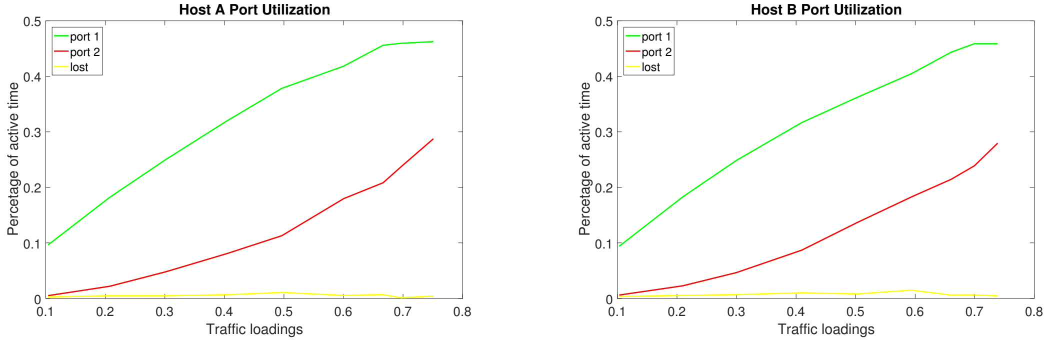


Figure 8.11. Average port usage of Host A and Host B.

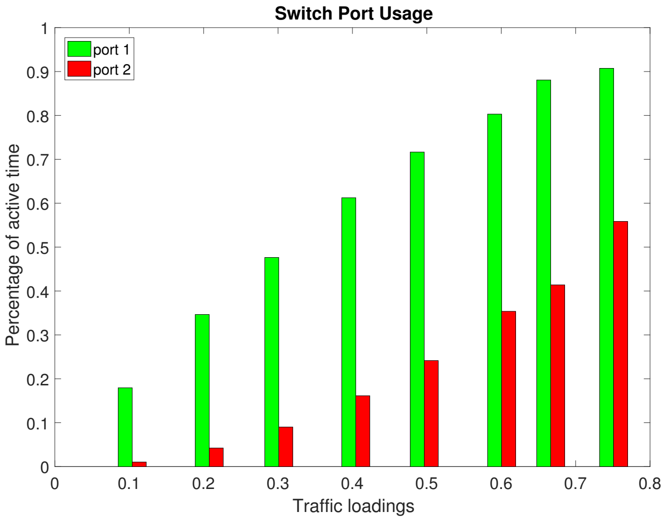


Figure 8.12. Total Port 1 and Port 2 utilization.

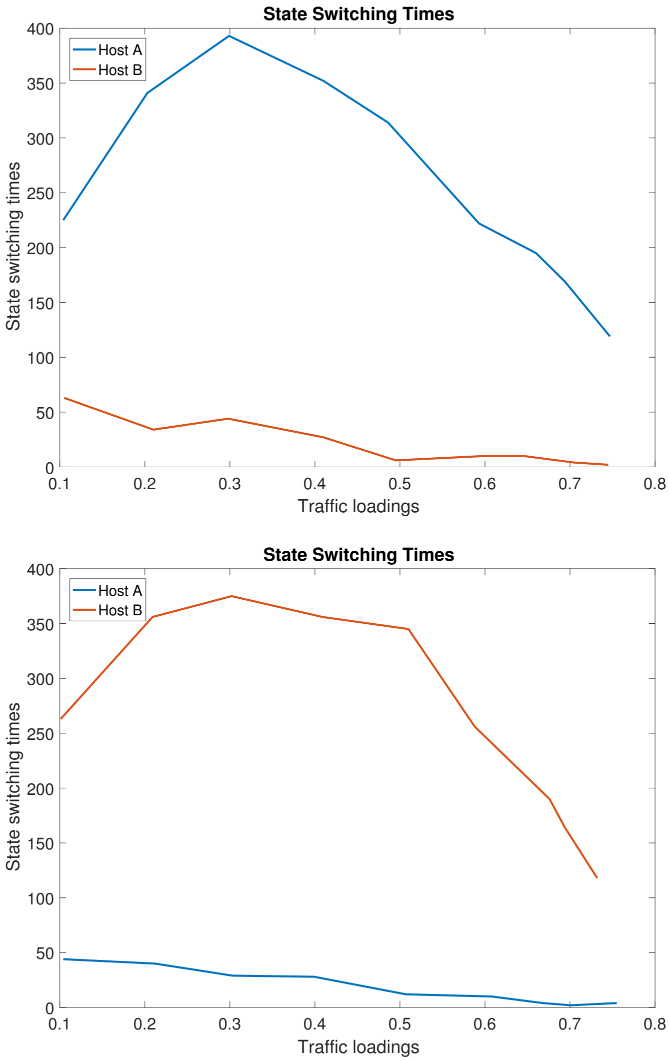


Figure 8.13. State switching times of the merge network in two experiments.

