

RICE UNIVERSITY

On Designing Convertible Data Center
Network Architectures

by

Yiting Xia

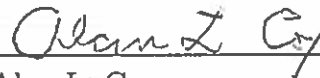
A THESIS SUBMITTED
IN PARTIAL FULFILLMENT OF THE
REQUIREMENTS FOR THE DEGREE

Doctor of Philosophy

APPROVED, THESIS COMMITTEE:



T.S. Eugene Ng, Chair
Professor of Computer Science and
Electrical and Computer Engineering



Alan L. Cox
Professor of Computer Science and
Electrical and Computer Engineering



Edward W. Knightly
Professor of Electrical and Computer
Engineering and Computer Science

Houston, Texas

April, 2018

RICE UNIVERSITY

**On Designing Convertible Data Center
Network Architectures**

by

Yiting Xia

A THESIS SUBMITTED
IN PARTIAL FULFILLMENT OF THE
REQUIREMENTS FOR THE DEGREE

Doctor of Philosophy

APPROVED, THESIS COMMITTEE:

T. S. Eugene Ng, Chair
Professor of Computer Science and
Electrical and Computer Engineering

Alan L. Cox
Professor of Computer Science and
Electrical and Computer Engineering

Edward W. Knightly
Professor of Electrical and Computer
Engineering and Computer Science

Houston, Texas

April, 2018

Abstract

Most data centers deploy fixed network topologies. This brings difficulties to traffic optimization and network management, because bandwidth locked up in fixed links is not adjustable to traffic needs, and changes of network equipments require cumbersome rewiring of existing links. Moreover, each network topology has unique properties, so it is infeasible to use a one-size-fit-all structure to satisfy the heterogeneous and ever-changing service requirements in data centers. We believe the solution is to build convertible data center network architectures, which can dynamically change the network topology through cable rewiring fully automated by software. We leverage low-cost small port-count converter switches to enable topology change and introduce three convertible data center architectures to experiment with the idea.

OmniSwitch is a production-ready modular container that serves as a universal building block for constructing data centers of various scales. It interleaves converter switches with Ethernet switches to provide local topology optimization and large-scale connectivity. Flat-tree improves transmission performance by dynamically changing topological clustering characteristics of the network. It enables conversion between Clos and approximate random graph networks to provide a suitable network topology for each traffic workload. ShareBackup improves reliability with the concept of “shareable backup”. It allows the network to share a small pool of backup switches that can be brought online instantaneously to recover from failures. These works demonstrate the powerful idea of convertible networks, which has the potential to improve a wide range of network performance characteristics, e.g. traffic optimization, load balancing, failure recovery, network expansion, power saving etc.

Acknowledgements

Above all, I would like to thank my advisor, Prof. T. S. Eugene Ng. This thesis would not have been possible without his inspiration, guidance, help, and support. I am grateful for his mentorship both on the academic and the personal level. I learnt a lot from his knowledge, enthusiasm, and integrity as a computer scientist. I would also like to thank Prof. Alan Cox and Prof. Edward Knightly for being on my committee and providing me with valuable feedbacks for the thesis. I want to thank Dr. Mike Schlansker and Dr. Jean Tourrilhes, our major collaborators for the OmniSwitch project. They proposed the initial idea of configurable Clos network using small port-count circuit switches, which is the basis of the concept of convertible network. My work would not have existed without their insights. I'm also grateful to my fellow group members, Xiaoye Steven Sun, Simbarashe Dzinamarira, Xin Sunny Huang, and Dingming Wu, who helped me with some of the experiments and gave me many constructive suggestions. Last but not least, my thanks go to my parents and all family and friends, who always believe in me and support me in whatever situations.

Contents

List of Illustrations	vii
List of Tables	xi
1 Introduction	1
2 Background and Related Work	10
2.1 Fixed Data Center Network Architectures	10
2.2 Configurable Data Center Network Architectures	11
2.3 Failure Recovery in Data Center Networks	12
2.4 The Concept of Convertible Network	13
2.5 Realization of Convertible Networks	15
2.6 Implementation Criteria for Convertible Networks	17
3 OmniSwitch: A Universal Switch as the Basic Building Block for Data Centers	19
3.1 Motivation	19
3.2 OmniSwitch Design	23
3.2.1 Architecture	23
3.2.2 Advantage Discussion	26
3.2.3 Control Plane	27
3.3 VM Clustering: A Case Study	29
3.3.1 Problem Formulation	29
3.3.2 Control Algorithm	30
3.3.3 Evaluation	31

3.4	Summary	35
4	Flat-tree: A Convertible Data Center Architecture from Clos to Random Graph	36
4.1	Motivation	36
4.2	Motivating Examples	41
4.2.1	The Case for Convertibility	41
4.2.2	Example Flat-tree Network	43
4.3	Flat-tree Architecture	45
4.3.1	Flat-tree Pod	45
4.3.2	Pod-Core Wiring	47
4.3.3	Inter-Pod Wiring	49
4.3.4	Server Distribution	50
4.3.5	Operation Modes	51
4.4	Control System	52
4.4.1	MPTCP	53
4.4.2	k -Shortest-Paths Routing	54
4.4.3	Topology Conversion	59
4.5	Evaluation	61
4.5.1	How well does flat-tree approximate random graph in theory?	61
4.5.2	Can multiple topologies coexist in flat-tree?	66
4.5.3	Is k -shortest-paths routing with MPTCP efficient enough?	66
4.5.4	Does flat-tree handle real traffic well?	70
4.5.5	Is flat-tree implementable?	75
4.5.6	Does convertibility benefit applications?	77
4.6	Summary	81
5	ShareBackup: Enabling Sharable Backup for Failure Re-	

covery in Data Center Networks	82
5.1 Motivation	82
5.2 Network Architecture	86
5.3 Control Plane	94
5.3.1 Fast Failure Detection and Recovery	94
5.3.2 Distributed Network Controllers	95
5.3.3 Offline Failure Diagnosis	97
5.3.4 Live Impersonation of Failed Switch	99
5.4 Discussion	100
5.5 Architecture Properties	103
5.5.1 Capacity to Handle Failures	103
5.5.2 Cost Analysis	104
5.5.3 Performance Characteristics	107
5.6 Summary	109
6 Future Work	111
6.1 Joint Optimization of Network Topology and Traffic	111
6.2 Combining Flat-tree and ShareBackup	113
6.3 Convertibility for Power Saving	114
7 Conclusion	116
Bibliography	122

Illustrations

3.1	An example data center network. The transmission hop count between VM 1 and VM 2 is originally 5. Moving the bold link to the dashed position reduces the hop count to 3.	21
3.2	Internal of an OmniSwitch cabinet	23
3.3	OmniSwitch mesh network. The right subfigures show topologies of the Ethernet switches. Switch 1, 2, 3, 4 are in one cabinet; switch 5, 6, 7, 8 are in another cabinet. Each line represents 4 individual fibers. Solid lines are connections within a cabinet; dashed lines are connections across cabinets.	24
3.4	OmniSwitch tree network. In subfigure (b), the 2 uplinks out of each cabinet refer to the 4 dark and light multilink connections in subfigure (a) respectively.	25
3.5	Average bandwidth rejection rate under different load	32
3.6	Algorithm computation time for various tenant size and load	33
4.1	Converter switch configurations	38

4.2	Example flat-tree network and some achievable topologies. Core switches in stripe, aggregation switches in grid, edge switches in shade, and servers as circles. Gray lines are connections in the original Clos network, which are replaced with the dashed links connected to converter switches to form flat-tree. The converter switches show the configuration for approximated random graph. Flat-tree uses a customized wiring pattern to connect Pods to core switches.	43
4.3	A flat-tree Pod. A pair of edge switch E_j and aggregation switch $A_{j/r}$ connected to n 4-port converter switches and m 6-port converter switches. Converter switches are placed evenly on both sides as matrices. Blade A and B has 4-port and 6-port converter switches respectively.	46
4.4	Pod-Core wiring for the same set of connectors across Pods. All connectors are on aggregation switches in Clos; flat-tree has 3 types of connectors on blade A, B, and aggregation switches, enabling core-server, core-edge, and core-aggregation connections respectively.	47
4.5	Illustration of the addressing scheme. “a” shows the IP address fields in flat-tree. In the “b” example, the server in strip connects to switch #3, switch #8, and switch #5 respectively in the global, local, and Clos mode, where the number of concurrent paths, or k , is chosen to be 16, 8, and 4. The IP addresses assigned to this server are shown in “c”. All these addresses for every flat-tree topology mode are preconfigured on the server.	55
4.6	Average path length of server pairs in the entire network	61
4.7	Average path length of server pairs in each Pod	62
4.8	Throughput of broadcast/incast traffic in 1000-server clusters	64
4.9	Throughput of all-to-all traffic in 20-server clusters	65

4.10	Average flow throughput normalized against LP minimum on selected flat-tree topologies	68
4.11	Box plots to show the distribution of flow throughput on the topo-1 topology under flat-tree global mode (topo-1 global). MPTCP uses 12 paths. The box contains the 25th to 75th percentiles of the data. The whisker lines extending above and below the box cover the data within 3 times the box range. The data in dots beyond the whisker are outliers. The bold line in the middle of the box shows the median and the diamond shows the average.	70
4.12	CDF of flow completion time in Facebook’s Hadoop-1, Hadoop-2, Web, and Cache data centers	72
4.13	A testbed implementing the flat-tree example in Figure 4.2	75
4.14	Summation of iperf throughput every 0.5 second on the testbed with the variation of flat-tree modes. Every server sends iperf traffic to its counterparts in the other Pods to saturate the network core. Traffic adapts to the topology change in 2 to 2.5 seconds.	78
4.15	Average data flow read duration (left y-axis) and average communication phase duration (right y-axis) in the Spark broadcast and Hadoop shuffle applications under different flat-tree topology modes	79
5.1	Impact of failures on flows and coflows	83
5.2	A $k = 6$ fat-tree [1]. To build a ShareBackup network from it, the blocks of devices like in the shaded areas should be replaced by the corresponding structures in Figure 5.3.	90

5.3 Substructures of a ShareBackup network where $k = 6$ and $n = 1$. The subfigures correspond to the shaded areas in Figure 5.2. Devices are labeled according to the notations in Table 5.1. Edge and aggregation switches are marked by their in-Pod indices; core switches and hosts are marked by their global indices. Switches in the same failure group are packed together, which share a backup switch in stripe on the side. Circuit switches are inserted into adjacent layers of switches/hosts. The connectivity in shade is the basic building block for shareable backup. The crossed switch and connections represent example node and link failures. Switches involved in failures are each replaced by a backup switch with the new circuit switch configurations shown at the bottom, where connections regarding the original red round ports reconnect to the new black square ports. . . . 92

5.4 Communication protocol in the control system. (a): Failure detection and recovery. (b): Diagnosis of link failure. 95

5.5 Circuit switch configurations for diagnosis of link failures shown by examples (b) and (c) in Figure 5.3. Circuit switches in a Pod are chained up using the side ports. Only “suspect switches” on both sides of the failed link and some related backup switches are shown. Through configurations ①, ②, and ③, the “suspect interface” on both “suspect switches” associated with the failure can connect to 3 different interfaces on one or multiple other switches. 97

5.6 Additional cost of ShareBackup, Aspen Tree, and 1:1 Backup relative to fat-tree at different network scales using market prices in Table 5.2 106

Tables

3.1	Average hop count when $load = 0.8$	31
4.1	Throughput of clustered traffic normalized against the minimum value in the compared architectures	41
4.2	List of flat-tree topologies for evaluating the control system. Abbreviations: Edge Switch (ES), Aggregation Switch (AS), Core Switch (CS), Upstream Port (UP), Downstream Port (DP), Oversubscription Ratio (OR).	67
4.3	Conversion delay of the throughput experiment in Figure 4.14	77
5.1	List of notations	86
5.2	Cost of compared architectures, where the data center uses electrical (E-DC) and optical (O-DC) transmissions respectively.	105
5.3	Performance characteristics of different network architectures	107

Chapter 1

Introduction

In this thesis, we appeal for rethinking the design of data center network architectures by introducing the concept of convertibility. We define convertibility as a network's ability to change its topology dynamically. This change should be completely managed by software, without involving human labor for rewiring the physical devices. With the power of convertibility, it is possible for the first time to build a data center that can function with different network architectures to combine the benefits of conventionally incompatible worlds. Our proposal is rooted in the recent trends in the development of data center networks.

The first trend is the continuous efforts towards two conflicting goals for data center network design: low implementation complexity vs. high transmission throughput. These efforts are reflected in the enthusiasm for Clos networks in industry and random graph networks in academia. Clos, or multi-rooted tree, is the *de-facto* standard data center network architecture because of its highly organized structure [2, 3]. Figure 4.2b shows an example Clos network. The central wiring between switches in adjacent layers is relatively easy to manage, and the network can be expanded to arbitrary size by adding stages. Bandwidth oversubscription can occur at any switch layer to save cost. Modular Pods are adopted as building blocks to further ease network deployment and management. However, Clos networks have suboptimal throughput, as traffic needs to traverse up and down the network hierarchy and the resulting inefficiency exacerbates oversubscription.

In contrast, random graphs are proven to have optimal throughput [4,5]. Without rigid structures, switches are more directly connected at shorter path lengths. If implemented using the same switches and servers as a Clos network, a random graph can provide richer bandwidth and effectively alleviate the oversubscription problem. The “fluid” structure also enables constructing random graphs at different scales to serve heterogeneous workloads in data centers [6–10], e.g. a network-wide random graph to serve large clusters and regional random graphs as part of the network to serve small clusters. Yet, the neighbor-to-neighbor wiring between random switch pairs is disorderly, making real-world implementation a daunting task.

Closely related to these conflicting stances is the second trend of stagnation in the emergence of new data center network architectures. Back in 2008 and 2009, the research community proposed a number of interconnection networks as the data center fabric, fat-tree [1], DCell [11], BCube [12], and HyperX [13] being the famous examples. However, there has been no breakthrough ever since. In the design space, these architectures fall between Clos and random graph at the extremes of the scale. They attempt to find the right middle ground between orderly implementation and good performance by tuning the degree of hierarchical vs. flat structure, central vs. neighbor-to-neighbor wiring, etc. Yet, the transmission performance of a network is traffic specific, thus each topology has the sweet spot for particular workloads [4]. It is hard to use a one-size-fit-all topology to address the heterogeneous and ever-changing service needs in data centers.

To combat the limitation of fixed network topologies, the third trend is optimization at various layers of the network stack to make better use of the network. The rich set of work include routing and transport protocols [6, 14–18], flow scheduling mechanisms [19–25], workload placement heuristics [26–29], etc. An extreme example

is Google building well-customized data centers with specialized hardware, network architecture, control system, computation framework, resource allocation scheme, etc [3, 30]. Despite the success of these efforts, some fundamental traits of applications are unchangeable. For example, even with proper optimizations of workloads and transmission performance, Facebook still observes very different traffic characteristics in its service clusters [10], indicating different stress points are placed on the network.

Relaxing the constraint of fixed topologies, the fourth trend is the advent of configurable data center networks that create ad-hoc links as needed. Some solutions provide a local remedy for fixed topologies by adding a small number of connections to alleviate hot spots [31–36], while others create a flexible network core for small-scale networks [37–41]. On one hand, these works demonstrate it is technically mature to change the network topology by software at runtime. On the other hand, the flexibility of network topology has potential to be extended to a wider range of the network and to larger-scale networks.

Based on the above evidence, we make the bold claim that it is time to promote convertible data center network architectures. Convertibility elevates link configurability to a higher level. It allows for global topology change in a data center network of any scale.

Convertibility is achievable by converter switches, which can pipe traffic point to point with no bandwidth contention from the input ports to the output ports using different permutations. By changing the configurations of the converter switches, cables are rewired to different outgoing connections, as if they were unplugged and replugged manually. We leverage small port-count converter switches to reduce the cost. If implemented using packet switches, the bare-minimum switching functionality

does not require expensive processor/buffering, sophisticated routing protocols, or general-purpose OS, etc. Cheap switching chips in small scale with support for simple port-to-port forwarding rules would suffice. If implemented using circuit switches, low-cost switching technologies with modest port count limited by signal losses can also apply [42, 43].

The benefits of convertibility is multi-fold. First, convertibility provides another dimension of flexibility to traffic optimization. Static network topologies lock up bandwidth in fixed links, so congested links cannot get more bandwidth even if it exists in the network. In a convertible network, however, bandwidth can be moved to transmission hot spots as needed. Convertibility can enhance traffic locality and reduce transmission hop count. Servers that exchange the most traffic can be relocated to a common switch to minimize the traffic sent to higher layers in the network hierarchy. Opposite to traffic locality, load balancing and failure resilience can be achieved by directing traffic relevant to the same service to different switches.

Second, convertibility also simplifies deployment, upgrade, and management for complex data center networks. Constructing data centers requires complex wiring, and cable rewiring for later changes is especially challenging. If cables are interconnected through converter switches, detailed rewiring after the initial deployment can be maintained automatically by cable management software. This introduces opportunities for dynamic topology optimization in case of a hardware failure, during a switch firmware upgrade, or after partial power-down during off-hour operation.

Third, convertibility enables incremental expansion of data centers. A data center can be partially populated with switches and servers by disconnecting a subset of links via converter switches. As more equipments are activated in the network, the converter switch settings are configured to adapt to the change, avoiding manual

rewiring of existing links.

Fourth, convertibility makes efficient backup possible. Most data centers deploy one-on-one backup for each Ethernet switch for fault tolerance. 1 out of N backup can be achieved with configurable topology. A single spare switch connected to multiple switches through converter switches can be brought online as needed to replace any switch that fails. This enables more efficient backup and reduces the cost for redundancy significantly.

This thesis is that it is economical and feasible to build data center networks with topological convertibility to improve transmission performance and fault tolerance. In this thesis, we propose three convertible data center network architectures. OmniSwitch is the first effort towards the exploration of convertibility. We design a production-ready modular container that use interleaving converter switches and Ethernet switches to provide both local convertibility within the container and large-scale connectivity. As universal building blocks of data centers, a number of OmniSwitch containers can be interconnected to form data center networks of different scales using different topologies, such as mesh and tree networks. We consider practical issues in this product design, including directing traffic by converter switches to desirable Ethernet switches within the container for traffic optimization, exposing multi-link connectors external to the container for easy wiring, allowing partial population of devices and enforcing modular design for incremental expansion, equipping a spare Ethernet switch shareable to a set of active ones for efficient backup, and employing small port-count converter switches instead of large ones for cost effectiveness. We demonstrate the potential of convertibility with the VM clustering case study. Simulations using a real data center workload show that compared to the state-of-the-art solutions, OmniSwitch reduces the average path length significantly and services more

bandwidth using minimal computation time. Small converter switches are proven to provide similar convertibility to a high port-count counterpart.