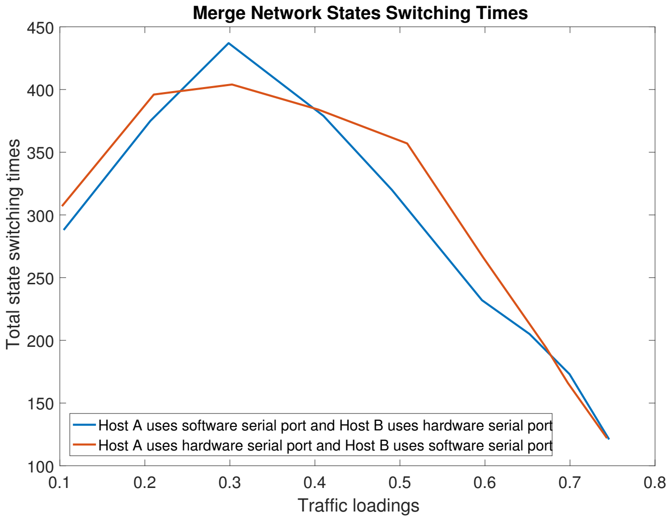


Figure 8.14. Total state switching times.

For controlling state switching, there are two possible strategies. One is non-blocking transfer: when a host's preferred port is busy, the host can connect to the other port immediately. The other strategy is blocking transfer: when a host's preferred port is busy, the host can wait some time to see if the preferred ports can be available soon. In our experiments, we use non-blocking strategy to guarantee that all traffic load is sent immediately without delay, thus to ensure the network performance. In future work, we can experiment with the blocking strategy and find the threshold of the waiting time that can guarantee reasonable throughput and latency.

8.3 HIGHER-ORDER MERGE NETWORKS

When we apply a merge network to all ports of a ToR switch, or several ToR switches within a datacenter pod, we need merge networks with more inputs/outputs. If we have a $N \times N$ optical switch like Figure 8.2, we can use a pair of

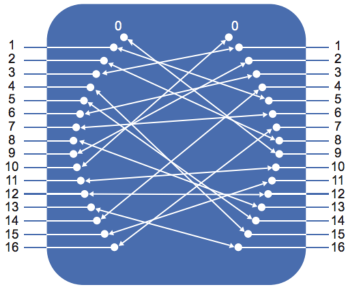


Figure 8.15. Example of a 16 × 16 MEMS matrix optical switch.

these optical switches to work as uplink and downlink switches to build a $N \times N$ merge network that connects N hosts to a N-port network switch, similar to the architecture described in Figure 8.6.

An example of such a switch is a MEMS matrix optical switch [4] that support all-optical cross connections in a fully non-blocking manner, allowing simultaneous connection between a number of input and output fibers. Figure 8.15 is a 16 × 16 matrix switch. Any of the 16 input fibers can be connected to any of the 16 output fibers.

The MEMS matrix optical switch is based on the micro-electro-mechanical system (MEMS) mirror technology, which uses a MEMS chip to rotate a matrix of movable silicon mirrors to change the coupling of light between input fibers and output fibers (Figure 8.16 left). The use of MEMS technology offers low cost and excellent optical performance of high reliability and fast switching time of less than $\leq 40ms$. Currently, the MEMS matrix switch is available in sizes up to 32 × 32 (Figure 8.16 right). Using a modular design, it is also possible to customize the optical switch to larger-size $N \times N$ configurations.

The MEMS matrix optical switch is controlled through a RS232 interface or I^2C. We generalize the algorithm we used for the 2×2 optical switch to the $N \times N$

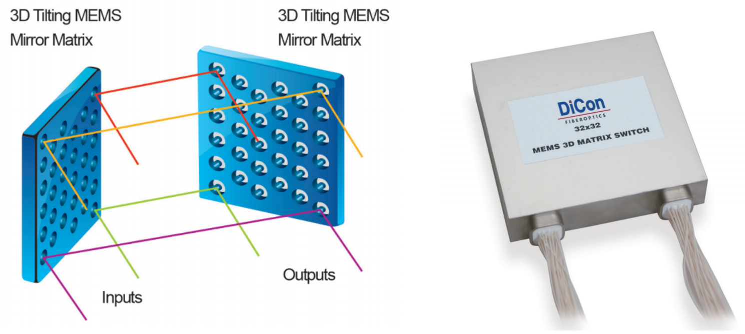


Figure 8.16. MEMS 3D matrix optical switch.

optical switch. We use an array to keep the STATE of the current switching state of optical switches. The array's size is equal to the dimension of the matrix optical switch. The value of the ith item of the array stores the name of the host whose connection is switched to the ith output port. For example, consider four hosts named A,B,C and D. Assume the STATE array of the 4 × 4 matrix switch shown in Figure 8.17 is CADB. This means host C is connected to port 1 (leftmost), host A is connected to port 2, and so on. When there is no traffic from a host, the value of corresponding item of STATE is set to 0. From the example shown in Figure 8.17, if at this time the packet flow from Host A is completed, the STATE array will be changed from CADB to C0DB. Following that, if Host C is done, the STATE changes to 00DB. The STATE array records which ports are available for the following traffic flows, and the algorithm can use the value of STATE giving the leftmost available port highest priority. For example, if the next traffic flow is from Host A again, the STATE will change from 00DB to A0DB.

The STATE array keeps the current outputs of the matrix optical switches. It contains information for two types of ports: for active ports that have data flows, it represents current switching logic of those ports. For output ports that are currently idle, it stores 0 in those corresponding positions. The 0s in the

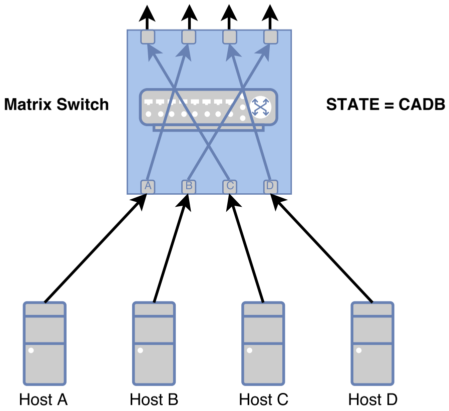


Figure 8.17. STATE of a 4 × 4 matrix switch.

STATE array are used to determine whether there is a further-left port available when new traffic arrives, thus to consolidate traffic to the left side of datacenter switches. However, since the matrix optical switch is non-blocking, the control signal sent to the optical switch has to specify the switching path of every port, even if there is no packet go through it. Since the STATE array may contain many 0s for idle ports, it cannot be used to control the switch state transfer. Therefore, we use another array, SIGNAL, to store the control signals for the optical switches.

We take a 4 × 4 optical switch as an example. The STATE array is of size 4 to store the output port states. It is initialized as 0000 since all the four ports are idle at the beginning. We set up SIGNAL = ABCD to start the optical switch in the bypass state initially. For the traffic flows, we use '+A' to represent that Host A starts sending data, and '-C' to describe that Host C's data transmission is completed. Ans we use a string to represent a sequence of starts and ends of data transmission from specified hosts. For example, "+A+B-B-A" means a sequence of events - "Host A starts data transmission; then Host B starts data transmission; Host B data-transmission ends; and then Host A data-transmission ends". We illustrate the changing values of STATE and SIGNAL arraies when we have the

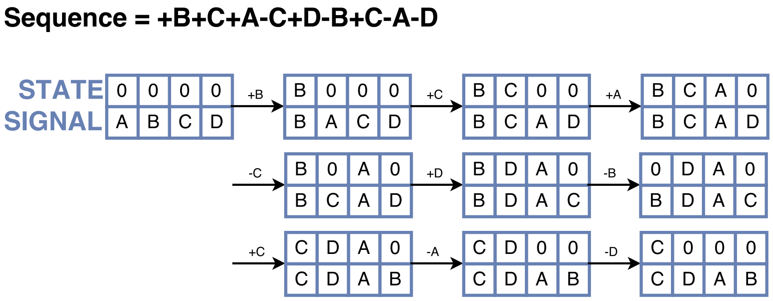


Figure 8.18. A 4 × 4 matrix switch: STATE and SIGNAL.

sequence of data flows shown in Figure 8.18.

For this 4 × 4 case, the STATE array starts from 0000, and changes every time when a new data flow starts and an old data flow completes. When a new data flow starts, the algorithm finds the first available port (first zero) from STATE array and updates that item to the name of the host that sends the data flow. When a host completes its data transmission, the algorithm traverses the STATE array again, finds the item with the host name and changes it to 0. The algorithm needs to traverse the STATE array twice for each data flow, so the time complexity is O(2n). The SIGNAL array starts from ABCD and updates simultaneously with the STATE array. For example, when a new data flow starts, the algorithm changes the ith item of STATE array to Host x. Also, it checks the value of ith item of SIGNAL. If it is not equal to Host x, the algorithm finds Host x at jth position of SIGNAL, and switches the ith item and jth item. The time complexity of updating SIGNAL array is O(n). Therefore, the overall complexity of this algorithm is O(3n). The pseudo-code of the algorithm is described in Algorithm 2. The SIGNAL array is used to control the matrix switches. It only needs to be updated when there is a new data flow coming in. However, the STATE array needs to be updated when a data flow starts and completes to store the real status of each ports of the merge network.

Algorithm 2 Algorithm to update state and control the matrix switch

```
function SwitchControl($host$)
    $STATE \leftarrow \{0\}$;
    $SIGNAL \leftarrow \{A, B, C, D\}$;
    loop
        if Host $x$ has data to transfer then
            for $i \leftarrow 0; i < SwitchDimension; i \leftarrow i + 1$ do
                if $STATE[i] == 0$ then
                    $STATE[i] \leftarrow x$;
                    if $SIGNAL[i] <> x$ then
                        for $j \leftarrow 0; j < SwitchDimension; j \leftarrow j + 1$ do
                            if $SIGNAL[j] == x$ then
                                $swap(SIGNAL[i], SIGNAL[j])$;
                                break;
                    break;
            Send SIGNAL to matrix switch;
        if Host $x$ complete data transferring then
            for $i \leftarrow 0; i < SwitchDimension; i \leftarrow i + 1$ do
                if $STATE[i] == x$ then
                    $STATE[i] \leftarrow 0$;
                    break;
```

When there are more than one hosts requesting to send data within a short period of time, it is more efficient to change STATE and SIGNAL for all host requests and send the final SIGNAL to the switch, considering the switching time required for each state change of the matrix optical switches.

In the prototype of 2×2 merge network, we implemented the control algorithm in the software running on the Arduino board, which receives inputs from the two hosts through serial ports. For the general implementation of a $N \times N$ merge network, we can use optical detectors to collect the data input fiber from each host. DiCon optical detector [5] provides in-line power monitoring by utilizing fused couplers on every input, which taps off a portion ($1\% \sim 10\%$)of the signal and delivers it to the output (Figure 8.19). We propose to integrate the optical switch, the micro-controller and the optical detectors as an integrated merge network module. The architecture design of the functional module is shown in Figure 8.20. We have found some successful examples of customized function modules. Figure 8.21 is an example of a module integrated by Dicon [6] with MEMS VOAs, tap-detectors, control electronics, and firmware, for optical power balancing and management. With our control algorithm, the micro-controller can be integrated with the tap-detectors, and the matrix switch, to make a functional module of an $N \times N$ merge network.