Flat-tree is a more in-depth study of convertibility. Unlike OmniSwitch that tunes the network topology locally, it aims at converting the entire network between the Clos topology and approximate random graph of various scales to achieve the conventionally conflicting goals of low implementation complexity and high transmission throughput. We flatten Clos’ tree structure by rewiring existing connections via converter switches, which have low cost and can be packaged into Pods to ease deployment. With regular wiring patterns between Pods and core switches and simple connections between adjacent Pods, we effectively approximate randomness in the network core and at the same time obtain low wiring complexity. Multi-path routing and congestion control are crucial to exploiting the path diversity in flat-tree, and we have shown that aggregation strategies can be applied to avoid an explosion of network states. Existing routing and transport protocols combined with our architecture-specific state aggregation schemes can balance between high network utilization and fair bandwidth sharing among flows. We explore the implementability of flat-tree using simulations with real data center traffic and a testbed implementation of the system. Flat-tree has similar average path length as random graphs and the traffic throughput is indistinguishable. We also observe flat-tree can optimize for diverse workloads with different topology options, and it brings performance improvements to applications with greater core bandwidth.

ShareBackup explores how convertibility can be used to enhance fault tolerance in the network. A measurement study has shown that failures are rare but disruptive in production data centers [44]. Moreover, we find the effect of failures is magnified hugely on the application level by our own experiments: under failures,

the number of impacted Coflows is significantly greater than the number of impacted individual flows. As a result, we should recover from failures immediately after they happen. OmniSwitch adds a spare switch as backup in the local container, while ShareBackup further develops the idea of efficient backup by creating a small pool of backup switches that can be shared by the entire network. ShareBackup is based on the fat-tree architecture. We organize switches into failure groups and allow them to share one or more backup switches. Switches in the same failure group, as well as the backup switches, are connected to the same set of converter switches, so that they can be replaced by the backup switches when failed. Link failures are addressed as node failures on both ends, and we use offline failure diagnosis to understand the cause of problem and to recycle healthy switches. We use distributed network controllers to share the burden of failure detection and recovery. For fast failure recovery, we support live impersonation of the failed switches on the control plane. Using market prices, the cost of ShareBackup is multi-fold lower than state-of-the-art failure-resilient architectures. It also provides more bandwidth compared to rerouting-based solutions.

The contributions of this thesis are as follows:

- We introduce the concept of “convertibility” as a new angle to the design of data center networks and propose to realize this idea by distributed placement of cost-effective small circuit switches. Compared to previous works of using a large central circuit switch to add ad-hoc links at runtime, we rearrange the structure of the network to provide greater average performance characteristics throughout the life cycle of the workload. The per-port cost of our targeting switching technology is significantly lower, and the scalability of the network is not limited by the port count of the central circuit switch. This new design

philosophy may motivate other novel architectures of data center networks.

- We design three data center network architectures as different use cases of convertibility. OmniSwitch is a rack-scale computing container. It explores the potential of local traffic optimization and efficient backup with convertibility. Flat-tree and ShareBackup extend these ideas to the scope of the entire network. Flat-tree enables network-wide conversion between Clos and approximate random graph networks to provide a suitable network topology for each traffic workload. ShareBackup allows the network to share a small pool of backup switches for fast failure recovery. We prove the power of convertibility through these architectures and discuss other promising applications, such as load balancing, network expansion, power saving etc.
- We provide complete design of the network architectures and the control systems to give sights for general principles of convertible networks. We give wiring plans of the network devices to serve the specific design purposes. We propose resource scheduling algorithms to convert the network when necessary and devise customized routing protocols to transmit data on the flexible network topology. We also consider practical problems in real-world deployment, such as host transparency, packaging, wiring, etc. We find the cross-layer optimization very effective, and our experience is helpful for follow-up works on the design of convertible networks.
- We conduct extensive evaluations of the proposed network architectures to demonstrate advantages of convertible networks, including theoretical analyses, flow-level simulations, packet-level simulations, and testbed implementations. We evaluate effectiveness of different aspects of the designs and compare

the performance with state-of-the-art solutions. Specifically, compared to VM placement schemes, OmniSwitch provisions more cloud virtual clusters, and its only takes 0.1% computation time; flat-tree increases bandwidth through topology conversion to fit different workloads, which translates to reduction of end-to-end data read time in Hadoop and Spark applications; ShareBackup restores bandwidth immediately after failures at orders of magnitude lower additional cost than other redundancy-featured data center architectures.

The rest of the thesis is organized as follows. Chapter 2 introduces the background and related work of convertible networks. Chapter 3, 4, and 5 explain the detailed architecture and system designs of OmniSwitch, flat-tree, and ShareBackup respectively. Chapter 6 presents future work, and Chapter 7 concludes the thesis.

Chapter 2

Background and Related Work

2.1 Fixed Data Center Network Architectures

A number of interconnections networks have been proposed as the fabric for data center networks. Fat-tree suggests moving from a traditional hierarchical data center design utilizing expensive specialized core switches to a network built of commercial Ethernet switches which nevertheless achieved high throughput [1]. Because the port counts of commercial switches are usually limited, Fat-tree forms non-blocking 3-layer folded Clos networks, which use a large number of parallel links to provide network-wide connectivity and full aggregation bandwidth. DCell is a server-centric modular data center network architecture [11], where servers can be interconnected to relay traffic. It is a recursively defined structure, in which a high-level DCell is constructed from many low-level DCells and DCells at the same level are fully connected with one another. BCube is also a server-centric network structure [12]. It has multiple layers of switching units, and the servers are connected to each of the layers with one port. The BCube topology shows good characteristics for one-to-one, one-to-several, and one-to-all traffic patterns. HyperX is an architecture extended from hypercube and flattened butterfly, two important topologies in high performance computing networks [13]. HyperX is a class of multi-dimensional networks using high-radix switches. In a HyperX, each switch is connected to all of its peers in each dimension and the number of switches in each dimension can be different. CamCube builds

a shipping container sized data center [45]. It replaces the traditional switch-based network with a 3D torus topology, with each server directly connected to 6 other servers. Unlike these structured networks, Jellyfish realizes degree-bounded random graph [5]. The switches are randomly connected to each other as long as there are available ports, and servers are uniformly connected to the switches.

2.2 Configurable Data Center Network Architectures

Our work is also related to the recent proposals of configurable data center network architectures. One group of works creates ad-hoc links at runtime to alleviate hot spots [31,32,34,35,46,47]. Helios and c-Through construct a separate optical network with an expensive high port-count 3D MEMS side by side with the existing data center to add core bandwidth on the fly [48,49]. This idea is extended to different traffic patterns other than point-to-point communication [46,47]. Flyways exploits 60 GHz wireless technology to augment the traditional data center network with dynamic wireless links between directional antennas on ToR switches [34]. To reduce interference, a follow-up work uses 3D beamforming to bounce 60 GHz signals off to data center ceilings, thus establishing indirect line-of-sight between any two racks in a data center [35].

Another group constructs an all-connected flexible network core with high bandwidth capacity [33,36–41]. OSA builds an all-optical network by introducing WDM and WSS technologies to provide multi-hop forwarding and tunable link capacity [50]. Mordia, Quartz, and Plexxi use fast optical circuit switches (WSS or WDM rings) to build a full-mesh aggregation layer that has high bandwidth capacity and low switching latency [39,40]. Because WSS and WDM rings scale poorly, these designs work best for small networks with tens of aggregation ports. FireFly and ProjecToR pro-

vide free-space optics solutions, which use modulated visible or infrared laser beams transmitted through free space [33, 36]. FireFly equips ToR switches with antennas and they shoot signals onto ceiling mirrors to avoid obstruction, whereas ProjecToR uses a DMD and mirror assembly combination as a transmitter and photodetector on each rack. WaveCube builds a 2D-torus network with optical components working as joint nodes [38], in which traffic is usually relayed by many intermediate hops.

However, these solutions are constrained by the port count of central switches when enabling configurability [31, 32, 37], the number of optical wavelengths that can be reused [37–41], or the interference and attenuation of wireless signals [33–36]. Due to these scalability concerns, only a small number of connections can be added as a local remedy or the size of the network is limited to a small scale. Our work is the first to realize globally convertible data center networks at large scale.

2.3 Failure Recovery in Data Center Networks

Many architectural solutions have been proposed to improve failure resiliency of data center networks. Fat-tree [1], DCell [11], BCube [12], VL2 [51], HyperX [13], and Jellyfish [5] build high-performance data center network architectures with redundant paths and provide customized rerouting schemes to bypass failures. PortLand enhances fault tolerance of fat-tree with a layer-2 routing and forwarding protocol [52]. F10 adjusts wiring of fat-tree to diversify alternative paths in the network structure [53]. It also improves responsiveness to failures by fast failure detection and local rerouting to longer paths. Aspen Tree adds redundancy to fat-tree to reduce failure convergence time, at the price of partitioning the network or introducing extra hardware [54].

Other architecture-transparent solutions overcome the routing disruption problem

in ISP networks, including IP fast reroute [55–58], and multipath routing [59–61]. Failure carrying packets (FCP) eliminates the convergence process after a failure by allowing data packets to carry failure information [62]. Packet Recycling is a contingent forwarding technique with small packet overhead that takes advantage of cellular graph embedding for fast packet rerouting in the event of link failures [63]. R-BGP pre-computes a few strategically chosen failover paths and provably guarantees that a domain will not become disconnected from any destination as long as it will have a policy-compliant path to that destination after convergence [64]. Data-Driven Connectivity (DDC) addresses connectivity issues separately from the more far-reaching distributed computations of the control plane and provides ideal forwarding-connectivity [65]. Keep Forwarding (KF) is a labeling-free local failure resilient routing framework that provides effective failure resilience for the general k -link failure case [66]. DF-EDST resilience was introduced to use edge-disjoint spanning trees to provide deadlock-free local fast failover [67]. Plinko explores the feasibility of implementing local fast failover groups in hardware by conducting forwarding table compression [68]. In the OpenFlow network, Schiff *et al.* have introduced a number of useful functions that rely on hardware fast failover group [69]. Borokhovich *et al.* describe an OpenFlow fast failover algorithm that guarantees delivery without looping packets by treating failover as a maze traversal problem [70]. FatTire introduces a language for specifying fault tolerance requirements in the SDN paradigm [71].

2.4 The Concept of Convertible Network

Convertibility is a network’s ability to change its topology dynamically. This change should be completely managed by software, without involving human labor for rewiring the physical devices. A convertible network is able to change between multiple net-

work topologies when necessary, thus it is possible for the first time to build a data center that can function with different network architectures to combine the benefits of conventionally incompatible worlds.

Convertible networks are distinguished from the fixed data center network architectures by its topological flexibility. Each of these fixed topologies has sweet spots for particular traffic patterns [4], whereas convertible architectures are able to change the network topology to adapt to different workloads. For instance, OmniSwitch can dynamically direct traffic to different Ethernet switches to reduce the hop count of transmission, and flat-tree has multiple topology options that are suitable for different traffic patterns respectively. These fixed architectures have varying implementation complexity and performance properties. With the power of convertibility, flat-tree can be implemented more easily as a Clos network and has better performance as random graph networks. OmniSwitch successfully addresses many practical issues, i.e. traffic optimization, easy wiring, efficient backup, and incremental expansion, thanks to convertibility.

The concept of convertible network is fundamentally different from existing proposals of configurable data center network architectures. First, it aims to achieve network-wide topology change in large-scale data centers. The scalability of many previous works is constrained by a centralized device that enables flexibility, such as 3D MEMS [31, 32, 37, 46, 47] and WDM ring [39–41]. To overcome this weakness, in our proposal the enabling devices are placed across the network in a decentralized manner. Second, instead of adding extra bandwidth to the network, a convertible network rearranges the network structure to utilize existing bandwidth resources more efficiently. Third, rather than incremental topology evolution according to the instantaneous traffic pattern, a convertible network changes the intrinsic characteristics of

the topology to fit the requirements of different workloads throughout their lifecycle.

Convertible networks can also improve reliability of data center networks. The weaknesses of the above rerouting-based solutions on fault tolerance motivate the use of convertible networks. First, rather than bypassing failures at compromised performance, we should replace failed devices completely to restore bandwidth. Second, redundancy may cause excessive hardware expenses [54], while we can enable shareable backup with convertibility to save cost. Third, alternative paths can have more hops [5, 11, 13, 53] and path re-computation may be expensive [5], whereas a convertible network can maintain original paths after failures to avoid rerouting overhead and path dilation. Fourth, some solutions experience slow failure propagation [1], while convertible networks can replace failed devices instantly and thus localize the effect of failures.

As aforementioned, the network topology should be converted infrequently to avoid disruptions of the network. OmniSwitch converts topology when cloud virtual tenants come and go; flat-tree converts topology when the network operator deploys different applications; and ShareBackup converts topology when failures occur. Because the network topology remains relatively stable between conversions, existing traffic optimization and failure management mechanisms can be applied to the convertible network. For example, transport protocols, flow scheduling algorithms, and traffic re-routing solutions can adapt to convertible networks naturally.

2.5 Realization of Convertible Networks

Convertibility is achievable using circuit switches. By changing the circuit switch configurations, cables can be rewired to different outgoing connections as if they are plugged/unplugged manually. In this thesis, we use “converter switch” to denote

circuit switch with the special functionality of achieving convertibility.

The choice of specific switching technology depends on the existing devices already deployed in the data center. If the data center has copper cables in place, electrical crosspoint switches whose per-port cost is as low as \$3 [72] can be used. Crosspoint switches can scale up to 160 ports, and the switching latency is only 70ns. These converter switches split some cables into two parts. Because crosspoint switches are passive devices, cables connected to a converter switch do not need active elements. If manufactured properly, the cost of two cables each with only one active element at the packet switch end is equivalent to the cost of the original cable.

Many data centers nowadays use optical fibers for cross-rack connections. To avoid the cost of extra transceivers, optical circuit switches are sensible options for converter switches. We exploit cheap small port-count circuit switches, such as 2D MEMS and Mach-Zehnder switches to minimize deployment costs. These switches are fabricated on a planar substrate using lithography. Losses from photonic signal crossings or other effects limit the port count to modest scale. The mass production cost is dominated by packaging. With significant advances in photonic packaging, the per-port cost of these switches will be far cheaper than their counterparts that scale to hundreds of ports. While we are not able to project future costs precisely, we anticipate that the per-port cost will become reasonably cheap as photonic packaging technology advances. These converter switches can scale to 32 ports, and the switching delay is around $40\mu\text{s}$. The difference between transmit power and receive sensitivity of commercial optical transceivers can be over 8dB [73], which easily overcomes the insertion loss of most optical switches. Amplifiers are thus not needed.

Despite the realization technology, converter switches have relatively small port count. Because small converter switches cannot provide general connectivity, building

large-scale data centers requires intimate combination with the existing switching power in the network. Therefore, convertible networks employ interleaving converter switches and Ethernet switches to provide topological flexibility for the full scope of a data center.

2.6 Implementation Criteria for Convertible Networks

The implementation and management of data center networks involve a lot of manual efforts, such as initial deployment of equipments, wiring, upgrade and expansion, and trouble-shooting. Standard principles have been introduced to simplify these procedures. We follow these criteria in the implementation of our convertible networks. They apply to each and every architectural designs in this thesis.

Modular Design: The last decade has seen the emergence of the modular data center [74], a self-contained shipping container complete with servers, network, power, and cooling, which is usually referred as a Pod. Organizations like Google and Facebook have begun constructing large data centers out of Pods [2, 3], and many traditional server vendors now offer products in this space [75–77]. These modular Pods can be easily integrated to form a data center, and the inter-Pod wiring is simplified with bundled cables and streamlined connectors. Convertible data center networks require converter switches to be distributed across the network. We aim to make the changes transparent to the network operator. We insert converter switches into the Pods and hide the additional wiring inside. As our new type of Pod is manufactured, the network operator can implement the convertible data center as easily as regular data centers.

Regularized Wiring: The inter-Pod connections in data centers follow a highly organized wiring pattern. For example, in the *status quo* Clos structure for data center

networks, a set of core switches connect to the Pods using the same wiring pattern, that is each core switch connects to all the Pods each with one link. In convertible networks, because of the different network structure, inter-Pod wiring needs to be changed. We seek regular wiring patterns to minimize the wiring complexity. These patterns follow straightforward rules, such as repetitive patterns, shifting patterns, and swapped patterns. In the specific architectures described later, we produce wiring plans or wiring algorithms that are readily executable for practical wiring.

Packaging: When failures happen, the network operator enters the Pods to diagnose the problem. Equipment packaging within the Pod is thus very important, since a well organized packaging plan largely simplifies the trouble shooting process. In convertible networks, we need to package the converter switches properly along with the other devices. Converter switches have very limited port count, so we deploy a multitude of them in the Pod to enable Pod-wide connectivity. With today's technology, converter switches can be manufactured at minimal size, and they can be easily compacted. For instance, Lucent's 64×64 switch has a size of $100 \times 120 \times 20 \text{ mm}^3$, which can be mounted on a standard circuit board [78]; and Fujitsu's 80×80 switch has a packaged size of $77 \times 87 \times 53 \text{ mm}^3$ [79]. We can compact the converter switches into two groups and place them on the left and right sides of the Pod. Each compacted group can have aggregated connectors pointing to the converter switch ports, which facilitates connections to regular switches in the Pod.

Chapter 3

OmniSwitch: A Universal Switch as the Basic Building Block for Data Centers

In this chapter, we introduce OmniSwitch as the first step towards convertible data center networks. OmniSwitch is a production-ready modular container that serves as an universal building block for constructing data centers of various scales. It interleaves converter switches with Ethernet switches to provide local topology optimization and large-scale connectivity. It also addresses practical issues such as easy wiring, efficient backup, and incremental expansion. We design an example control algorithm for a traffic optimization use case. Simulation results show that our solution is effective in provisioning bandwidth for cloud tenants and reducing transmission hop count at low computation cost. The detailed design is presented in the following sections.

3.1 Motivation

In traditional data centers, thousands of servers are connected through a multi-rooted tree structure of Ethernet switches. Figure 5.3 depicts an example data center network. At each layer of switches, the upstream bandwidth is only a fraction of the downstream bandwidth, creating a bottleneck in the network core. Nowadays, novel network architectures with high bisection bandwidth have been studied to overcome this limitation [1, 11, 12].

Yet measurement studies show that the utilization of core links is highly imbal-

anced [80,81], indicating making good use of the existing bandwidth is more critical than adding bandwidth to the network. A recent trend is to optimize the bandwidth utilization leveraging the diverse routing paths in data centers. This set of works include multi-path routing and transport protocols for load balancing [14–18,82], flow scheduling mechanisms for transmission acceleration [19–21,83], and virtual tenant allocation heuristics for cloud service performance guarantees [26–29,84].

Besides routing flexibility, there is another level of flexibility that was rarely explored for bandwidth optimization: **topological flexibility**. Static network topologies lock up bandwidth in fixed links, so congested links cannot get more bandwidth even if it exists in the network. With a configurable network topology, bandwidth can be moved to transmission hot spots as needed. In the Figure 5.3 example, virtual machine (VM) 1 and 2 are placed in different edge subnetworks and must communicate through the network core no matter how the traffic is routed. If we move the bold link to the dashed position, we construct a shorter path between the VMs and reduce the bandwidth consumption in the network core. Although migrating VM 1 to location 3 achieves the same effect, it is undesirable because VM migration is expensive [85] and a tenant may request for storage (SAN) and connectivity (WAN) that are not movable.

Topological flexibility is achievable using circuit switches. By changing the circuit switch configurations, cables can be rewired to different outgoing connections as if they are plugged/unplugged manually. Modern data centers have optical fibers and optical transceivers in place for high-bit-rate transmission [86]. Optical circuit switches align well with the existing data center infrastructure, and thus become a sensible choice of implementation. The link change in Figure 5.3 can be realized by inserting an optical circuit switch between the relevant aggregation and ToR switches.

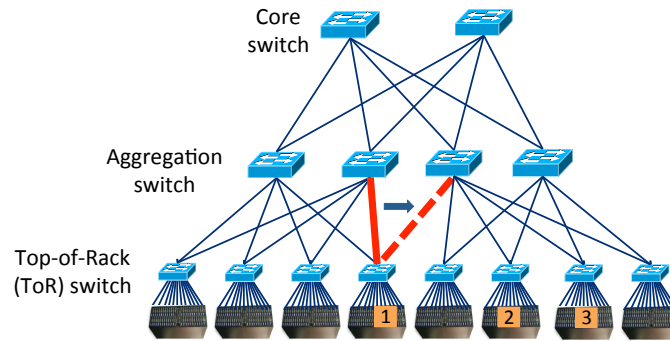


Figure 3.1 : An example data center network. The transmission hop count between VM 1 and VM 2 is originally 5. Moving the bold link to the dashed position reduces the hop count to 3.

Topological flexibility provided by optical circuit switches also simplifies deployment, upgrade, and management for complex data center networks. Constructing data centers requires complex wiring, and cable rewiring for later changes is especially challenging. If cables are interconnected through circuit switches, detailed rewiring after the initial deployment can be maintained automatically by cable management software. This introduces opportunities for dynamic topology optimization in case of switch or server failures, adding new equipments for incremental expansion, firmware upgrade for offline switches, and switch power-down during off-hour operation. Most data centers deploy one-on-one backup for each Ethernet switch for fault tolerance. 1 out of N sparing can be achieved with configurable topology. A single spare switch connected to multiple switches through optical circuit switches can be brought online as needed to replace any switch that fails. This enables more efficient backup and reduces the cost for redundancy significantly.

We present OmniSwitch, a convertible data center network architecture, and leverage its topological flexibility to utilize and manage the data center efficiently. Om-

niSwitch exploits cheap small port-count optical switches, such as 2D MEMS, Mach-Zehnder switches, and switches using tunable lasers with array waveguide gratings, to minimize deployment costs. These switches are fabricated on a planar substrate using lithography. Losses from photonic signal crossings or other effects limit the port count to modest scale. The mass production cost is dominated by packaging. With significant advances in photonic packaging, the per-port cost of these switches will be far cheaper than their counterparts that scale to thousands of ports. Because small optical switches cannot provide general connectivity, building large-scale data centers requires intimate combination with the existing switching power in the network. OmniSwitch employs interleaving optical switches and Ethernet switches to provide topological flexibility for the full scope of a data center. Evaluations in Section 3.3.3 demonstrate small optical switches integrated in the OmniSwitch architecture are effective enough to give considerable topology flexibility.

In the rest of the chapter, we describe the OmniSwitch architecture and the control plane design. We use VM clustering as a case study and propose a control algorithm that enhances locality of traffic in the same tenant. We evaluate our solution using simulations in the tenant provisioning scenario. Compared to intelligent VM placement on a fixed network topology, running our VM clustering algorithm given dumb VM placement on the OmniSwitch configurable topology can reduce the rejected bandwidth by 60%. Our approach also reduces the provisioning time for large tenants from 17min to 1s.

3.2 OmniSwitch Design

3.2.1 Architecture

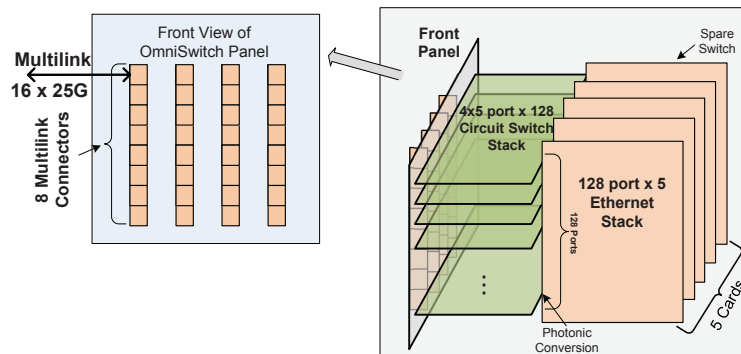


Figure 3.2 : Internal of an OmniSwitch cabinet

OmniSwitch deploys identical hardware building blocks to provision port count, bandwidth, and reliability. Figure 3.2 illustrates an OmniSwitch module that combines electrical packet switches and optical circuit switches into a single cabinet. The Ethernet stack can be populated with up to 5 cards each having a 128-port Ethernet switch ASIC. The 5th card is a spare switch to provide fault tolerance and always-on maintenance. The Ethernet switches are connected through electrical-to-optical converters, and then a stack of 4×5 photonic circuit switches, to optical front panel connectors. 16 25Gbps bidirectional fibers are bundled into one multilink to reduce the number of manually installed cables. After plugged into a multilink connector, these individual fibers are connected vertically across 16 adjacent optical circuit switches. Each circuit switch allows arbitrary optical connections between a row of individual links inside the front panel and a corresponding row of Ethernet ports that span the Ethernet stack. Multilink connectors provide connectivity to both end devices (servers or ToR switches) as edge bandwidth and to other multilink connectors

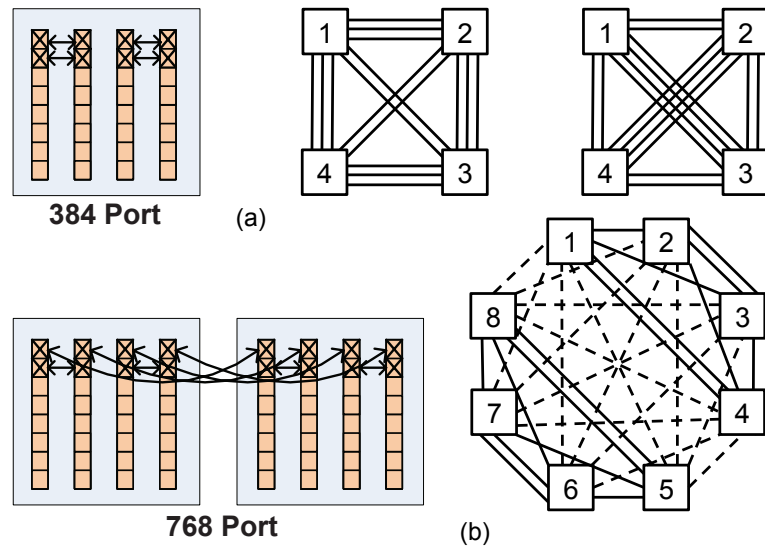


Figure 3.3 : OmniSwitch mesh network. The right subfigures show topologies of the Ethernet switches. Switch 1, 2, 3, 4 are in one cabinet; switch 5, 6, 7, 8 are in another cabinet. Each line represents 4 individual fibers. Solid lines are connections within a cabinet; dashed lines are connections across cabinets.

in the same or different OmniSwitch cabinets as core bandwidth. The proportion of core over edge bandwidth determines the oversubscription ratio.

Mesh networks can be realized using single or multiple OmniSwitch cabinets. In Figure 3.3 (a), the 4 active Ethernet switches are each connected to other Ethernet switches through 2 multilinks. The remaining 384 individual fiber ports can be used for end devices. Specific connections among the Ethernet switches are decided by the circuit switch permutations. We present two possible topologies, where the total bandwidth between switch pairs are adjustable depending on traffic demand. Figure 3.3 (b) shows a larger network that gives 768 end-device ports. Ethernet switches in the same cabinet connect to each other using 1 multilink each. They each also connect to switches in the other cabinet using 1 multilink. A possible topology is

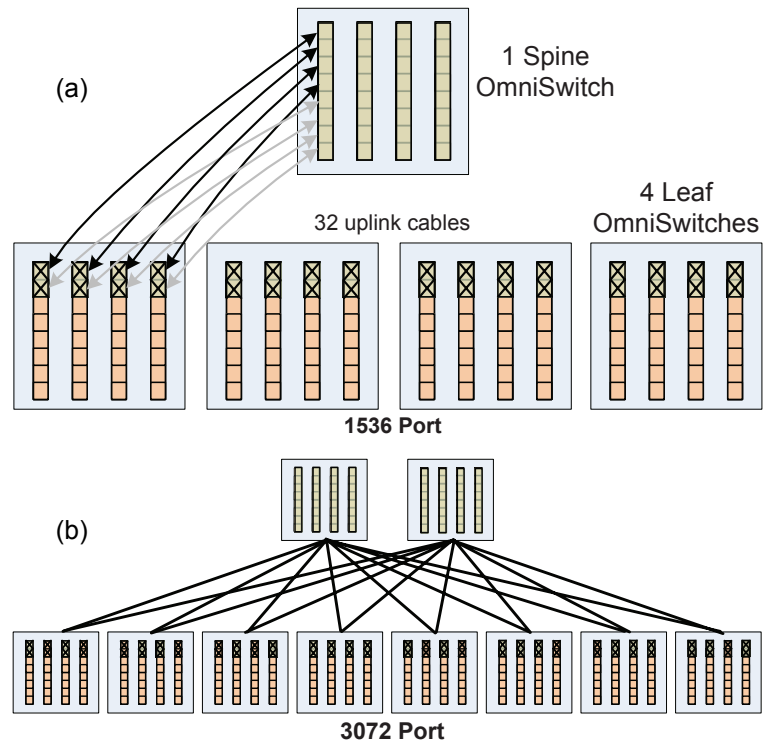


Figure 3.4 : OmniSwitch tree network. In subfigure (b), the 2 uplinks out of each cabinet refer to the 4 dark and light multilink connections in subfigure (a) respectively.

shown.

OmniSwitch cabinets can also be structured as tree networks. Spine cabinets are interior nodes in the tree and only provide connectivity for the children cabinets. Leaf cabinets connect to both end devices and the parent spine cabinets. Figure 3.4 (a) is the topology used for our evaluations in Section 3.3.3. 4 leaf OmniSwitch cabinets each provide 8 upward and 24 downward multilink ports. The dark and light lines show how uplink ports on leaf cabinets are connected to the spine cabinet. In our example network, 8 ToR switches are connected to each multilink connector, each ToR switch having 2 25Gbps individual uplinks. A ToR switch hosts 8 servers each using a 25Gbps downlink. The network has 6144 servers. The ToR switches are 4:1

oversubscribed and each cabinet provides 3:1 edge over core bandwidth, so the overall oversubscription ratio in the network is 12:1.

Figure 3.4 (b) shows a 3072 port configuration using 8 leaf cabinets and 2 spine cabinets as a Clos topology. The leaf cabinets are connected to the core cabinets in a similar fashion to Figure 3.4 (a). The lines between the leaf and spine cabinets each represent 4 multilink cables. For each leaf cabinet, the two lines refer to the dark and light multilink connections in Figure 3.4 (a) respectively.

3.2.2 Advantage Discussion

Easy wiring: OmniSwitch reduces wiring complexity using multilink cables. Detailed interleaving for individual links are handled by circuit switch configuration software. This enables automatic cable rewiring after a hardware failure, during a switch firmware upgrade, or after partial power-down during off-hour operation.

Incremental expansion: OmniSwitch cabinets can be partially populated with Ethernet switches and servers. As new equipments are added to the cabinet, circuit switch settings are configured to adapt to the change, avoiding manual rewiring of existing links. As shown in Figure 3.3 and Figure 3.4, it is also simple to purchase additional OmniSwitch cabinets to form larger networks.

Efficient backup: As Figure 3.2 shows, by configuration the circuit switches, the 5th spare switch can be brought online to replace any Ethernet switch that fails. Compared to most data centers where each switch has a stand-by backup, OmniSwitch reduces the sparing hardware and achieves more efficient backup.

Traffic optimization: Topological flexibility can enhance traffic locality and reduce transmission hop count. Links that exchange the most traffic can be optically configured to a common Ethernet switch to minimize the traffic sent to higher layers in

a network hierarchy. Opposite to traffic locality, load balancing and failure resilience can be achieved by optically directing traffic relevant to the same tenant to different Ethernet switches.

Cost effectiveness: Small optical switches are potentially far cheaper than the large counterpart, despite less flexibility. Circuit switches require one-to-one mapping between the input and output ports. As Figure 3.2 depicts, the cables connected to the same optical switch cannot reach the same Ethernet switch. Evaluation result in Section 3.3.3 shows small optical switches can provide considerable topological flexibility, thus OmniSwitch makes a good tradeoff between configurability and cost.

3.2.3 Control Plane

The OmniSwitch architecture requires a control plane (1) to program optical switch connectivities for topology optimization and (2) to enforce routing for quick adaptation to topology changes. Because a data center is administered by a single entity, an emerging trend is to leverage centralized network control to achieve global resource management [16, 19, 52]. We follow this trend to deploy a centralized network controller for OmniSwitch, which is a user process running on a dedicated machine in a separately connected control network.

Most optical switches can be configured via a software interface, and existing works provide basic routing mechanisms we can borrow. For example, after the topology is determined, our network controller can pre-compute the paths and program the routing decisions on switches using software-defined networking (SDN) protocols [16, 18, 82] or VLANs [15, 49, 87], or on end hosts by source routing [26]. The control logic should be customized to different use cases, such as localizing traffic to save core network bandwidth, balancing workload to improve service availability, powering

down some Ethernet switches to save energy, activating the spare Ethernet switch to recover from failure, etc. We design a control algorithm that configures topology and routing simultaneously for the VM clustering use case.

3.3 VM Clustering: A Case Study

In cloud computing terminology, tenant refers to a cluster of reserved VMs. A VM communicates with a subset of other VMs in the same tenant; there is almost no communication between VMs in different tenants [88]. VM clustering is to localize traffic within the same tenant by optically configuring end-device links that exchange the most traffic to a common Ethernet switch. The algorithm requires no control of VM placement and seeks opportunities for optimization in the network.

3.3.1 Problem Formulation

VM clustering can be realized at the flow level or the tenant level, reconfiguring the network either to address instant traffic changes at real-time or to address tenant bandwidth requirements that last for substantial time. We perform tenant management in this case study, because frequent topology changes cause disruptions in the network and degrade transport performance. Tenant bandwidth requirements can be expressed by different network abstraction models [26, 27, 29, 89]. Here we use the simple pipe model that specifies bandwidth requirement between each VM pair as a Virtual Link (VL) [26, 84]. Other models apply to OmniSwitch as well. The pipe model can be constructed either by user specification or by bandwidth prediction tools [90].

Our problem is to place the VLs in the OmniSwitch network, so that maximum amount of bandwidth that tenants require can be hosted. Placing VLs on a fixed network topology can be formulated as a multi-commodity flow problem. Because splitting VLs across several physical links can cause packet reordering, we seek integer assignments to the problem, which is NP-complete [91]. In a configurable network like OmniSwitch, there are numerous possible topologies, making the search space

even larger. We design a heuristic algorithm to approximate the global optima.

3.3.2 Control Algorithm

For each tenant, the algorithm takes in the physical locations of VMs and the bandwidth requirements of VLs. It accepts the tenant if it accommodates all the VLs with bandwidth guarantees, otherwise it rejects the tenant and recycles the allocated resources. We assume a tree structure of OmniSwitch cabinets, as shown in Figure 3.4. The OmniSwitch cabinets run the same sub-procedure, layer by layer from the edge to the root of the tree. The output of children cabinets is the input of parent cabinets.

In each cabinet, the algorithm handles VLs in the order of decreasing bandwidth requirement. Because configuring the optical switches can rewire cables to different Ethernet switches, we search through the egress and ingress uplinks with sufficient bandwidth on the source and destination VMs respectively to check what Ethernet switches can service the VL. If the egress and ingress uplink can reach up to the same Ethernet switch, the VL can be provisioned within this cabinet, demanding no extra bandwidth from the upstream cabinets. Optical circuit switch allows an input port to be connected to only one output port, thus locating VLs from the same tenant onto the same physical link saves optical ports for other tenants. We use a scoring function for the VL uplink assignment, which favors links heavily utilized by the tenant. If the VL must traverse different Ethernet switches, e.g. optical ports connected to the same Ethernet switch occupied already, we place the VL on egress and ingress uplinks with high scores and let the upstream cabinet deal with the connection between the Ethernet switches.

Table 3.1 : Average hop count when $load = 0.8$

dumb	SecondNet	OmniSwitch	OmniSwitch
+fixed Clos	+fixed Clos		(big OCS)
4.622	4.164	3.217	3.048

3.3.3 Evaluation

Simulation Setup

To demonstrate the power of topological flexibility, we compare two solutions in a tenant provisioning scenario: dumb VM placement on the configurable topology vs. judicious VM placement on a static topology. For the first solution, we simulate the example OmniSwitch architecture in Figure 3.4 (a). When a tenant is subscribed, we provision VMs by contiguous placement and run the VM clustering algorithm to accommodate bandwidth for VLs. For the second solution, we simulate a Clos network with the same number of Ethernet switches and servers, and run the SecondNet tenant provisioning algorithm [26]. We simulate dumb VM placement on the fixed network as the baseline of comparison. To analyze the effectiveness of small optical switches, we also compare the original OmniSwitch with an alternative implementation using one big optical switch for each cabinet.