8.4 SUMMARY

This chapter describes the implementation of a 2×2 merge network using optical switches. Hosts that have data to transfer send a request to an Arduino controller, which calculates the control signal and sends it to the optical switch. The 2×2 merge network successfully consolidates the data to the left port of the network

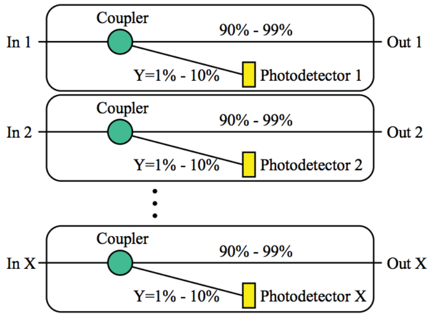


Figure 8.19. DiCon Tap/Detector module.

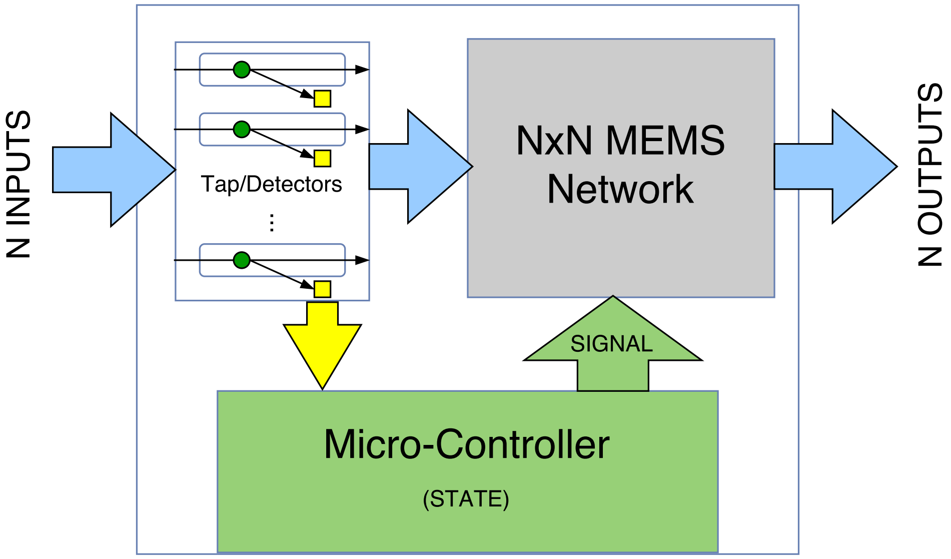


Figure 8.20. Customized functional module of merge networks.

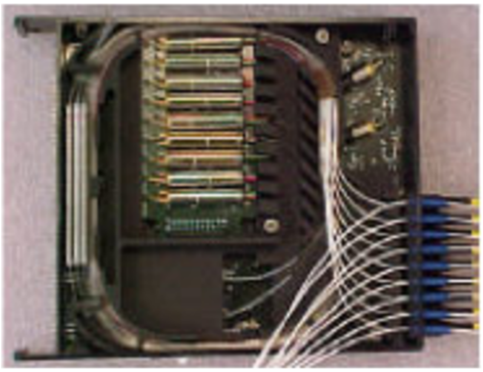


Figure 8.21. DiCon customized module.

switch. We extend the algorithm to a general $N \times N$ case and discuss the architecture design of integrating optical tap-detector, optical switches, electronic controller and firmware to form a functional $N \times N$ merge network that can be used to consolidate traffic to leftmost ports of edge switches within a datacenter pod.

Chapter 9

CONCLUSIONS

9.1 SUMMARY

In this research, we consider the energy efficiency problem of datacenter networks. Since *fat-tree* topologies are the predominant choice for datacenter networks, lot of our work was based on this topology. While *fat-trees* provide full bisection bandwidth, which minimizes latency and boosts throughput, the energy consumption of this network, and other datacenter networks, is not proportional to the network load. We find that the usage of the network devices in a *fat-tree* network is greatly dependent on the type of traffic, the traffic load, and the selected routing algorithm. For instance, if most of the traffic is between servers located in the same pod, the core switches are never used, even at high loads. On the other hand, if most of the traffic is between servers in different pods, the better part of the network switches will be in active states even at low loads since more switches in the network need to be utilized for routing.

We first analyze the problem of energy consumption in *fat-tree* networks by deriving expressions for the fraction of active switches and traffic losses for arbitrary traffic loads and traffic losses. The developed analytical models for energy consumption enable us to study *fat-tree* DCNs theoretically. We show that there is a base cost of approximately 45% (due to edge switches), but after that point energy consumption can scale linearly by appropriately consolidating traffic flows.

A practical application of the models is to jointly optimize task scheduling and flow assignment so as to maximize the traffic consolidation for given job loads.

Based on the energy consumption model, we investigate skinnier network topologies that meet performance requirements of realistic loads, thus saving not only energy but capital cost as well. Through a comprehensive study of the sub-graphs of *fat-trees* for different traffic characteristics, we conclude that it is possible to further reduce the number of active switches by up to 10% by consolidating corresponding jobs to fewer servers, particularly at low loads. Furthermore, we find that edge switches account for a large portion of the energy cost even at very low loads. We propose to replace the edge switches with high cardinality switches and build energy proportional DCN. We evaluate the DCN power consumption using the power data of Cisco modular switches. We find that the overall power consumption is significantly reduced by using high-radix edge switches in the edge layer of *fat-tree* DCNs.