The simulated networks have 6144 servers. SecondNet places VMs within tenant onto different servers. For fair comparison, we give each server the capacity to host a single VM in these experiments. Each simulation run consists of 1000 Poisson tenant arrivals and departures. The tenant size and bandwidth requirements are sampled from the Bing data center workload [88]. The mean tenant size (S) is 79 and the

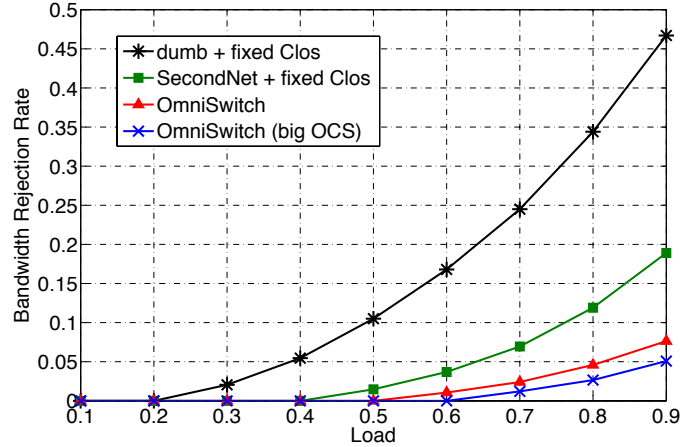


Figure 3.5 : Average bandwidth rejection rate under different load

largest tenant has 1487 VMs. We keep the tenant duration time (T) fixed and vary the mean arrival rate (λ) to control the *load* on the data center, which is defined as $\frac{S \times \lambda \times T}{6144}$, or the proportion of requested over the total VM slots.

Simulation Results

The simulated networks have 12:1 oversubscription ratio, so tenants may be rejected due to lack of network capacity. In this case, all bandwidth required by the VLs are considered rejected. We define **bandwidth rejection rate** as the amount of rejected bandwidth relative to the total requested bandwidth. We use this metric to evaluate each solution’s efficacy to accommodate tenants.

Figure 3.5 shows OmniSwitch rejects very little bandwidth even when the load is high, which demonstrates its effectiveness in localizing traffic given the simple VM placement. The OmniSwitch implementation using big optical circuit switches only reduces the rejection rate slightly, indicating small optical switches can provide considerable topological flexibility. SecondNet is much better than the dumb solution,

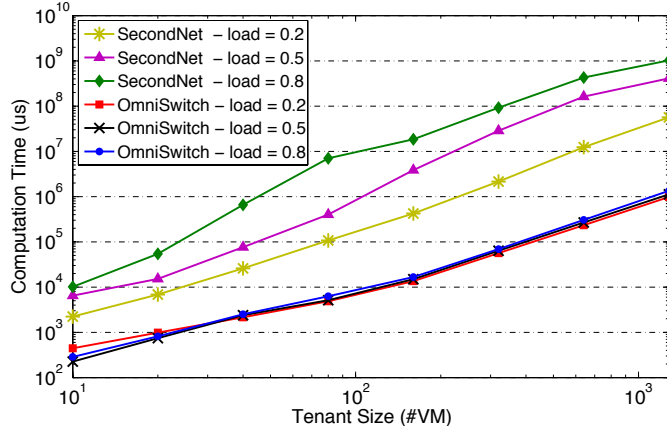


Figure 3.6 : Algorithm computation time for various tenant size and load

because it pre-configures the servers into clusters by hop count and prioritizes small-hop-count clusters for VM placement. However, it still rejects over $2\times$ as much bandwidth as OmniSwitch. On the fixed topology, if a cluster cannot host the entire tenant, SecondNet must move to large hop-count clusters for resources. OmniSwitch utilizes bandwidth more efficiently by constructing connections dynamically according to bandwidth requirement of individual VLS.

We measure the **average hop count** of the provisioned VLS to help interpret the above results. As shown in Table 3.1, the average hop count on the OmniSwitch network is significantly shorter than that of the SecondNet solution, which explains why OmniSwitch can host a lot more requested bandwidth. Big optical switches further reduce the hop count, but the bandwidth rejection rate in Figure 3.5 makes little difference. This is because OmniSwitch successfully reduces path length for most VLS, leaving sufficient core bandwidth for the rest VLS.

In Figure 3.6, we compare the **computation time** of the SecondNet algorithm and the OmniSwitch VM clustering algorithm. OmniSwitch can finish provisioning a large tenant with over 1000 VMs in around 1s, and the computation time is not

sensitive to variation of load; while SecondNet takes up to 17min and the computation time grows significantly as the load increases. Although SecondNet pre-clusters the data center to reduce the problem size, it still needs to do exhaustive search in each cluster. This is quite expensive, especially when the servers are heavily occupied. The search space for the OmniSwitch VM clustering algorithm is very small. Since it seeks optimization for pre-allocated VMs, it only needs to search through a few uplinks and possible optical switch connections. Table 3.1 shows the algorithm keeps most traffic within edge cabinets even at high load, so the search space does not enlarge with load increase.

3.4 Summary

This chapter presents OmniSwitch, a modular data center network architecture that integrates small optical circuit switches with Ethernet switches to provide both topological flexibility and large-scale connectivity. Mesh and tree networks can be easily constructed with identical OmniSwitch building blocks. Topological flexibility can improve traffic optimization and simplify network management. We demonstrate its potential with the VM clustering case study, where we give a control algorithm that optimizes both topology and routing to enhance locality of traffic within tenant. Our approach is evaluated using simulations driven by a real data center workload. Compared to the state-of-the-art solution, it reduces the average path length significantly and services more bandwidth using minimal computation time. Small optical switches are proven to provide similar topological flexibility to a high port-count counterpart.

Chapter 4

Flat-tree: A Convertible Data Center Architecture from Clos to Random Graph

In this chapter, we introduce flat-tree, which further explores convertible data center networks by changing the entire network from one topology to another. It can be implemented as a Clos network and later be converted to approximate random graphs of different sizes, thus achieving both Clos-like implementation simplicity and random-graph-like transmission performance. To convert between these least-alike topologies, It performs cable rewiring via converter switches to flatten the Clos' tree structure and redistribute servers across the switches. We also design an architecture-specific control system that achieves multi-path routing with a moderate number of network states. We evaluate the performance of flat-tree using extensive simulations and a testbed implementation. Flat-tree effectively approximates random graph, and it is able to optimize diverse workloads with different topology options. We demonstrate bandwidth increase through topology conversion, and this improvement can be translation into reduction of communication time in Spark and Hadoop applications. The detailed design is presented in the following sections.

4.1 Motivation

The fundamental trade-off in data center network design is low implementation complexity versus high transmission throughput. Clos, or multi-rooted tree, is the *de-facto* standard data center network architecture because of its highly organized struc-

ture [2, 3]. Figure 4.2b shows an example Clos network. The central wiring between switches in adjacent layers are relatively easy to manage, and the network can be expanded to arbitrary size by adding stages. Bandwidth oversubscription can occur at any layer of switches to save cost. Modular Pods are usually adopted as building blocks to further ease network deployment and management. However, Clos networks have suboptimal throughput, as traffic needs to traverse up and down the network hierarchy and the resulting inefficiency exacerbates oversubscription.

In contrast, random graphs are proven to have optimal throughput [4, 5]. Without rigid structures, switches are more directly connected at shorter path lengths. If implemented using the same switches and servers as a Clos network, a random graph can provide richer bandwidth and effectively alleviate the oversubscription problem. To address the heterogeneous workloads in data centers, it is desirable to construct random graphs at different scales to adapt to the various service cluster sizes [6–9], e.g. a network-wide random graph to serve large clusters and regional random graphs as part of the network to serve small ones. Yet the neighbor-to-neighbor wiring between random switch pairs are complicated, making real-world implementation a daunting task.

This dilemma poses a natural question: is it possible to have random-graph-like performance at various scales with Clos-like implementation simplicity?

We address this question by an unconventional proposal: a convertible data center network architecture called flat-tree*, which converts topologies between Clos and approximated random graphs. We combine the best of both worlds by building the

*The name “flat-tree” captures the dual nature of the proposed architecture. It can function as approximated random graphs (“flat” networks) and Clos (multi-rooted “tree”). It is as easy to implement as a “tree” network and has good performance as “flat” networks.

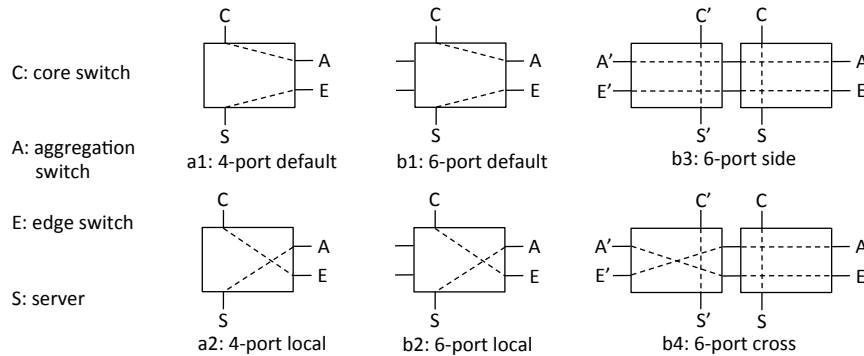


Figure 4.1 : Converter switch configurations

data center as a Clos network and converting it to approximate random graphs at different scales.

Flat-tree leverages inexpensive small port-count converter switches to convert topologies dynamically. By changing the configurations of the converter switches, cables are rewired to different outgoing connections, as if they were unplugged and replugged manually. Flat-tree takes a pragmatic approach to start from a Clos network and addresses challenges of flattening the tree structure to approximate random graphs. Specifically, how to equalize switches in different layers and relocate servers from edge to aggregation and core switches? How to break the hierarchy and connect the network core and edge directly? How to enable connections between switches in the same layer at minimum wiring complexity?

Flat-tree inherits the merits of packaging and wiring from Clos networks. It adopts the modular Pod design. Additional hardware and wiring are packaged in Pods, leaving the same external connectors as a Clos counterpart. Pods are connected to core switches with a customized regular wiring pattern. Adjacent Pods are interconnected through multi-link side connectors to allow simple neighbor-wise wiring.

Flat-tree can approximate random graphs at different scales, ranging from a Pod,

to a subnetwork comprising multiple Pods, to the entire network. It can also function as Clos, which benefits applications that require rich equal-cost redundant links, predictable path length, and rack-level locality. Flat-tree can operate in hybrid mode: the network is organized into functionally separate zones each having a different topology. Workloads are placed into suitable zones to optimize their performance. As the workloads change, the network can be reorganized to adapt to the new requirements.

We discuss design options for the control system and present the implementation details given the current technology. To exploit the link diversity in flat-tree, we adopt k -shortest-paths routing [92] and MPTCP [17], whose deployment in large-scale data centers is an open challenge. The enormous number of paths lead to explosion of network states. We propose an architecture-specific addressing scheme to aggregate IP addresses and use SDN-based source routing to relieve state-keeping at the switches. Packet-level simulations show that given various traffic patterns on flat-tree networks of different scales, the pragmatic implementation of k -shortest-paths routing and MPTCP constantly achieves comparable throughput to optimal routing from linear programming.

Linear programming simulations show that the performance of flat-tree is close to random graphs. Compared to a network-wide random graph, the difference in average path length is within 5% and the difference in throughput for large-clustered traffic is negligible. Flat-tree in hybrid mode optimizes traffic in different zones without interference and achieves the same throughput as separate flat-tree networks. To further evaluate the practical performance of flat-tree, we run packet-level simulations given real traffic traces from several production data centers each carrying different services. The results show that flat-tree is able to optimize for diverse workloads with different topology options. We implement a flat-tree prototype on a 20-switch

24-server testbed and run Spark and Hadoop applications with different topologies. The traffic reaches the maximal throughput only 2.5s after a topology change, proving the feasibility of converting the topology at runtime. The network core bandwidth is increased by 27.6% just by converting the topology from Clos to approximate random graph. This improvement can be translated into acceleration of applications as we observe reduced communication time in Spark and Hadoop jobs.

Table 4.1 : Throughput of clustered traffic normalized against the minimum value in the compared architectures

Cluster Size	Fat-tree	Random Graph	Two-stage Random Graph
8	1.91	1	1.16
30	1	1.38	1.65
100	1	1.59	1.17

4.2 Motivating Examples

4.2.1 The Case for Convertibility

Two reasons contribute to the diversity of data center workloads. First, enterprise data centers may deploy different services that have different traffic characteristics [9, 10]. For instance, the Facebook data centers with different services show different locality features. The Hadoop site has rack-level locality, while the web and cache sites have Pod-level locality [10]. Second, in public clouds, the virtual tenants have different sizes and traffic patterns [6–8]. For example, in a Microsoft data center, the mean tenant size is 79 VMs and the largest tenant has 1487 VMs [8, 93]. In this subsection, we use a simple example to motivate the necessity of using different network topologies to serve different workloads.

We construct a $k = 16$ fat-tree network [1], and use the same devices to form random graph and two-stage random graph networks [5]. The two-stage random graph network first forms a random graph in each Pod and takes the Pods as super nodes to form another layer of random graph together with core switches. Figure 4.2b, 4.2c and 4.2d show approximations of these topologies. To simulate intra-tenant commu-

nications in cloud data centers, we pack consecutive servers into clusters and create all-to-all traffic in each cluster. We measure the throughput following a well-adopted methodology [5], which assumes optimal routing and allocates bandwidth to flows using a linear programming solver.

Table 4.1 shows the normalized throughput with different cluster sizes. In the fat-tree network, each edge switch is connected to 8 servers, and there are 64 servers per Pod. 8-server clusters generate local traffic only, so fat-tree, without bottleneck in the network core, yields the highest throughput. Servers are distributed uniformly across all switches in the random graph. In the two-stage random graph, servers in each Pod are distributed uniformly across switches in the Pod, and core switches take no servers. As a result, the two-stage random graph has the second best performance since the traffic is served with better locality than in the random graph. For 30-server clusters, most of the traffic stays in Pods, so the two-stage random graph has the highest throughput. Random graph is particularly suitable for network-wide traffic because of the rich core bandwidth, so it performs the best for the cross-Pod traffic from 100-server clusters.

This example shows that different topologies perform better for different workloads, depending on the extent of locality they exhibit. We believe the network should be convertible between multiple topologies to adapt to different workloads. Our flat-tree architecture can work as a Clos network and can approximate random graph and two-stage random graph. The network can be configured to the topology that best suits the workload. In hybrid-mode, the flat-tree network is organized into functionally separate zones each having a different topology. Clusters of different sizes can be placed into suitable zones to optimize their performance. Our simulation experiments with real data center traffic in Section 4.5.4 demonstrate the performance

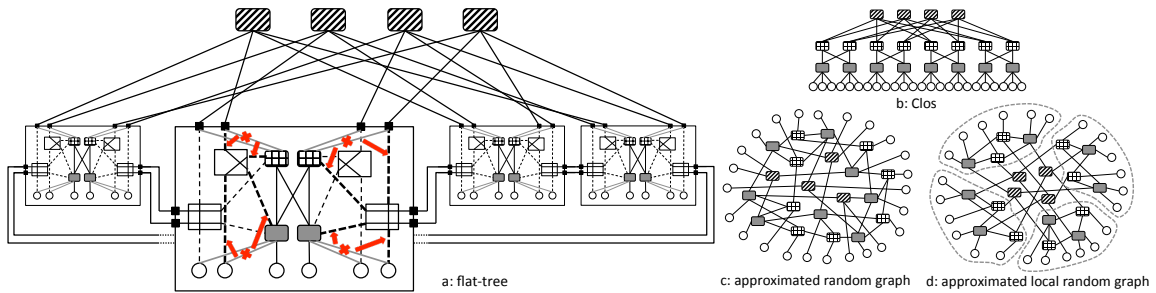


Figure 4.2 : Example flat-tree network and some achievable topologies. Core switches in stripe, aggregation switches in grid, edge switches in shade, and servers as circles. Gray lines are connections in the original Clos network, which are replaced with the dashed links connected to converter switches to form flat-tree. The converter switches show the configuration for approximated random graph. Flat-tree uses a customized wiring pattern to connect Pods to core switches.

advantage of each supported topology under different traffic.

4.2.2 Example Flat-tree Network

We use the simple flat-tree example in Figure 4.2 to demonstrate how to convert a Clos network to an approximate random graph. The gray lines represent original connections in the Clos Pod that need to be replaced by the dashed links in the flat-tree Pod. The most notable differences between Clos and random graphs are server distribution and types of links. In Clos networks, servers are attached to edge switches only and all links are hierarchical, either between edge and aggregation switches or between aggregation and edge switches. All switches are equal in random graphs. Servers are uniformly distributed to the switches, and the links are between random switch pairs. So, the first step of conversion is to relocate servers to aggregation and edge switches and to diversify the types of links.

These can be achieved by small port-count converter switches. As shown in the zoomed-in Pod, flat-tree breaks an edge-server link and an aggregation-core link in the Clos network, and connects the corresponding server, edge, aggregation, and core switches to a converter switch. Figure 4.1 illustrates the valid configurations of 4-port and 6-port converter switches. The “default” configuration enables the original Clos connections. The “local” configuration relocates the server to the aggregation switch and connects the core and edge switches directly. This change is local in the Pod.

4-port converter switches should not be used to relocate servers to core switches. If we connect the server and the core switch, the edge and aggregation switches must be connected as well, otherwise we waste a link. There are sufficient edge-aggregation links in the Pod, so this change fails to diversify the types of links. 6-port converter switches introduce side ports, through which two converter switches can be interconnected. The “side” and “cross” configurations both relocate servers to core switches, but connect edge and aggregation switches to their peers in different ways. We only allow 6-port converter switches in adjacent Pods to be interconnected for simple neighbor-to-neighbor wiring.

The number of 4-port and 6-port converter switches are determined by the layout of the Clos network. In Figure 4.2, each pair of edge and aggregation switches are connected to a 4-port converter switch and a 6-port converter switch, which show the approximate random graph configuration. Converter switches and the additional wiring are packaged in the Pod, keeping the same core connectors as a Clos Pod. The side connectors of 6-port converter switches are bundled as multi-link connectors to simplify inter-Pod wiring. Flat-tree Pods are connected to core switches via a customized wiring pattern (details in Section 4.3.2). In this example, the uplinks from Pods are swapped in different ways, so that servers are distributed uniformly

across the core switches.

Flat-tree converts between multiple topologies with different converter switch configurations. Figure 4.2b shows the Clos network, when all converter switches take the “default” configuration. Figure 4.2c shows an approximate global random graph, with the 4-port “local” and 6-port “side” configurations. In practice, we can also use the 6-port “cross” configuration to swap connections. Figure 4.2d shows approximate local random graphs in each Pod. It is configured in a way that half servers are connected to the edge switches and half to the aggregation switches. In this example, we use 4-port “local” and 6-port “default” configurations. Flat-tree can also operate in hybrid mode, with different combinations of the above topologies each in a number of Pods.

This chapter limits the discussion to one Pod layer connected by core switches. Flat-tree can be extended to multi-stages of Pods: the lower-layer Pods consider the edge switches in the upper-layer Pods as core switches; intermediate switch-only Pods take relocated servers from lower-layer Pods as their own servers. We leave the details to future work.