In order to find the minimum subset of a network, we formulate an optimization model for computing routes with the goal of minimizing energy consumption and use Cplex solver to find optimal solutions for a small *fat-tree* network. Routing plays an important part in the potential for energy savings. Compared to routing algorithms that seek to balance load, our routing algorithm consolidates traffic into a few paths to save energy at the idle switches. We use a universal greedy flow assignment algorithm, which is proved to be able to find flow assignments close to that achieved from the optimization solver for a variety of loading scenarios. Although the greedy bin-packing algorithm used in ElasticTree also finds the shortest route using the left-most heuristics, it leverages the regularity of hierarchical DCNs. Our greedy algorithm can find flow assignments close to the MIP model, for not just hierarchical network topologies, but also random or irregular

DCN topologies.

Many approaches that address the DCN energy efficiency problems can hardly achieve the goal of energy proportionality. There is still considerable amount of energy waste especially when the network is very lightly loaded during off-peak hours. For example, in ElasticTree, the edge switches are always fully powered on, even during the idle hours, because they are connected to servers. At the aggregation layer, switches that are powered on do not fully load their interfaces facing the edge switches. Our proposed merging approach explores additional savings made possible by use of a hardware device called a *merge network*, which further consolidates the traffic to fewer switches and enables powering off a subset of interfaces in active switches, thus to manage the power at a finer granularity.

We attach merge networks to each switch of the network, and alternatively, to all switches of the same layer within a pod, so as to scale the network energy cost to the number of busy interfaces. We customize the analytical model to include the merge networks by including the number of active interfaces as a parameter in the minimization function. The model shows that, in addition to the savings obtain by forcing traffic to the left, as shown in ElasticTree, we can achieve significantly additional savings by powering off unused interfaces in active switches, which is made possible by merge networks. Simulation results prove that the merge networks can reduce the energy consumption by around 50% at light loads, and the DCN energy consumption can scale linearly by appropriately consolidating traffic flows.

In simulation of larger *fat-tree* networks, we analyze the energy savings obtained when using merge networks. With very light load, our approach reduces 20% to 40% energy cost compared with ElasticTree by applying merge networks to each switch, depending on the traffic types. Localized traffic can benefit even more from traffic merging. When deploying merge network at edge-layer and aggregation-layer switches within the same pod, the traffic merging achieves up to $70\% - 90\%$ total energy savings and the network exhibits power-usage behavior close to that of an energy-proportional system.

As a proof of concept, we design and build a hardware prototype of a 2×2 merge network using fiber links and passive optical devices. We experiment with the merge network in a small test bed built on Linux boxes and Arduino. The merge network consolidates data to one interface, with a slight time delay due to switching time of the optical switches. The energy cost of the prototype is minimal. We extend the prototype to larger-size merge networks by generalizing the control algorithm and hardware design. The system can be built with reasonable expense compared with the cost of datacenter network devices. The time and space complexity of the control algorithm is linear and the system requires minimal change at the end hosts.

9.2 CONCLUSIONS

An important conclusion of this research is that the type of traffic has a close correlation with the potential energy savings for datacenter networks. This is clearly demonstrated in the simulations as well as in the analytical models developed in this thesis. The key thought is to keep traffic local as much as possible in order to save energy. Another conclusion is that the symmetric design and homogeneous network equipment is not generally energy-efficient. Topology enhancement for providing external connectivity and heterogeneous deployment of network devices has an impact on the overall energy consumption.

REFERENCES

[1] http://www.datacenterknowledge.com/archives/2011/08/01/report-google-uses-about-900000-servers/.

[2] http://tools.cisco.com/cpc/.

[3] http://www.fs.com/products/32884.html.

[4] https://www.diconfiberoptics.com/products/mems_matrix_optical_switches.php.

[5] https://www.diconfiberoptics.com/products/scd0308/scd0308A.pdf.

[6] http://www.diconfiberoptics.com/products/prd_custom.php?sec=modules.

[7] IEEE 802.3az. http://www.ieee802.org/3/az/.

[8] Dennis Abts, Michael R. Marty, Philip M. Wells, Peter Klausler, and Hong Liu. Energy Proportional Datacenter Networks. In *ISCA*, 2010.

[9] Hussam Abu-Libdeh, Paolo Costa, Antonu Rowstron, Greg O'Shea, and Austin Donnelly. Symbiotic Routing in Future Data Centers. In *SIGCOMM*, pages 51–62, 2010.

[10] Muhhamad Abdullah Adnan and Rajesh Gupta. Path Consolidation for Dynamic Right-sizing of Data Center Networks. In *Proceedings IEEE Sixth International Conference on Cloud Computing*, 2013.

[11] Mohammad Al-Fares, Alexander Loukissas, and Amin Vahdat. A Scalable, Commodity Data Center Network Architecture. In *SIGCOMM*, pages 63–74, 2008.

[12] Mohammad Al-Fares, Sivasankar Radhakrishnan, Barath Raghavan, Nelson Huang, and Amin Vahdat. Hedera: Dynamic Flow Scheduling for Data Center Networks. In *NSDI*, 2010.

[13] Mohammad Alizadeh, Albert Greenberg, David A. Maltz, Jitendra Padhye, Parveen Patel, Balaji Prabhakar, Sudipta Sengupta, and Murari Sridharan. Data Center TCP (DCTCP). In *SIGCOMM*, pages 63–74, 2010.