4.3 Flat-tree Architecture

4.3.1 Flat-tree Pod

Figure 4.3 depicts a flat-tree Pod. Without loss of generality, we assume the number of edge switches is a multiple of the number of aggregation switches. There are d edge switches and d/r aggregation switches. We pair up each edge switch E_j with aggregation switch $A_{j/r}$ and connect them to n 4-port converter switches and m 6-port converter switches. n and m represent the number of servers that can be relocated

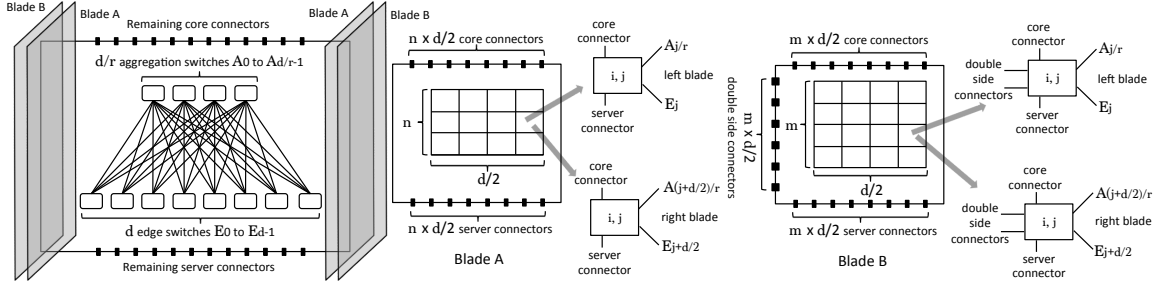


Figure 4.3 : A flat-tree Pod. A pair of edge switch E_j and aggregation switch $A_{j/r}$ connected to n 4-port converter switches and m 6-port converter switches. Converter switches are placed evenly on both sides as matrices. Blade A and B has 4-port and 6-port converter switches respectively.

dynamically to aggregation and core switches. We place the converter switches evenly on the two sides of the Pod: those connected to $E_0 \dots E_{d/2-1}$ locate on the left of the Pod and those connected to $E_{d/2} \dots E_{d-1}$ locate on the right. This forms a $n \times d/2$ matrix of 4-port converter switches, i.e. blade A in figure, and a $m \times d/2$ matrix of 6-port converter switches, i.e. blade B in figure, on each side of the Pod.

For both types of blades, converter switch $\langle i, j \rangle$ on the left blade is connected to edge switch E_j and aggregation switch $A_{j/r}$, and that on the right is connected to edge switch $E_{j+d/2}$ and aggregation switch $A_{(j+d/2)/r}$. Each 4-port converter switch connects to a core switch and a server, so blade A has $n \times d/2$ core connectors and server connectors. Each 6-port converter switch has a pair of side connectors as well, so blade B has $m \times d/2$ core connectors, server connectors, and double side connectors. There may be remaining core connectors on the aggregation switches and server connectors on the edge switches. The total number of core connectors and server connectors are equal to those in a Clos counterpart. If d is odd, a middle

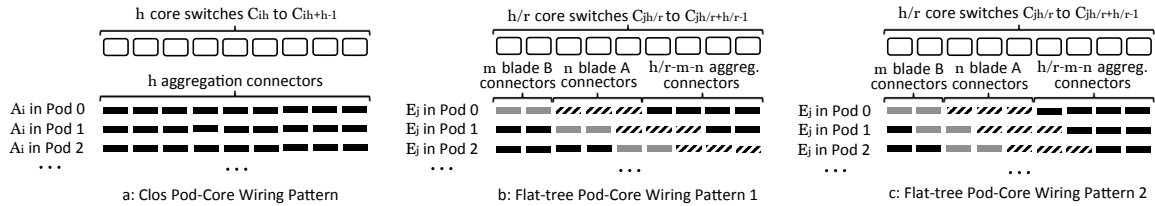


Figure 4.4 : Pod-Core wiring for the same set of connectors across Pods. All connectors are on aggregation switches in Clos; flat-tree has 3 types of connectors on blade A, B, and aggregation switches, enabling core-server, core-edge, and core-aggregation connections respectively.

converter switch can be on either side, but the side connectors of the 6-port converter switch are unused.

4.3.2 Pod-Core Wiring

In a Clos network, all Pod-core connections are between aggregation and core switches. Suppose each aggregation switch has h uplinks. As Figure 4.4a illustrates, aggregation switches with the same index i in different Pods are connected to the same group of h core switches via the aggregation connectors. Repeatedly for each Pod, this wiring pattern links the h connectors for each aggregation switch consecutively to core switches.

In flat-tree, as shown in Figure 4.3, there are 3 types of core connectors. Core switches can be connected 1) to servers via blade B connectors, 2) to edge switches via blade A connectors, and 3) to aggregation switches via aggregation connectors. The Pod-core wiring determines the distribution of servers and different types of links (to an edge or aggregation switch) across the core switches, thus affecting how closely flat-tree approximates a random graph.

As each aggregation switch corresponds to r edge switches, the h aggregation connectors in Clos are replaced with $n \times r$ blade A connectors, $m \times r$ blade B connectors, and $h - m \times r - n \times r$ aggregation connectors. The Clos wiring pattern is based on aggregation switches, each connected to h core switches. Since flat-tree has edge-core connections, its wiring pattern should be based on edge switches. Each edge switch corresponds to n blade A connectors, m blade B connectors, and $h/r - m - n$ aggregation connectors, which connects to overall h/r core switches.

We offer two wiring options, shown in Figure 4.4b and 4.4c. Connectors corresponding to the edge switches with the same index j in different Pods are connected to the same group of h/r core switches. Both wiring patterns connect the group of core switches consecutively to blade B connectors, followed by blade A connectors and aggregation connectors. They rotate in different ways across Pods. Pattern 1 packs blade B connectors continuously Pod by Pod throughout the set of core switches. Pattern 2 moves them forward by one more core switch as the Pod index grows. Both patterns wrap around within the group.

Physically, we suggest wiring Pod 0 first, by linking every m blade B connectors, n blade A connectors, and $h/r - m - n$ aggregation connectors in turn to core switches consecutively. We start from the left blades and move on to the right blades, until all connectors in the Pod are consumed. In this process, we mark the mapping between each edge switch and the corresponding group of h/r core switches. For the following Pods, connectors corresponding to each edge switch are connected to the marked h/r core switches according to the rotating patterns.

These wiring patterns have the following properties:

Property 1: For both wiring patterns, servers are distributed uniformly across the core switches.

Property 2: For both wiring patterns, the core switches have an equal number of links of the same type.

Flat-tree maintains structure to ease implementation, so servers and links must be permuted by wiring. These properties ensure that flat-tree well-approximates a random graph.

Because these patterns follow straightforward rules, they have low wiring complexity. Pattern 1 has better performance, because a core switch does not connect to servers from adjacent Pods at the same time, thus it takes advantage of side connections between adjacent Pods to the greatest extent. Yet when h/r is a multiple of m , different Pods are likely to repeat the same pattern, thus reducing the wiring diversity. In this case, pattern 2 is more favorable.

4.3.3 Inter-Pod Wiring

For adjacent Pods p and $p + 1$, the 6-port converter switches on the left blade B of Pod $p + 1$ are connected to those on the right blade B of Pod p by the side connectors. Recall from Figure 4.3 that the converter switches in the same column connect to the same pair of edge and aggregation switches. We want to connect an edge/aggregation switch to as many different switches as possible in the adjacent Pod, so we design a shifting wiring pattern such that the converter switches in the same column of the right Pod are connected to converter switches each in a different column of the left Pod. Specifically, let i and j be the row and column of the converter switch matrices, converter switch $\langle i, j \rangle$ on the left of Pod $p+1$ is connected to converter switch $\langle i, (d/2 - 1 - j + i) \% (d/2) \rangle$ on the right of Pod p , which represents the converter switch in the same row i and in the column i slots shifted from the mirrored column $d/2 - 1 - j$. We want the converter switches to be interconnected by different configurations, so

we have both peer-wise and edge-aggregation connections across Pods. If i is even, they take the 6-port “side” configuration (in Figure 4.1); if i is odd, they take the 6-port “cross” configuration. To streamline the connection of adjacent Pods, the side connectors on the same side of a Pod are bundled as a multi-link connector that integrates this wiring pattern.

4.3.4 Server Distribution

In a random graph, servers are distributed uniformly across the switches, because the random links roughly connect the switches in a uniform manner. Yet flat-tree maintains structures, e.g. the Clos connections between edge and aggregation switches, core switches connected to the Pods, though using customized wiring patterns, and the neighbor-to-neighbor wiring restricted to adjacent Pods. The path length of switch pairs is not uniform for flat-tree, so we should place servers intelligently to leverage the shorter paths in the network.

Recall that 6-port converter switches can relocate servers to core switches, and 4-port ones can relocate servers to aggregation switches, so the server distribution is determined by the choice of m and n . Because flat-tree aims at converting generic Clos networks, which may have very different layouts, it is difficult to pre-define the m and n values for optimal transmission performance. We suggest a profiling scheme: under the preferred Pod-core wiring pattern described in Section 4.3.2, vary m and n until they result in the shortest average path length over all server pairs. Xia *et al.* provided the sensitivity test for this approach [94].

4.3.5 Operation Modes

Global: Flat-tree approximates a network-wide (or global) random graph in the “global” mode. 6-port converter switches take either the “side” or the “cross” configuration (Figure 4.1 b3 or b4) depending on their row index in the matrix as described in Section 4.3.3. 4-port converter switches take the “local” configuration (Figure 4.1 a2).

Local: Flat-tree approximates a two-stage (or local) random graph in the “local” mode. It first forms random graphs in each Pod and takes the Pods as super nodes to form another layer of random graph together with core switches. 6-port and 4-port converter switches take the “local” configuration (Figure 4.1 a2 and b2) to relocate half servers to aggregation switches. Any remaining 6-port converter switches take the “default” configuration (Figure 4.1 b1).

Clos: Flat-tree functions as a Clos network by default. All converter switches take the “default” configuration (Figure 4.1 a1 and b1).

Hybrid: Flat-tree can be configured in the unit of a Pod, so it can have arbitrary combinations of the above three topologies each in a number of Pods. The converter switch configurations follow the rules in their corresponding mode.

4.4 Control System

Flat-tree requires a control system to change the network topology and to conduct routing accordingly. The main contribution of this work is the network architecture, and the control system is orthogonal to it. Here we give possible designs for the control system and show the implementation details. We acknowledge there may be alternative solutions.

Because a data center is administered by a single authority, we follow the recent trend of using a centralized network controller for global network management. Flat-tree has several operation modes with pre-known topologies, which designate a fixed set of configurations for the converter switches. The controller changes the topology by configuring the converter switches, via specific control mechanisms depending on the realization technology. For instance, most optical switches can be programmed via a software interface. The converter switch configurations for different flat-tree modes can be hard-coded into the controller.

For flat-tree Clos mode, we can use ECMP [95], two-level routing [1], or customized SDN routing with pre-computed paths [3]. We omit the discussions for these readily available solutions. The study on random graph network [5] suggests using k -shortest-paths routing [92] and MultiPath TCP (MPTCP) [17]. We adopt this approach for flat-tree global mode and local mode, because they approximate random graph and two-stage random graph respectively. Deploying k -shortest-paths routing and MPTCP in large data centers is an open challenge due to the scalability concern to maintain a huge number of network states. We explore feasible technologies for the deployment in the following subsections.

4.4.1 MPTCP

MPTCP has been standardized and widely used in academia and industry [96–98]. The kernel implementation has been released [99]. MPTCP establishes subflows via multi-homing: the end hosts using multiple IP addresses to distinguish paths. In flat-tree, servers have one uplink only, so we must associate multiple IP addresses to a single NIC. IP aliasing gives the solution by setting multiple virtual network interfaces. These virtual interfaces are linked to the physical interface by default, so traffic with different IP addresses can be forwarded by the physical interface.

The full-mesh option in MPTCP allows subflows with different combinations of the source-destination IP address pairs. For instance, with 2 IP addresses on both the sender and the receiver, we obtain $2 \times 2 = 4$ subflows. Therefore, the number of IP addresses per server is the square root of the number of concurrent paths, or k in k -shortest-paths routing. Not all subflows are needed sometimes. For example, 8-shortest-paths routing requires 3 IP addresses per server, thus creating one extra subflow. In such case, a straightforward workaround is to limit the routing logic to the necessary subflows only, and MPTCP will not allocate traffic to subflows with no end-to-end reachability.

This simple way of assigning IP addresses defines a flat address space, which may be inefficient considering the great number of servers in a large data center. The property of MPTCP to send traffic only with routable addresses gives the freedom for more intelligent addressing mechanisms. Generally, address assignment depends on the structure of the network and serves for the ease of routing. This task is particularly difficult for flat-tree, which has completely different network structures and routing paths for each topology. We propose a customized addressing scheme specific to the flat-tree architecture in the next subsection.

4.4.2 k -Shortest-Paths Routing

In k -shortest-paths routing, there are k routes for every source-destination server pairs. A critical consequence of the enormous number of paths is explosion of the network states. Let n and N be the number of servers and switches in the data center and L be the average path length, the average number of network states per switch is $\frac{n^2 \times k \times L}{N}$. For a large data center, this number can easily reach tens of million, far exceeding the storage and processing capacity of switches. k -shortest-paths routing requires matching both the source and destination IP addresses, and traditional ways of aggregation, such as destination IP lookup or prefix matching, do not readily work. A switch may forward packets for the same receiver to different ports, because they need to take different routes. Servers can be relocated to different switches under different flat-tree topology modes, making the definition of common prefix very challenging. We need novel approaches to factoring down the number of network states.

Addressing

We have two important observations from the flat-tree architecture and from an extensive analysis of the computed k -shortest paths in the network.

Observation 1: A server is connected to one and only one ingress/egress switch, regardless of the fact that it may be relocated to a different ingress/egress switch as the topology changes. So, there is no path diversion between servers and the connected ingress/egress switches.

Observation 2: The number of equal-cost paths is small in the approximate random graph flat-tree creates. The k -shortest paths between server pairs are nearly deterministic, with uncommon exception of ties. So, the k -shortest paths between

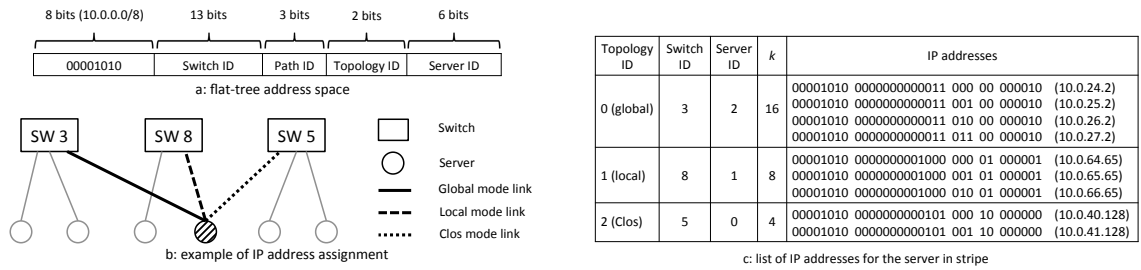


Figure 4.5 : Illustration of the addressing scheme. “a” shows the IP address fields in flat-tree. In the “b” example, the server in strip connects to switch #3, switch #8, and switch #5 respectively in the global, local, and Clos mode, where the number of concurrent paths, or k , is chosen to be 16, 8, and 4. The IP addresses assigned to this server are shown in “c”. All these addresses for every flat-tree topology mode are preconfigured on the server.

ingress and egress switches almost capture the full set of selected paths between source and destination servers.

Given these observations, it is promising to conduct prefix matching on the ingress/edge switch level. This way, the average number of network states per switch is reduced from $\frac{n^2 \times k \times L}{N}$ to $\frac{S^2 \times k \times L}{N}$, S being the number of ingress/egress switches. Usually 20 to 40 servers are connected to a top-of-rack switch (ToR) in a data center, so the number of network states can be reduced by a factor of 400 to 1600.

As discussed previously, the major difficulty in flat-tree is servers’ mobility. To guarantee common prefix for servers under the same ingress/egress switch, we need a different set of IP addresses for each flat-tree topology mode. Because we aim to change the network topology at runtime by software, it is infeasible to reset the server IP addresses manually for each topology. Thanks to the property of MPTCP to send traffic only with routable addresses, we can preconfigure all possible IP addresses for

each topology onto the servers and let the network controller dynamically load the routing logic for the subset of addresses particular to the topology in use.

Our definition of the address space is shown in Figure 4.5a. We assume IPv4 addresses and allocate IP addresses within the private 10.0.0.0/8 block. The first 13 bits after the fixed heading octet represent the switch ID of the ingress/egress switch. In flat-tree, all switches may serve as an ingress/egress switch. We associate each switch with a unique ID, which is not changed with the conversion of topology. This 13-bit field allows for 8196 switches, which is sufficient for an immense-scale data center. The next 3 bits are for the path ID in the k -shortest paths. As aforementioned, MPTCP distinguishes paths by different combinations of IP addresses between server pairs. This 3-bit field allows for 8 addresses at sender/receiver and thus supports $8^2 = 64$ concurrent paths at most, covering the range of k most data centers will use. The next 2 bits are used to specify the 3 possible flat-tree topologies. The rest 6 bits show the server ID under the ingress/egress switch. Because of the limited IPv4 address space, we cannot afford to assign a unique ID for every individual server. So, these IDs are reused for servers under different ingress/egress switches. This 6-bit field supports 64 servers per switch, which is enough for the 20 to 40 servers per ToR in most data centers. By this address assignment, we match the /24 prefix at the ingress/egress switches. This addressing scheme can be easily extended to IPv6 addresses, which even support globally unique server IDs.

Figure 4.5b shows an example of the address assignment. The server in stripe is connected to 3 different ingress/egress switches under different flat-tree modes. The servers under the same ingress/egress switch are ordered from left to right, so the server ID in the global, local, and Clos mode is 2, 1, and 0 respectively. The number of concurrent paths, or k , can be different under each mode, because each

topology may have optimum transmission performance with a different k . In this example, k equals 16, 8, and 4 for each topology, so we need 4, 3, and 2 IP addresses accordingly. Figure 4.5c lists the allocated IP addresses according to our addressing scheme. All these addresses for every flat-tree topology are preconfigured on the server at deployment time.

One possible problem is the overhead of MPTCP probing unused IP addresses for potential paths. In our small testbed with 4 concurrent paths, as shown in Section 4.5.5, we implement this addressing mechanism (6 addresses per server, 2 for each topology) as well as the naive address assignment (2 addresses per server, no unnecessary addresses). We observe no noticeable difference in throughput between the two approaches. Whether the overhead is a valid concern in large data centers is the direction of future work.

Source Routing

A common solution to relieving state management at switches is source routing [26, 100–102]. Segment routing is a natural fit to this request in the SDN world [103–105]. In segment routing, the k -shortest-paths routing algorithm can be implemented in the Path Computation Element (PCE), an equivalent of the centralized network controller, which enforces per-route states only at ingress switches. It relies on the MPLS [106] and IPv6 architecture. The ingress switch encodes the hops of a path as a stack of MPLS labels. The transit switches forward packets by dumb compliance of the label on top of the stack and pop it upon completion.

Not all data centers have the MPLS and IPv6 forwarding fabric, so we provide an alternative solution in the better recognized OpenFlow paradigm. Source routing is not supported in OpenFlow by default. From the rich literature of workarounds [100–

102, 107, 108], we pick a readily deployable approach without modification of the OpenFlow protocol [101]. We encode the path, represented as a list of next-hop output ports, into the source MAC address and use TTL as the location pointer in the path. Flat-tree is a small diameter network, where paths traverse less than 3 switches on average [94]. The 48-bit MAC address is able to hold 6 hops for switches having as many as 256 ports, which is sufficient for the need of the network. OpenFlow 1.3 allows matching arbitrary bits of a given field [109]. We can thus concatenate the transit hops in the MAC address and let intermediate switches match different bits using a mask depending on the TTL. For instance, if TTL equals 253 (2nd hop), we apply the mask 000000001111111100000000..... on the MAC address and match the extracted bits to all possible 256 ports to decide the right output port. This way, we need an entry per TTL per output port. So, the number of OpenFlow rules on the transit switches is $D \times C$, where D is the diameter of the network and C is the switch port count. This number is at most up to one thousand, far below the capacity of an OpenFlow switch. These rules remain the same as the flat-tree topology changes, so they can be preconfigured statically.

With source routing, the number of network states per ingress/egress switch is reduced to $S \times k$. This number is at most a few tens of thousand, within the capacity of high-end OpenFlow switches [101]. There is large room for optimization to further bring down this number. For example, in public clouds, tenants request virtual clusters where only machines within the cluster talk to each other. In this case, we can set in-cluster routing logic, which involves a small number of ingress/egress switches. Traffic is skewed in many enterprise data centers [6–10]. We can use diverse paths (large k) for a small number of elephant flows, and simple paths (small k) for a large number mice flows.

4.4.3 Topology Conversion

The conversion delay of flat-tree topologies is determined by the switching delay of the converter switches and the delay of changing the routing logic. Depending on the realization technology, the switching delay of converter switches ranges from several μs to hundreds of ms [41–43, 110, 111]. The network controller takes roughly 1ms to add/delete a network state [112, 113]. Instead of streaming the states all from a single network controller, we can speed up the state distribution by having a set of controllers each managing a number of switches. We designate a logically centralized controller to maintain the global network graph. It observes link failures and updates the graph, which happens infrequently and does not cause heavy burden. The k -shortest-paths routing algorithm is easily parallelizable, because the computation of paths between different nodes is independent. So, the distributed controllers can either work as dumb agents of the logically centralized controller and preload paths from it, or compute paths independently based on a consistent network graph. Following the trend of building customized switches for data centers [3], it is conceivable to push the computation to switches. This way, switches can update network states locally on simple signaling of topology change. The paths and the resulting network states can also be precomputed and stored into a table in memory to save the computation time. With these implementation options, we estimate the delay of changing the routing logic to be on the order of seconds.

We do not expect the network topology to be converted very frequently, e.g. constantly changing to optimize individual flows. We recommend converting the network topology for management purpose, e.g. to reorganize services, to deploy new services, to adapt to user requirement changes, etc. Generally, applications and transport protocols are tolerant to disruptions of a few seconds. However, there

are opportunities for optimizing data center applications and transport protocols to better align with convertibility.

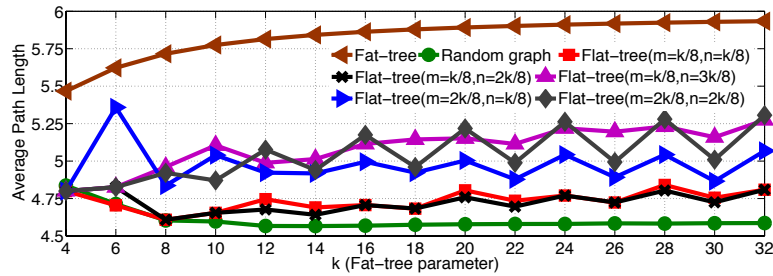


Figure 4.6 : Average path length of server pairs in the entire network

4.5 Evaluation

4.5.1 How well does flat-tree approximate random graph in theory?

We first evaluate the performance of flat-tree by linear programming simulations. Although flat-tree targets at converting generic, especially oversubscribed, Clos networks, our evaluations are based on fat-tree [1]. Because generic Clos networks can have very different layouts, e.g. arbitrary number of switches and servers, oversubscription at any possible layer, it is difficult to have a “typical” example for evaluation. Fat-tree gives the upper-bound performance for Clos networks, thus serving as a stress test for our solution. We construct fat-tree, random graph, and our flat-tree using the same equipments with the variance of k , the fat-tree parameter that defines the switch port count, the number of Pods, as well as the number of switches in each layer. We consider two flat-tree configurations: approximated network-wide random graph and approximated local random graphs in each Pod. When flat-tree approximates local random graphs, we compare it with two-stage random graph, which first forms random graphs in each Pod with the same number of links as flat-tree, and takes the Pods as super nodes to form another layer of random graph together with core switches. We also evaluate flat-tree in hybrid mode: having different proportions of

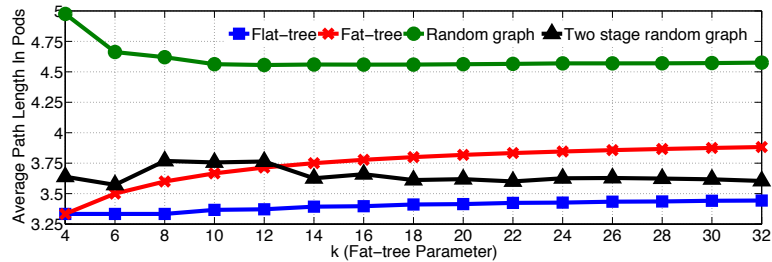


Figure 4.7 : Average path length of server pairs in each Pod

the network functioning as approximated global and local random graph respectively.

We use average path length in hops and throughput as the evaluation metrics. We assume converter switches function in the physical layer and do not contribute to path length. The throughput experiments follow a well-adopted methodology [4, 5]. We assume optimal routing and solve the maximum concurrent multi-commodity flow problem [114] using a linear programming solver. All links have one unit bandwidth. We relax the bandwidth constraints at the servers to show the switch-level capacity, which is relevant to the maximum number of servers a topology can accommodate. Measurement studies show two pervasive traffic patterns in data centers: broadcast/incast traffic from/to hot spots to/from a large number of servers, and all-to-all traffic within small clusters [6, 8, 9]. We simulate them by broadcast/incast traffic from/to a random target in 1000-server clusters and all-to-all traffic in 20-server clusters. We consider strong, weak, and no locality of workload placement to evaluate the topologies' sensitivity to it. Specifically, the workload is placed continuously across servers, randomly in Pods, or randomly in the entire network.

Average Path Length

We first determine m and n for flat-tree through the profiling mechanism described in Section 4.3.4. Flat-tree has the same equipments as the fat-tree counterpart, so $m + n \leq k/2$. We vary m and n at the interval of $k/8$, rounded to the closest integer if fractional. This process can happen at finer granularity with smaller intervals. We use Pod-core wiring pattern 2 when k is a multiple of 4 and pattern 1 otherwise for reason discussed in Section 4.3.2.

Figure 4.6 compares the average path length of flat-tree under the settings of different m and n against that of fat-tree and random graph. The desirable values for m and n are $k/8$ and $2k/8$, when flat-tree has the minimal average path length. It is notably shorter than that of fat-tree, and within only 5% difference to random graph. k as multiples of 4 are hard cases where pattern 1 tends to repeat frequently. Pattern 2 successfully maintains the average path length at a relatively low level. These results demonstrate that with the right choice of m and n , flat-tree approximates global random graph well and it improves against fat-tree significantly. We set $m = k/8$ and $n = 2k/8$ for the rest experiments.

We further evaluate the average path length between server pairs in the same Pod, when flat-tree functions as approximated local random graphs within each Pods. Figure 4.7 shows the result against fat-tree, global random graph, and two-stage random graph. Random graph performs the worst as servers scatter around the network, followed by fat-tree whose servers at the edge switches have locality. Flat-tree moves half the servers from edge to aggregation switches, reducing the distance between servers connected to different types of switches. Surprisingly, it outperforms two-stage random graph. In flat-tree, servers evenly distributed over edge and aggregation switches are connected by the regular Clos edge-aggregation links, which is more efficient than

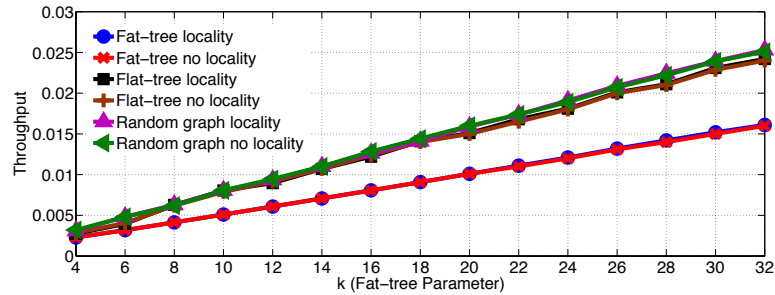


Figure 4.8 : Throughput of broadcast/incast traffic in 1000-server clusters

pure randomness in server distribution and inter-switch connections.

Throughput

We create 1000-server clusters, each server being involved in a single cluster. One random server in each cluster is the source/destination of broadcast/incast traffic to/from all the other servers in the same cluster. In the locality case, we pack clusters continuously across the servers; in the no locality case, we place them randomly throughout the network. Each Pod has $k^2/4$ servers, so one cluster spans multiple Pods even for large k under the locality setting. Flat-tree approximates a global random graph to accommodate such large clusters, so we compare its throughput with fat-tree and random graph. As shown in Figure 4.8, the throughput of flat-tree is very close to that of random graph and is $1.5\times$ that of fat-tree. The throughput grows linearly with k , the switch port count, as the few hot spots have increasing sending/receiving capacity. None of the topologies is sensitive to locality, due to heavy cross-Pod traffic. This set of results demonstrate that with arbitrary workload placement, flat-tree can achieve near-optimal performance for the prevalent broadcast/incast traffic at hot spots.

Then we create 20-server clusters featuring all-to-all traffic, which can fit in the

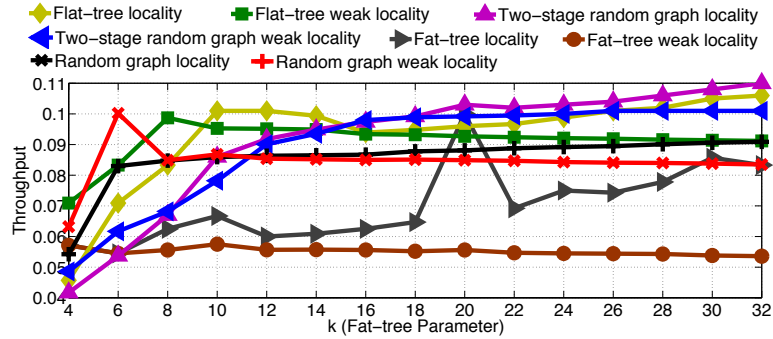


Figure 4.9 : Throughput of all-to-all traffic in 20-server clusters

Pod for most k . Flat-tree approximates local random graphs within each Pod to accommodate small clusters in Pods. So, besides fat-tree and random graph, it is also compared with two-stage random graph. We consider workload with locality, clusters packed continuously across servers, and workload with weak locality, clusters packed randomly in Pods as long as there are remaining servers. Weak locality is the worst-case simulation of resource fragmentation in workload placement.

Figure 4.9 shows flat-tree well approximates local random graph. It outperforms two-stage random graph for small networks ($k \leq 14$), and the difference in throughput is less than 6% and 9% respectively with strong and weak workload locality for larger networks. Flat-tree has shorter average path length in Pods, as shown in Figure 4.7, whereas two-stage random graph forms closer inter-Pod connections. The result is the outcome of the interplay between these factors. Traffic locality has greater impact on flat-tree, because the regular direct links between adjacent Pods are more likely to benefit consecutively packed servers. Fat-tree is highly sensitive to workload placement. It has sweet spots, such as $k = 20$, when most traffic is local in Pods. Its throughput drops significantly for weak locality, as even local traffic in Pods takes more hops through aggregation switches. Random graph has moderate throughput as

it does not specialize in local clustered traffic, but it is the least sensitive to workload locality. In reality, we expect the performance of flat-tree to be between the locality and the weak locality curves, given a reasonable level of fragmentation. In summary, flat-tree optimizes all-to-all traffic in small clusters effectively.

4.5.2 Can multiple topologies coexist in flat-tree?

Flat-tree can work in hybrid mode with different topologies each in a number of Pods. Workloads placed in different zones share the network core. We use experiments to answer the question whether flat-tree can optimize multiple workloads in separate zones without interfering with each other.

We construct flat-tree with 30 Pods, i.e. $k = 30$, and organize the network into two separate zones with varying proportions at an interval of 10%. We let flat-tree operate as an approximated global random graph in one zone and as approximated local random graphs within each Pod in the other zone. Each topology gets the same traffic pattern as the corresponding complete network as described in Section 4.5.1. We observe that regardless of the proportion, each zone constantly achieves the same throughput as that of the corresponding complete network under the same locality setting. Therefore, flat-tree in hybrid mode is as effective as building separate flat-tree networks, and the workloads in different zones can be segregated perfectly.

4.5.3 Is k -shortest-paths routing with MPTCP efficient enough?

We then evaluate the performance of k -shortest-paths routing and MPTCP to see how close the throughput they achieve is to the theoretical bound. We use the MPTCP packet-level simulator [115] and integrate Yen’s k -shortest loopless paths algorithm [92] into it. We compute k -shortest paths between ingress and egress

Table 4.2 : List of flat-tree topologies for evaluating the control system. Abbreviations: Edge Switch (ES), Aggregation Switch (AS), Core Switch (CS), Upstream Port (UP), Downstream Port (DP), Oversubscription Ratio (OR).

ID	#ES (#UP, #DP)	#AS (#UP, #DP)	#CS (#DP)	OR at ES	OR at AS	#Server
topo-1	128 (8, 32)	128 (8, 8)	64 (16)	4	1	4096
topo-2	72 (6, 24)	72 (6, 6)	36 (12)	4	1	1728
topo-3	128 (8, 64)	128 (8, 8)	64 (16)	8	1	8192
topo-4	128 (8, 32)	64 (16, 16)	32 (32)	4	1	4096
topo-5	128 (16, 32)	128 (8, 16)	64 (16)	2	2	4096
topo-6	128 (16, 32)	64 (32, 16)	32 (32)	2	2	4096

switches, because the proposed control system, described in Section 4.4, performs prefix matching at that level. We construct various flat-tree networks based on generic Clos networks of different layouts. Table 4.2 lists the evaluated flat-tree topologies. We use topo-1 as the baseline topology and create other topologies by varying the network scale, oversubscription ratio, and arrangement of switches. topo-1 has 4:1 oversubscription at edge switches only. topo-2 is a proportional down-scale of topo-1. topo-3 is two times more oversubscribed at the edge than topo-1. topo-4 replaces the aggregation and core layers of topo-1 with fewer switches of larger port counts. topo-5 moves half of topo-1’s oversubscription to the aggregation switch level. topo-6 replaces the aggregation and core switches of topo-5 with larger ones. These topologies capture the major variations in Clos networks. We have flat-tree function in both global and local mode for each topology.

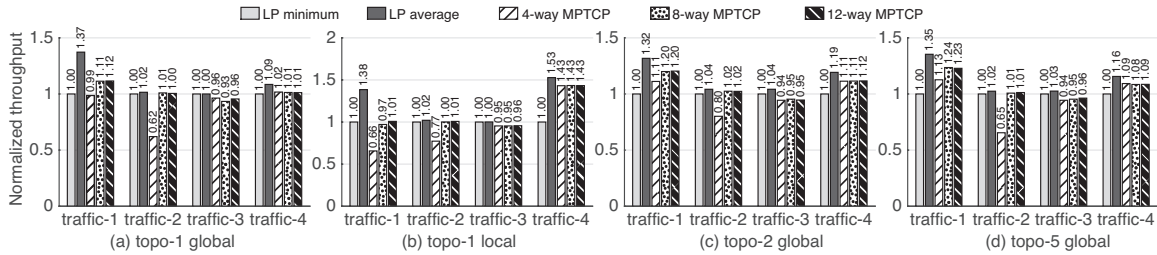


Figure 4.10 : Average flow throughput normalized against LP minimum on selected flat-tree topologies

A standard approach for evaluating routing in interconnection networks is to measure the throughput of flows given well-studied traffic patterns [116], so we use the following widely used synthetic traffic patterns to drive the simulation.

Permutation (traffic-1): every server sends a single flow to a unique server other than itself at random. This pattern creates uniform traffic across the network.

Pod Stride (traffic-2): every server sends a single flow to its counterpart in the next Pod. This traffic pattern creates heavy contention in the network core.

Hot spot (traffic-3): every 100 servers form a cluster, in which one server broadcasts to all the others. It simulates the multicast phase in many machine learning applications.

Many-to-many (traffic-4): every 20 servers form a cluster with all-to-all traffic. This traffic pattern simulates the shuffle phase in MapReduce jobs.

We also vary k , the number of concurrent paths in k -shortest-path routing, to evaluate the sensitivity of throughput against it. Given the above traffic, we compare the flow throughput from simulation to the optimum bandwidth allocation, which is the solution to the multi-commodity flow problem [114]. We make two linear programming (LP) formulations with different optimization goals: 1) maximizing

the minimum flow throughput (denoted as “LP minimum”) to achieve ideal load balancing; 2) maximizing the average flow throughput (denoted as “LP average”) to achieve best network utilization.

Figure 4.10 shows the average flow throughput on selected topologies, and the topologies not shown have similar trends. We normalize against LP minimum for each evaluated method for readability, as throughput numbers are vastly different in scale. The number of concurrent paths, k , affects the MPTCP performance. If k is too small, the path diversity cannot be fully exploited, thus many links are underutilized. In these experiments, 8 concurrent paths is sufficient, and larger k cannot improve the throughput further. This result is consistent with the performance of MPTCP and k -shortest-path routing in random graph networks [5].

k -shortest-path routing plus MPTCP reaches a reasonable middle ground between LP minimum and LP average. To scrutinize at the throughput of individual flows, we zoom in on topo-1 global mode and show the distribution of flow throughput with box plots in Figure 4.11. Neither LP minimum nor LP average is realistic. To balance the load among flows, LP minimum stops allocating residual bandwidth after it has successfully maximized the minimum flow throughput. LP average assigns some zero throughputs and some high or even full throughputs to maximize the network utilization. MPTCP balances between these extremes. It achieves higher average throughput than LP minimum, and the variance of flow throughput is smaller than LP average. Leveraging multi-paths and congestion control, MPTCP can dynamically adapt to the link utilization to get high throughput and maintain fair bandwidth sharing across flows.