[14] Ganesh Ananthanarayanan and Randy H. Katz. Greening the Switch. In *Proc. USENIX HotPower*, San Diego, CA, Dec 2008.

[15] Anders S. G. Andrae and Tomas Edler. On Global Electricity Usage of Communication Technology: Trends to 2030. *Challenges*, April 2015.

[16] Woongki Baek and Trishul M. Chilimbi. Green: A framework for supporting energy-conscious programming using controlled approximation. *SIGPLAN Not.*, 45(6):198–209, June 2010.

[17] M. Baldi and Y. Ofek. Time for a ”greener” internet. In *Communications Workshops, 2009. ICC Workshops 2009. IEEE International Conference on*, pages 1–6, June 2009.

[18] J. Baliga, R. Ayre, K. Hinton, and R. S. Tucker. Photonic switching and the energy bottleneck. In *Photonics in Switching, 2007*, pages 125–126, Aug 2007.

[19] Luiz Andre Barroso and Urs Holzle. The Case for Energy-Proportional Computing. In *Computer*. IEEE Computer Society, 2007.

[20] Theophilus Benson, Aditya Akella, and David A. Maltz. Network Traffic Characteristics of Data Centers in the Wild. In *IMC*, 2010.

[21] Kashif Bilal, Samee U. Khan, Joanna Kolodziej, Limim Zhang, Khizar Hayat, Sajjad A. Madani, Nasro Min-Allah, Lizhe Wang, and Dan Chen. A Comparative Study of Data Center Network Architectures. In *26th European Conference on Modeling and Simulation (ECMS)*, pages 526–532, Koblenz, Germany, 2012.

[22] J. Blackburn and K. Christensen. A simulation study of a new green bittorrent. In *Communications Workshops, 2009. ICC Workshops 2009. IEEE International Conference on*, pages 1–6, June 2009.

[23] Alessandro Carrega, Suresh Singh, Roberto Bruschi, and Raffaele Bolla. Traffic Merging for Energy-Efficient Datacenter Networks. In *International Symposium on Performance Evaluation of Computer and Telecommunication Systems*, 2012.

[24] J. Chabarek, S. Banerjee, P. Sharma, J. Mudigonda, and P. Barford. Networks of Tiny Switches (NoTS): In Search of Network Power Efficiency and Proportionality. In *Proceedings of the 5th Workshop on Energy-Efficient Design*, June 2013.

[25] J. Chabarek, J. Sommers, P. Barford, C. Estan, D. Tsiang, and S. Wright. Power awareness in network design and routing. In *INFOCOM 2008. The 27th Conference on Computer Communications. IEEE*, April 2008.

[26] Kai Chen, Ankit Singla, Atul Singh, Kishore Ramachandran, Lei Xu, Yueping Zhang, Xitao Wen, and Yan Chen. OSA: An Optical Switching Architecture for Data Center Networks with Unprecedented Flexibility. In *NSDI*, 2012.

[27] L. Chiaraviglio, M. Mellia, and F. Neri. Reducing power consumption in backbone networks. In *Communications, 2009. ICC '09. IEEE International Conference on*, pages 1 – 6, June 2009.

[28] Kenneth J. Christensen and Franklin 'Bo' Gulledge. Enabling power management for network-attached computers. *Int. J. Netw. Manag.*, 8(2):120–130, March 1998.

[29] Charles Clos. A Study of Non-Blocking Switching Networks. *The Bell System Technical Journal*, 32(2):406–424, March 1953.

[30] Andrew R. Curtis, Wonho Kim, and Praveen Yalagandula. Mahout: Low-overhead Datacenter Traffic Management using End-host-based Elephant Detection. In *INFOCOM*, 2011.

[31] Andrew R. Curtis, Jeffrey C. Mogl, Jean Tourrihes, Praveen Yalagandula, Puneet Sharma, and Sujata Banerjee. DevoFlow: Scaling Flow Management for High-Performance Networks. In *SIGCOMM*, pages 254–265, 2011.

[32] William James Dally and Brian Towles. *Principles and Practices of Interconnection Networks*. ELSEVIER, 2004.

[33] David Coudert and Alvinice Kodjo and Truong Khoa Phan. Robust energy-aware routing with redundancy elimination. *Computers and Operations Research*, 64:71–85, 2015. http://www.sciencedirect.com/science/article/pii/S0305054815001252.

[34] V. Eramo, A. Germoni, A. Cianfrani, E. Miucci, and M. Listanti. Comparison in Power Consumption of MVMC and BENES Optical Packet Switches. In *Proceedings IEEE NOC (Network on Chip)*, pages 125–128, 2011.

[35] Xiaobo Fan, Wolf dietrich Weber, and Luiz Andre Borroso. Power Provisioning for a Warehouse-sized Computer. In *ISCA*, 2007.

[36] Nathan Farrington, George Porter, Sivasankar Radhakrishnan, Hamid Hajabdolali Bazzaz, Vikram Subramaya, Yeshaiahu Fainman, George Papen, and Amin Vahdat. Helios: A Hybrid Electrical/Optical Switch Architecture for Modular Data Centers. In *SIGCOMM*, pages 339–350, 2010.

[37] Will Fisher, Martin Suchara, and Jennifer Rexford. Greening backbone networks: Reducing energy consumption by shutting off cables in bundled links. In *Proceedings of the First ACM SIGCOMM Workshop on Green Networking*, Green Networking '10, pages 29–34, New York, NY, USA, 2010. ACM.

[38] Albert Greenberg, James Hamilton, David A. Maltz, and Parveen Patel. The Cost of a Cloud: Research Problems in Data Center Networks. In *SIGCOMM CCR*, pages 68–73, 2009.

[39] Albert Greenberg, James R. Hamilton, and Navendu Jain. VL2: A Scalable and Flexible Data Center Network. In *SIGCOMM*, pages 51–62, 2009.

[40] C. Gunaratne, K. Christensen, and B. Nordman. Managing energy consumption costs in desktop PCs and LAN switches with proxying, split TCP connections, and scaling of link speed. *INTERNATIONAL JOURNAL OF NETWORK MANAGEMENT*, 15:297310, 2005.

[41] C. Gunaratne, K. Christensen, B. Nordman, and S. W. Yuen. Reducing the Energy Consumption of Ethernet with Adaptive Link Rate (ALR) . 57:448–461, April 2008.

[42] C. Gunaratne, K. Christensen, and S. W. Yuen. Ethernet Adaptive Link Rate

(ALR): Analysis of a Buffer Threshold Policy . In *GLOBECOME*, November 2006.

[43] Chaunxiong Guo, Haitao Wu, Kun Tan, Lei Shi, Yongguang Zhang, and Songwu Lu. DCell: A Scalable and Fault-tolerant Network Structure for Data Centers. In *SIGCOMM*, pages 75–86, 2008.

[44] Chuanxiong Guo, Guohan Lu, Dan Li, Haitao Wu, Xuan Zhang, Yunfeng Shi, Chen Tian, Yongguang Zhang, and Songwu Lu. BCube: A High Performance, Server-centric Network Architecture for Modular Data Centers. In *SIGCOMM*, pages 63–74, 2009.

[45] Deke Guo, Tao Chen, Dan Li, Yunhao Liu, and Guihai Chen. BCN: Expansible Network Structures for Data Centers Using Hierarchical Compound Graph. In *INFOCOM*, 2011.

[46] M. Gupta, S. Grover, and S. Singh. A feasibility study for power management in LAN switches. In *Proceedings of the 12th IEEE International Conference on Network Protocols*, page 361371, Washington, DC, 2004.

[47] M. Gupta and S. Singh. Using low-power modes for energy conservation in Ethernet LANs. In *IEEE INFOCOM*, page 24512455, 2007.

[48] Maruti Gupta and Suresh Singh. Greening of the Internet. In *Proceedings of ACM SIGCOMM*, 2003.

[49] László Gyarmati and Tuan Anh Trinh. Scafida: A scale-free network inspired data center architecture. *SIGCOMM Comput. Commun. Rev.*, 40(5):4–12, October 2010.

[50] R. Hays. Active/Idle Toggling with 0BASE-x for Energy Efficient Ethernet, November 2007. presentation to the IEEE802.3az Task Force.

[51] Brandon Heller, Srini Seetharaman, Priya Mahadevan, Yannis Yiakoumis, Puneet Sharma, Sujata Banerjee, and Nick McKeown. ElasticTree: Saving Energy in Data Center Networks. In *NSDI*, 2010.

[52] C. Hu, C. Wu, W. Xiong, B. Wang, J. Wu, and M. Jiang. On the design of green reconfigurable router toward energy efficient internet. *IEEE Communications Magazine*, 49(6):83–87, June 2011.

[53] Jeremy Blackburn and K Christensen. Green Telnet Modifying a client-server application to save energy. In *DR DOBBS JOURNAL*, volume 33, 2008.

[54] M. Jimeno, K. Christensen, and B. Nordman. A network connection proxy to enable hosts to sleep and save energy. In *Performance, Computing and Communications Conference, 2008. IPCCC 2008. IEEE International*, pages 101–110, Dec 2008.

[55] Aman Kansal and Feng Zhao. Fine-grained energy profiling for power-aware application design. *SIGMETRICS Perform. Eval. Rev.*, 36(2):26–31, August 2008.

[56] Changhoon Kim, Matthew Caesar, and Jennifer Rexford. Floodless in SEATTLE: A Scalable Ethernet Architecture for Large Enterprises. In *SIGCOMM*, pages 3–14, 2008.

[57] John Kim, William J. Dally, and Dennis Abts. Flattened Butterfly: A Cost-Efficient Topology for High-Radix Networks. In *ISCA*, pages 126–137, 2007.

[58] Jonathan G. Koomey. Growth in Data Center Electricity Use 2005 to 2011. Technical report, Stanford University, 2011.

[59] L. Irish and K. J. Christensen. A Green TCP/IP to Reduce Electricity Consumed by Computers. In *Proceedings IEEE Southeastern Engineering for a New Era*, 1998.

[60] Dan Li, Chuanxiong Guo, Haitao Wu, Kun Tan, Yongguang Zhang, and Songwu Lu. FiConn: Using Backup Port for Server Interconnection in Data Centers. In *INFOCOM*, 2009.

[61] Yong Liao, Dong Yin, and Lixin Gao. DPillar: Scalable Dual-Port Server Interconnection for Data Center Networks. In *Proceeding of 26th International Conference on Computer Communication Networks*, 2010.

[62] Gongqi Lin, Sieteng Soh, and Kwan-Wu Chin. Energy-aware traffic engineering with reliability constraint. *Computer Communications*, 57:115–128, 2015.