From the above results, we conclude that k -shortest-path routing plus MPTCP is efficient enough. With the right choice of k , it constantly achieves comparable

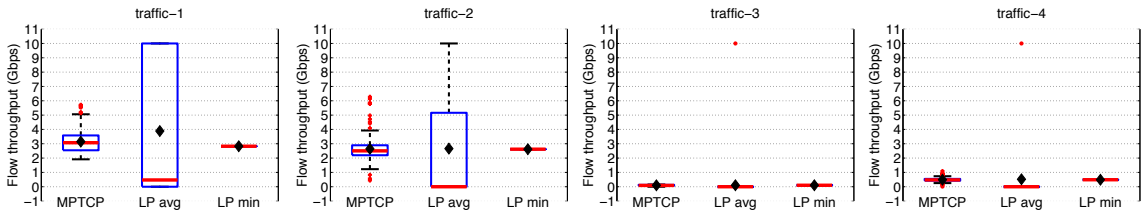


Figure 4.11 : Box plots to show the distribution of flow throughput on the topo-1 topology under flat-tree global mode (topo-1 global). MPTCP uses 12 paths. The box contains the 25th to 75th percentiles of the data. The whisker lines extending above and below the box cover the data within 3 times the box range. The data in dots beyond the whisker are outliers. The bold line in the middle of the box shows the median and the diamond shows the average.

throughput to optimal bandwidth allocation from LP solutions given a set of traffic patterns on diverse flat-tree topologies. It balances between high network utilization and load balancing, which are important indicators of good performance in practice.

4.5.4 Does flat-tree handle real traffic well?

To further evaluate the flat-tree performance with real data center traffic, we need a practical network topology. Large-scale data center network designs in recent years show the trend of non-blocking switch fabric with oversubscription only at the network edge [2, 3]. We follow this trend to use topo-1 as a representative flat-tree topology for the remaining simulations. We run traffic traces from 4 Facebook data centers each carrying different services. They are from the following two sources.

- 1) We obtain the one-hour trace in a Hadoop data center (denoted as Hadoop-1) from the Coflow benchmark [117], which contains aggregated rack-level traffic through a 1Gbps single-switch network core. Our flat-tree network uses 10Gbps links and has

8 uplinks per edge switch. For each rack-to-rack flow from the trace, we create 8 flows between servers under the source and destination edge switches to stress the switch uplinks and give 10 times the original traffic volume to each the 8 flows to adjust the bandwidth difference.

2) We obtain traffic statistics for 3 other data centers (denoted as Hadoop-2, Web, and Cache) from the Facebook measurement study [10]. The full traces are not released, so we generate our own traces based on the publicly shared sampling data [118] and the reported results from the paper [10]. The source and destination servers of the flows are inferred from the sampling data. The flow size and the flow arrival rate are reverse-engineered from Figure 6 and Figure 14 in the paper. We omit inter-data-center traffic in the data.

The traffic has the following characteristics.

Hadoop-1: the trace reflects the shuffle phase of MapReduce jobs. The traffic does not have clear locality. We observe one-to-many, many-to-one, and many-to-many traffic involving a large number of machines network-wide.

Hadoop-2: different from the above trace, the traffic shows strong rack and Pod level locality. 75.7% of the traffic is intra-rack, and almost all the remaining traffic is intra-Pod.

Web: the traffic has strong Pod level locality. There is a tiny amount of intra-rack traffic. Around 77% of the traffic is intra-Pod, and the rest is inter-Pod.

Cache: the traffic shows even stronger Pod level locality. There is almost zero intra-rack traffic. Around 88% of the traffic is intra-Pod, and the rest is inter-Pod.

Figure 4.12 shows the distribution of flow completion time of these different traffic traces. Regardless of the traffic, the performance of flat-tree in the global mode is close to that of the random graph, and the performance of flat-tree in the local mode

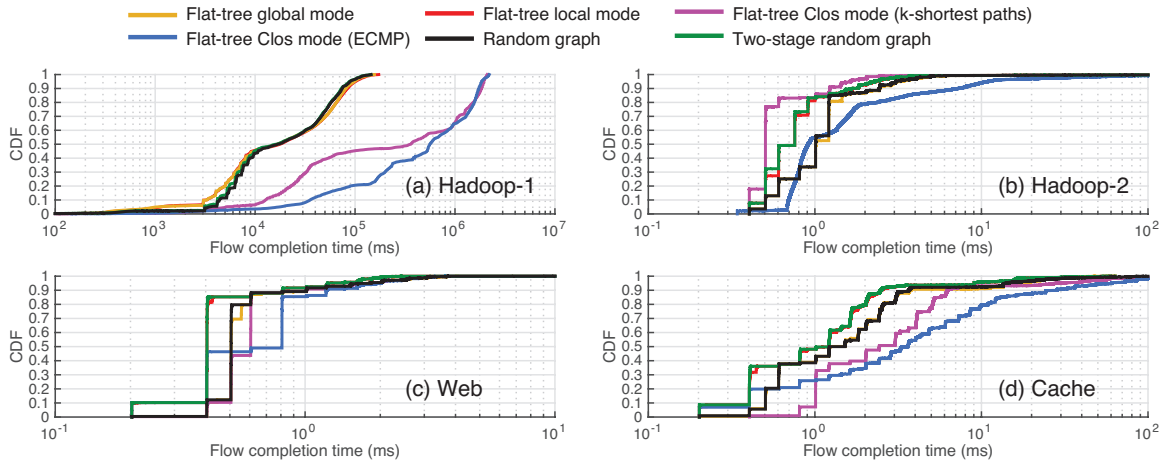


Figure 4.12 : CDF of flow completion time in Facebook’s Hadoop-1, Hadoop-2, Web, and Cache data centers

is close to that of the two-stage random graph. It demonstrates that flat-tree well approximates random graphs of different scales given real data center workloads, which is consistent with the conclusion in the prior study [94].

In practice, Clos networks usually implement ECMP and TCP. For fair comparison, we simulate the flat-tree Clos mode with k -shortest-path routing and MPTCP as well to avoid the handicap of less efficient routing and congestion control mechanisms. As expected, the performance of flat-tree Clos mode with ECMP and TCP is remarkably worse than the other networks. For ECMP, the next hop at each switch is determined pseudo-randomly by header field hashing, so each TCP flow traverses only one of the equal cost shortest paths. Being unable to use multi paths concurrently is especially bad for large flows. In later discussions, we focus on the Clos mode with k -shortest path routing and MPTCP.

Most importantly, we observe that different modes of flat-tree are best suited for different types of traffic. In Figure 4.12(a), for the network-wide traffic, flat-tree

global mode has an order of magnitude improvement over the Clos network. Flat-tree local mode has similar performance to the global mode for two reasons. First, the traffic is not intensive enough to saturate the links on these topologies, although the Clos network is already heavily congested. Second, there is a considerable amount of intra-Pod traffic in the network-wide traffic. Since the global mode has richer core bandwidth than the local mode, we expect greater benefit from the global mode given heavier traffic and more inter-Pod communications.

In Figure 4.12(b), the performance of flat-tree Clos mode is the best due to the large proportion of intra-rack traffic. Flat-tree local mode is the second best, because the topology handles intra-rack traffic relatively well and there is still around 24.3% intra-Pod traffic. Flat-tree global mode is not very efficient for traffic with strong locality. For traffic with Pod-level locality, as shown in Figure 4.12(c) and (d), flat-tree local mode has the best performance, followed by the global mode and the Clos mode. This result reflects the distribution of network bandwidth. The global mode has less intra-Pod bandwidth than the local mode, but the rich network-wide bandwidth makes it more efficient than the Clos network. The difference among topologies is more significant in Figure 4.12(d) due to stronger locality and higher traffic volume.

These simulation results of real data center traffic on a practical data center topology validate the design purpose of flat-tree. Flat-tree can be configured into different modes to optimize traffic with different locality features, i.e. Clos mode for rack-level locality, local mode for Pod-level locality, and global mode for no locality. If the network is used for a different service, the network topology can be easily reconfigured to adapt to the new traffic. This flexibility in topology is particularly useful for public clouds where the service requirements are constantly changing. For a production data center like Facebook with integral parts of different services, flat-tree

can be used in the hybrid mode with various service-specific zones, interconnected by the network core for inter-zone communication. When the services are reorganized, the network zones can be repartitioned to accommodate the change of needs.

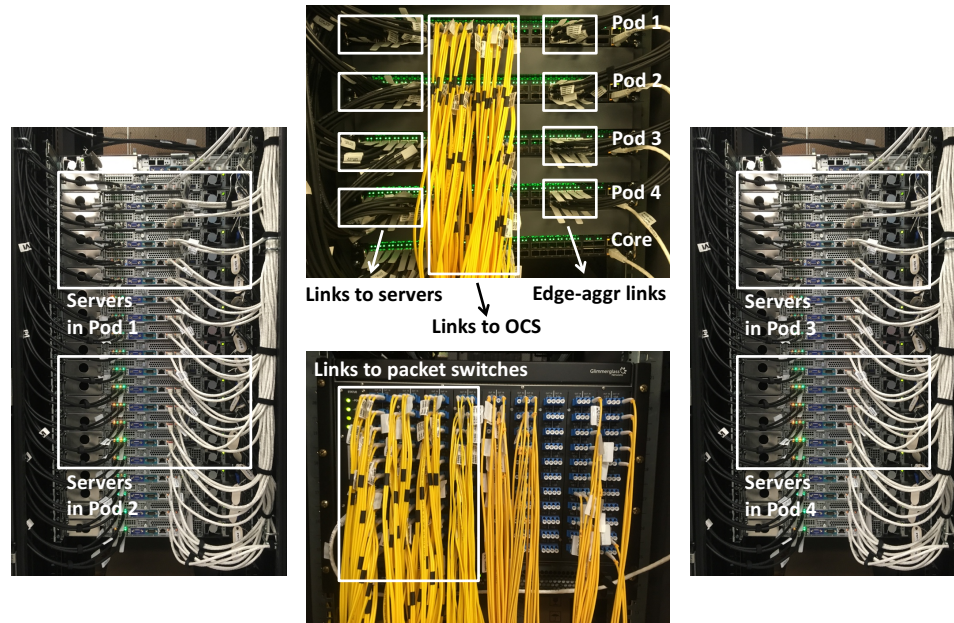


Figure 4.13 : A testbed implementing the flat-tree example in Figure 4.2

4.5.5 Is flat-tree implementable?

To explore the feasibility of implementing flat-tree in practice, we build the example network in Figure 4.2 on a hardware testbed. As shown in Figure 4.13, it consists of 5 48-port packet switches, one 192-port 3D-MEMS optical circuit switch (OCS), and 24 servers each with 6 3.5GHz dual-hyperthreaded CPU cores and 128GB RAM. All links are 10Gbps. The first 4 packet switches are partitioned into edge and aggregation switches in each Pod, and the last packet switch is partitioned into the 4 core switches. The converter switches are logical partitions of the OCS. To make the testbed more manageable, we connect servers to converter switches via an extra hop on packet switches.

We implement k -shortest-paths routing and MPTCP for all 3 flat-tree topologies. k is set to 4 as it yields the best performance in the simulation of this network. We

realize the addressing scheme as described in Section 4.4.2. Our packet switches use a legacy OpenFlow 1.0 image. It does not support arbitrary bits matching of a field, which is necessary for source routing as shown in Section 4.4.2. So, we conduct prefix matching for the source and destination IP addresses on the switches a path traverses. The maximum number of OpenFlow rules per switch under each topology is 242, 180, and 76 respectively. The difference is due to the different number of ingress/egress switches in each topology. With source routing, these numbers will be significantly less.

We demonstrate the functionality of the testbed with an iperf throughput experiment. On every server, we send iperf traffic to the 3 servers with the same index in the other 3 Pods. This traffic pattern enables the measurement of the core bandwidth in the network. iperf is set to update the flow throughput every 0.5 second. Throughout the 5-minute experiment, we change the network topology to different flat-tree modes. We add up the throughputs of individual flows to obtain the real-time bidirectional core bandwidth.

Figure 4.14 plots the variation of core bandwidth as we change the network topology. The local mode and the Clos mode have around 145Gbps average total throughput. Compared to the Clos mode, the local mode rearranges servers within Pods only, so there is no change to the core bandwidth. Our testbed is 1.5:1 oversubscribed, so the Clos network has $24 \times 10\text{Gbps} / 1.5 = 160\text{Gbps}$ total bandwidth. This result shows that the overhead of MPTCP and k -shortest-paths routing is within 9.38% of the bandwidth, which is reasonable as the MPTCP packet processing lays extra burden on CPU and k -shortest-paths routing is not perfect. The average total throughput in the global mode is around 185Gbps. With the power of convertibility, the network core bandwidth increases by 27.6% in this small testbed. We envision

Table 4.3 : Conversion delay of the throughput experiment in Figure 4.14

Topology	OCS configuration	Rule deletion	Rule installation	Total
Global	160ms	477ms	644ms	1281ms
Local	160ms	202ms	482ms	844ms
Clos	160ms	635ms	209ms	1004ms

greater improvement in real data centers with a larger number of switches and more flexibility of conversion.

From the figure, we observe that iperf reaches the maximum throughput in 2s to 2.5s after topology change. Table 4.3 shows the accurate measurement of the conversion delay by the control software. The delay can be broken down into the time for reconfiguring the OCS, deleting old OpenFlow rules, and adding new OpenFlow rules. The rule deletion and installation delay are proportional to the number of rules for the topology before and after conversion. Our implementation has room for improvement. First, the legacy switches process rules more slowly than the main-stream technology [112, 113]. Second, the packet switches and the OCS are configured sequentially, and they can be easily parallelized. Even with these artifacts, the network topology can be converted in roughly 1s and the application adapts to the topology change in another 1.5s.

4.5.6 Does convertibility benefit applications?

Most data center applications are computation-oriented, inter-node communications serving the purpose of exchanging intermediate computation data. For this reason, the behavior of data transmission is influenced by many factors in the computation

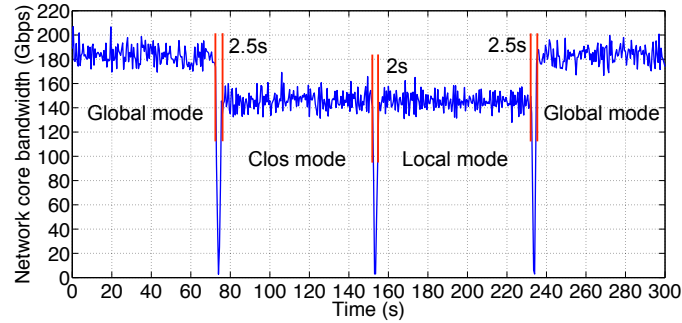


Figure 4.14 : Summation of iperf throughput every 0.5 second on the testbed with the variation of flat-tree modes. Every server sends iperf traffic to its counterparts in the other Pods to saturate the network core. Traffic adapts to the topology change in 2 to 2.5 seconds.

framework. For instance, read/write data serialization/deserialization adds to the end-to-end data transmission time; imperfect synchronization of computation nodes disorganizes traffic patterns; garbage collection may block communications, etc. In our testbed, converting the network topology from the Clos mode to the global mode improves the core bandwidth by 27.6%. However, with all these overheads from the computation framework, whether the bandwidth increase can be translated into acceleration of data center applications is yet another question.

We answer this question by running Spark and Hadoop, the most widely used computation frameworks, on our testbed. Among the 24 servers, we set the first server as the master node and all the other servers as slave nodes. We change the network topology and compare the end-to-end data read time under different flat-tree modes. The characteristics of the jobs are as follows.

Spark broadcast: we run Word2Vec, the iterative machine learning job for document feature extraction. In each iteration, the master node broadcasts the updated

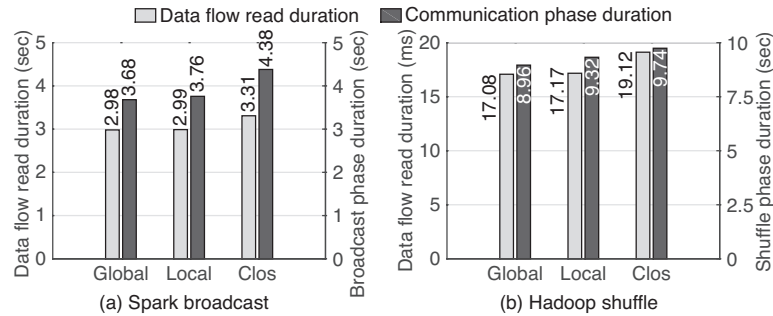


Figure 4.15 : Average data flow read duration (left y-axis) and average communication phase duration (right y-axis) in the Spark broadcast and Hadoop shuffle applications under different flat-tree topology modes

model to all workers. We choose the “torrent” option for the broadcast operation to distribute the data in the BitTorrent fashion. Spark promotes in-memory computation, so the data to be transmitted is readily available in memory, although data serialization and deserialization are needed.

Hadoop shuffle: we run the Sort job on Tez [119], a variant of Hadoop MapReduce. It has a heavy shuffle phase, where all the nodes as mappers send data to a subset of nodes as reducers. We store the data on a RAM disk to prevent the hard drive being the bottleneck of data read/write.

Figure 4.15 shows the average end-to-end data read time and the duration of the communication phase for the above two applications. The end-to-end data read time includes the time for data serialization and deserialization. In the Spark broadcast application, flat-tree global mode reduces the average data read time by 10% and reduces the broadcast phase duration by 16% compared to the Clos topology. In the Hadoop shuffle application, the reduction in the average data read time and in the shuffle phase duration are 10.5% and 8% respectively. With this visible difference,

we conclude that the improvement of network topology is reflected in the application performance. The global mode only slightly outperforms the local mode, because their network structures are not hugely different at this small scale (Figure 4.2 c vs. d). Both applications have many-to-many traffic pattern, which is similar to the Hadoop-1 trace in the simulation. As shown in Figure 4.12(a), flat-tree global mode performs better than the local mode by 36.08% and better than the Clos mode by an order of magnitude. From this evidence, we expect more considerable performance improvement to applications from the change of topology in a large-scale data center.

4.6 Summary

Flat-tree is the first effort towards building convertible data center networks. By converting between Clos and approximate random graph of various scales, it achieves the conventionally conflicting goals of low implementation complexity and high transmission throughput. Convertibility can be achieved by a set of small port-count converter switches distributed across the network. They have low cost and can be packaged into Pods to ease deployment. We find flattening Clos' tree structure does not require global rewiring. With regular wiring patterns between Pods and core switches and simple connections between adjacent Pods, we effectively approximate randomness in the network core and at the same time obtain low wiring complexity. Multi-path routing and congestion control are crucial to exploiting the path diversity in flat-tree, and we have shown that aggregation strategies can be applied to avoid an explosion of network states. Existing routing and transport protocols combined with our architecture-specific state aggregation schemes can balance between high network utilization and fair bandwidth sharing among flows. We explore the implementability of flat-tree using simulations with real data center traffic and a testbed implementation of the system. We observe flat-tree can optimize for diverse workloads with different topology modes, and it brings performance improvements to applications with greater core bandwidth. Flat-tree is merely one design point in the broad space of convertible data center networks. We believe our experience will motivate future studies on convertibility.