[63] Minghong Lin, Adam Wierman, Lachlan L. H. Andrew, and Eno Thereska. Dynamic Right-sizing for Power-proportional Data Centers. In *INFOCOM*, 2011.

[64] Priya Mahadevan, Puneet Sharma, Sujata Banerjee, and Parthasarathy Ranganathan. A power benchmarking framework for network devices. In *Proceedings of the 8th International IFIP-TC 6 Networking Conference*, NETWORKING '09, pages 795–808, Berlin, Heidelberg, 2009. Springer-Verlag.

[65] M. Mandviwalla and Nian-Feng Tzeng. Energy-efficient scheme for multiprocessor-based router linecards. In *Applications and the Internet, 2006. SAINT 2006. International Symposium on*, pages 8–163, Jan 2006.

[66] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parylkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonanthan Turner. OpenFlow: Enabling Innovation in Campus Networks. In *SIGCOMM CCR*, pages 69–74, 2008.

[67] D. Meisner, B. T. Gold, and T. F. Wenisch. PowerNap: Eliminating Server Idle Power. In *Proceeding of the 14th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS 2009)*, pages 205–216, March 2009.

[68] Jayaram Mudigonda, Praveen Yalagandula, Muhammad Al-Fares, and Jeffrey C. Mogul. SPAIN: COTS Data-Center Ethernet for Multipathing over Arbitrary Topologies. In *NSDI*, 2010.

[69] Radhika Niranjan Mysore, Andreas Pamboris, Nathan Farrington, Nelson Huang, Pardis Miri, Sivasankar Radhakrishnan, Vikram Subramnya, and Amin Vahdat. PortLand: A Scalable Fault-Tolerant Layer 2 Data Center Network Fabric. In *SIGCOMM*, pages 39–50, 2009.

[70] Sergiu Nedevschi, Lucian Popa, Gianluca Iannaccone, Sylvia Ratnasamy, and David Wetherall. Reducing Network Energy Consumption via Sleeping and Rate-Adaptation. In *NSDI*, 2008.

[71] Costin Raiciu, Sebastien Barre, Christopher Pluntke, Adam Greenhalgh, Damon Wischik, and Mark Handley. Improving Datacenter Performance and Robustness with Multipath TCP. In *SIGCOMM*, pages 266–277, 2011.

[72] Forrester Research. Power and Cooling Heat Up the Data Center. September 2011.

[73] Brunilde Sanso and Hakim Mellah. On Reliability, Performance and Internet Power Consumptiong. In *Proceedings of 7th International Workshop on the Design of Reliable Communication Networks*, Oct 2009.

[74] Ji-Yong Shin, Bernard Wong, and Emin Gün Sirer. Small-world datacenters. In *Proceedings of the 2Nd ACM Symposium on Cloud Computing*, SOCC '11, pages 2:1–2:13, New York, NY, USA, 2011. ACM.

[75] Suresh Singh and Candy Yiu. Putting the Cart Before the Horse: Merging Traffic for Energy Conservation. In *IEEE Communications Magazine*. June 2011.

[76] Ankit Singla, Chi-Yao Hong, Lucian Popa, and P. Brighten Godfrey. Jellyfish: Networking Data Centers Randomly. In *NSDI*, 2012.

[77] T. Song, W. Fu, O. Ormond, M. Collier, and X. Wang. Energy evaluation of gigabit routers towards energy efficient network. In *Local Metropolitan Area Networks (LANMAN), 2014 IEEE 20th International Workshop on*, pages 1–5, May 2014.

[78] Tian Song, Xiangjun Shi, and Xiaowei Ma. Fine-grained power scaling algorithms for energy efficient routers. In *Proceedings of the Tenth ACM/IEEE Symposium on Architectures for Networking and Communications Systems*, ANCS '14, pages 197–206, New York, NY, USA, 2014. ACM.

[79] Theophilus Benson and Ashok Anand and Aditya Akella and Ming Zhang. Understanding Data Center Traffic Characteristics. In *WREN*, 2009.

[80] Guohui Wang, David G. Andersen, Michael Kaminsky, Konstantina Papagiannaki, T. S. Eugene Ng, Michael Kozuch, and Michael Ryan. c-Through: Part-time Optics in Data Centers. In *SIGCOMM*, pages 327–338, 2010.

[81] Xiaodong Wang, Yanjun Yan, Xiaorui Wang, Kefa Lu, and Qing Cao. CARPO: Correlation-aware Power Optimization in Data Center Networks. In *INFOCOM*, pages 1125–1133, 2012.

[82] Indra Widiaja, Anwar Walid, Yanbin Luo, Yang Xu, and H. Jonathan Chao. Switch Sizing for Energy Efficient Datacenter Networks. In *Proceedings Green-Metrics 2013 Workshop (in conjunction with ACM Sigmetrics 2013)*, Pittsburgh, PA, June 2013.

[83] A. Wierman, L. L. H. Andrew, and A. Tang. Power-Aware Speed Scaling in Processor Sharing Systems. In *Proceedings of the 28th Annual IEEE Conference on Computer Communications. (INFOCOM 2009)*, April 2009.

[84] Christo Wilson, Hitesh Ballani, Thomas Karagiannis, and Ant Rowstron. Better Never than Late: Meeting Deadlines in Datacenter Networks. In *SIGCOMM*, pages 50–61, 2011.

[85] Candy Yiu and Suresh Singh. Merging Traffic to Save Energy in the Enterprise. In *E-Energy*, 2011.