Chapter 5

ShareBackup: Enabling Sharable Backup for Failure Recovery in Data Center Networks

In this chapter, we continue the study on convertible data center networks with the emphasis on enhancing fault tolerance of the network. We propose ShareBackup to improve reliability with the concept of “shareable backup”, which allows the network to share a small pool of backup switches that can be brought online instantaneously to recover from failures. We organize switches into failure groups. Through the same connectivity to a set of converter switches, these switches share one or more backup switches. Link failures are addressed as node failures on both ends, and we use offline failure diagnosis to understand the cause of problem and to recycle healthy switches. We use distributed network controllers to share the burden of failure detection and recovery. For fast failure recovery, we support live impersonation of the failed switches on the control plane. Using market prices, the cost of ShareBackup is multi-fold lower than state-of-the-art failure-resilient architectures. It also provides more bandwidth compared to rerouting-based solutions. The detailed design is presented in the following sections.

5.1 Motivation

As the underlying infrastructure for cloud computing, data center networks should provide high reliability to guarantee service performance. The mainstream solution to fault tolerance is rerouting: many data center network architectures provide re-

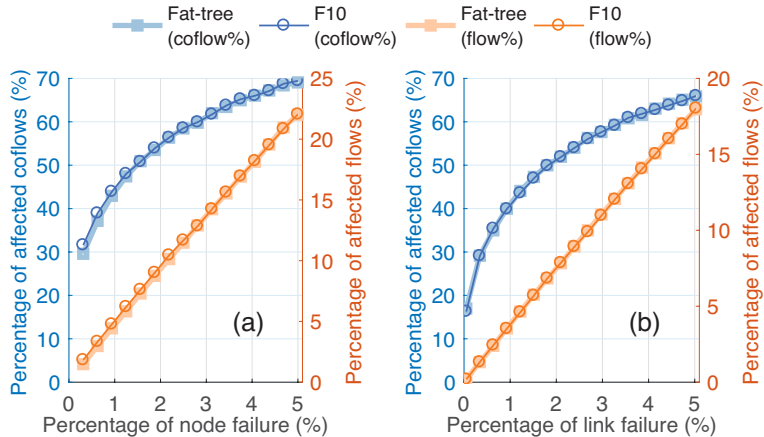


Figure 5.1 : Impact of failures on flows and coflows

dundant paths to increase bandwidth, and alternative paths can be used to reroute traffic around failures [1, 5, 11–13, 51–54]. While rerouting maintains connectivity, bandwidth is nonetheless degraded under failures, which may jeopardize application performance drastically.

This concern has been validated by a measurement study: in a path-rich production data center, 10% less traffic is delivered for the median case of the analyzed failures, and 40% less for the worst 20% of failures [120]. Because data center traffic is expressed as sets of flows—known as coflows—to capture the application-level requirements [20], a small number of straggler flows influenced by failures can prolong the duration of the entire coflow. The effect of failure is thus exacerbated on the coflow level. The following experiment shows this phenomenon.

We run the coflow trace of real data center traffic [117] on packet-level simulators of the fat-tree [1] and F10 [53] networks, two widely used fault-tolerant data center architectures. The trace contains aggregated rack-level traffic from a 150-rack 10:1 oversubscribed network, so we map the traffic to similar-sized $k = 16$ fat-tree and F10

networks with the same oversubscription ratio at the edge switches. Fat-tree and F10 both use ECMP routing. Under failures, fat-tree uses global optimal rerouting, and F10 uses its local three-hop rerouting [53]. We simulate the final states after failures without the transient dynamics and measure the percentage of flows and coflows affected by failures with the variance of failure rate. A flow is considered affected if it traverses a failed node or link, and a coflow is affected if at least one flow in its set gets affected.

We observe from Figure 5.1 that the impact of failures gets magnified significantly on the coflow level. The percentage of affected coflows is $3.3\times$ to $90\times$ that of individual flows. The coflow curves climb faster in the beginning, indicating a small number of failures have huge impact on applications. With only a single node and link failure, as many as 29.6% and 17% of coflows are affected respectively. An application can proceed only after an entire coflow has finished, so occasional failures have devastating harm to application performance. These results show necessity of repairing failures rapidly rather than rerouting traffic to mitigate its effect.

The only approach that can restore network to its full capacity immediately after failures happen is backup. Switches can keep a hot spare; hosts are multi-homed to the primary and the backup switches; and every link between two primary switches is duplicated by a mesh amongst them and their shadows. In this way, a switch can failover to its spare without bandwidth loss in the network. However, this 1:1 backup consumes a large number of backup switches and doubles the port requirements on hosts and switches. The prohibitive cost of extra hardware prevents the deployment of network-wide backup, so most data centers only backup a few crucial devices.

Two recent trends provide new opportunities for network-wide backup. First, commercial devices in data centers are increasingly more reliable. Despite the dis-

astrous effect, the same measurement study shows failures are rare and transient: most devices have over 99.99% availability; and failures usually last for only a few minutes [120]. Therefore, 1:1 backup is unnecessary for occasional failure events. Second, configurable interconnects can facilitate physical-layer adaptation of the network topology, and novel architectures have been proposed to create paths on the fly according to traffic requirements [31–40]. This configurability can be repurposed for efficient failure recovery.

In this chapter, we introduce *shareable backup*, where a small pool of backup devices can repair failures on demand. This solution is both desirable and achievable from the above evidence. By connecting a group of switches and a few backup switches to highly reliable configurable interconnects, e.g. circuit switches, a backup switch can be brought online to replace any failed switch via simple circuit reconfiguration.

We realize this idea in ShareBackup*, a prototype failure recoverable fat-tree network. We focus on fat-tree because fat-tree and its variants are widely adopted by many industrial data centers [2, 3]. The ShareBackup design faces many challenges. The per-port cost of small circuit switches are considerably lower than large ones [32, 42]. How to design the architecture to leverage this cost benefit? Switches need to be grouped together to share backup switches. How to partition switches into groups so that all switches are covered by backup switches and the placement of circuit switches can align with the fat-tree packaging and wiring? Link failures can be treated as failures of the associated switches, but it is hard to determine which end has lost connectivity. Instead of replacing switches on both sides of the link, how to diagnose the problem and only replace the faulty switch? Backup switches can replace regular switches on the physical layer, but how to impersonate a replaced switch

*The name is inspired by the emerging trend of Shareconomy.

Table 5.1 : List of notations

Notation	Meaning
k	Fat-tree parameter: switch port count and # Pods [1]
n	# backup switches shared by $\frac{k}{2}$ switches per failure group
H_j	The j th host
$E_{i,j}$	The j th Edge switch in the i th Pod
$A_{i,j}$	The j th Aggregation switch in the i th Pod
C_j	The j th Core switch
$CS_{l,i,j}$	The j th Circuit Switch in the i th Pod on the l th layer
$FG_{l,u}$	The u th Failure Group on the l th layer
$BS_{l,u,v}$	The v th Backup Switch in $FG_{l,u}$
UP_p	The p th UPward facing port of a circuit switch
$DOWN_p$	The p th DOWNward facing port of a circuit switch

on the control plane? Moreover, how to avoid extra delay from the impersonation process so that ShareBackup can recover failures as fast as the highly responsive local rerouting? We address these challenges and analyze the properties of our proposed architecture. Compared to rerouting-based solutions, ShareBackup provides more bandwidth with short path length at low cost.

5.2 Network Architecture

Algorithm 1 Wiring algorithm

```

// Edge layer
1: for each  $CS_{1,i,j}$  where  $0 \leq i < k$ ,  $0 \leq j < \frac{k}{2}$  do
2:    $E_{i,j} \in FG_{1,i}$ 
3:   for each  $p$  where  $0 \leq p < \frac{k}{2}$  do
4:      $DOWN_p \longleftrightarrow H_{\frac{k}{2} \times p + j + i \times (\frac{k}{2})^2}$ 
5:      $UP_p \longleftrightarrow E_{i,p}$  // External connection
6:      $DOWN_p \dashrightarrow UP_p$  // Internal connection
7:   for each  $p$  where  $\frac{k}{2} \leq p < \frac{k}{2} + n$  do
8:      $UP_p \longleftrightarrow BS_{1,i,p-\frac{k}{2}}$ 

// Aggregation layer
9: for each  $CS_{2,i,j}$  where  $0 \leq i < k$ ,  $0 \leq j < \frac{k}{2}$  do
10:   $A_{i,j} \in FG_{2,i}$ 
11:  for each  $p$  where  $0 \leq p < \frac{k}{2}$  do
12:     $DOWN_p \longleftrightarrow E_{i,p}$ 
13:     $UP_p \longleftrightarrow A_{i,p}$ 
14:     $DOWN_p \dashrightarrow UP_{(p+j)\% \frac{k}{2}}$ 
15:  for each  $p$  where  $\frac{k}{2} \leq p < \frac{k}{2} + n$  do
16:     $DOWN_p \longleftrightarrow BS_{1,i,p-\frac{k}{2}}$ 
17:     $UP_p \longleftrightarrow BS_{2,i,p-\frac{k}{2}}$ 

```

Algorithm 1 Wiring algorithm (continued)

```

// Core layer
18: for each  $CS_{3,i,j}$  where  $0 \leq i < k$ ,  $0 \leq j < \frac{k}{2}$  do
19:    $C_{\frac{i}{2} + \frac{k}{2} \times j} \in FG_{3, \frac{i}{2}}$ 
20:   for each  $p$  where  $0 \leq p < \frac{k}{2}$  do
21:      $DOWN_p \leftarrow A_{i,p}$ 
22:      $UP_p \leftarrow C_{\frac{k}{2} \times p + j}$ 
23:      $DOWN_p \leftarrow\!\!\!\rightarrow UP_p$ 
24:   for each  $p$  where  $\frac{k}{2} \leq p < \frac{k}{2} + n$  do
25:      $DOWN_p \leftarrow BS_{2,i,p-\frac{k}{2}}$ 
26:      $UP_p \leftarrow BS_{3,j,p-\frac{k}{2}}$ 

```

ShareBackup has stringent requirements on cost and failure recovery delay, which guide our choice of circuit switch technologies. No existing circuit switch has enough ports to connect to all switches in the data center plus the pool of backup switches. Cascading multiple circuit switches wastes many intermediate ports, increases insertion loss thus requiring more powerful (and expensive) transceivers, and causes large end-to-end switching delay that slows down failure recovery. Instead, recent works promote partial configurability in small network regions using circuit switches with considerably low per-port cost and switching delay [72, 121]. For instance, a commercial 160-port 10Gbps electrical crosspoint switch costs \$3 per port and has 70ns switching delay [72]; 32-port 25Gbps optical 2D-MEMS has been developed, with 40 μ s switching delay at an estimated cost of \$10 per port [42, 122]. These technologies meet our demand.

These targeted circuit switches have modest port count, so we divide the network into smaller failure groups and deploy them in each group. Measurement studies show that failures in data centers are rare, uncorrelated, and spatially dispersed [120, 123]. ShareBackup’s distributed design is a good match for these characteristics and can provide good coverage. Fat-tree has $\frac{k}{2}$ edge and aggregation switches per Pod. To align with the architecture, we cluster $\frac{k}{2}$ switches into a failure group and allow them to share n backup switches. All switches in a failure group, including the backup switches, must connect to the same circuit switch with the same wiring pattern. In this way, a backup switch can be brought online at runtime to replace a failed switch or failed links associated with it. This circuit switch should have at least $\frac{k}{2} \times (\frac{k}{2} + n)$ ports, which exceeds the port count of the targeted circuit switches for a large data center. We combine $\frac{k}{2}$ individual circuit switches side by side and design a wiring pattern to achieve equivalent functionality. Figure 5.2 and Figure 5.3 give intuitions

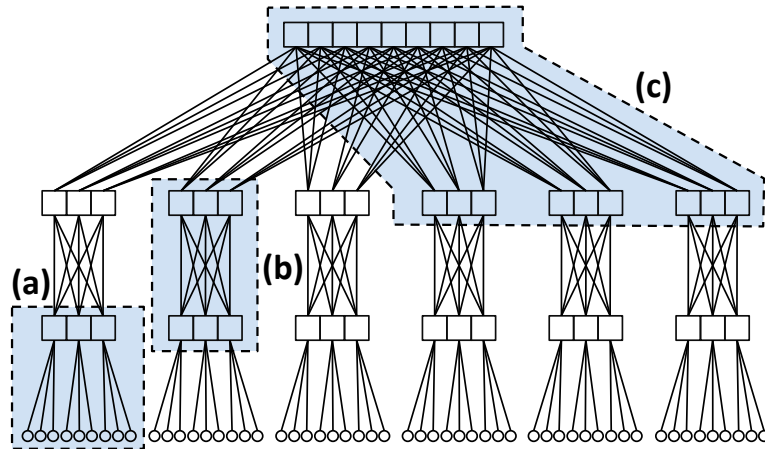


Figure 5.2 : A $k = 6$ fat-tree [1]. To build a ShareBackup network from it, the blocks of devices like in the shaded areas should be replaced by the corresponding structures in Figure 5.3.

of the architecture design. Algorithm 1 shows the wiring plan, with notations listed in Table 5.1.

Figure 5.3(a) illustrates the basic building block for ShareBackup. The edge switches in the same Pod form a failure group (*line 2* in Algorithm 1). We place $\frac{k}{2}$ units of $(\frac{k}{2}+n+2)$ by $(\frac{k}{2}+n+2)$ circuit switches between the edge switches and the hosts. Every switch, regular and backup switch alike, connects to these $\frac{k}{2}$ circuit switches each with a link (*line 5 and 8*). As shown in Figure 5.5, these $\frac{k}{2}$ switches are chained together via 2 side ports, which is omitted in Figure 5.3 for simplicity. Hosts connect to the edge switches via straight-through connections on the intermediate circuit switches (*line 4 and 6*). The ports to backup switches are unconnected internally. When a switch is down, the internal connections to it on all the circuit switches are reconfigured to connect to a backup switch, which thus replaces the failed switch completely. A switch whose links are down is replaced in the same manner so as to

fix the link failures.

In Figure 5.3(b), the aggregation switches in the same Pod form a failure group (*line 10*). Edge and aggregation switches in their failure groups repeat the building block of connectivity in Figure 5.3(a) to another set of $\frac{k}{2}$ circuit switches (*line 12, 13, 16, and 17*). In a fat-tree Pod, an edge/aggregation switch connects to each and every aggregation/edge switch, so we use a rotational wiring pattern in the circuit switches (*line 14*) to achieve this shuffle connectivity, i.e. the different internal connections on $CS_{2,1,0}$ to $CS_{2,1,2}$.

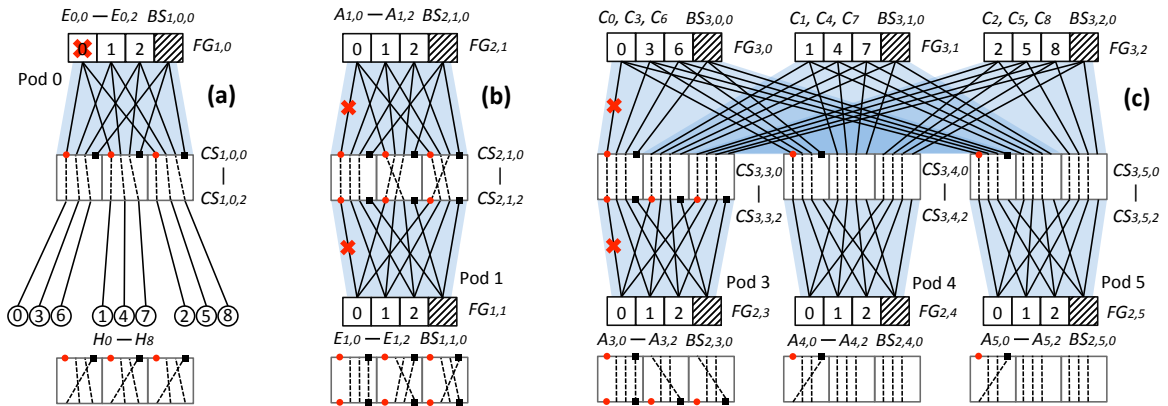


Figure 5.3 : Substructures of a ShareBackup network where $k = 6$ and $n = 1$. The subfigures correspond to the shaded areas in Figure 5.2. Devices are labeled according to the notations in Table 5.1. Edge and aggregation switches are marked by their in-Pod indices; core switches and hosts are marked by their global indices. Switches in the same failure group are packed together, which share a backup switch in stripe on the side. Circuit switches are inserted into adjacent layers of switches/hosts. The connectivity in shade is the basic building block for shareable backup. The crossed switch and connections represent example node and link failures. Switches involved in failures are each replaced by a backup switch with the new circuit switch configurations shown at the bottom, where connections regarding the original red round ports reconnect to the new black square ports.

Similarly, aggregation switches in each failure group shown in Figure 5.3(c) are connected upward to $\frac{k}{2}$ circuit switches with the wiring pattern in the building block (*line 21 and 25*). As Figure 5.2 shows, the connections from aggregation switches in each Pod iterate through all the core switches in consecutive order. Because the aggregation switches are already connected to the circuit switches, we set wiring between the core switches and the circuit switches to achieve the fat-tree connectivity. The core switches connect to $\frac{k}{2}$ circuit switches with a stride of $\frac{k}{2}$, and we set straight-through connections in the circuit switches (*line 22 and 23*). Based on the building block for shareable backup, only switches connected to the same set of circuit switches can be put into a failure group. As a result, core switches whose indices are in $\frac{k}{2}$ intervals form a failure group (*line 19*). We give each failure group n backup switches and connect them up in the same way as regular switches (*line 26*).

In fat-tree, edge and aggregation switches are packaged into Pods for ease of deployment. In each ShareBackup Pod, there are n additional edge and aggregation switches respectively as backup switches, and 3 sets of $\frac{k}{2}$ circuit switches between adjacent layers of switches and hosts. It is straightforward to package the backup switches and the circuit switches into the original fat-tree Pods with simple changes of wiring as shown in Figure 5.3. In practice, the core switches can be placed as in the original fat-tree, with the backup core switches added to the end. The reordering depicted in Figure 5.3 is unnecessary. By streamlining the connectors from within each Pod, we can maintain the original Pod-host and Pod-core wiring patterns in fat-tree.

5.3 Control Plane

5.3.1 Fast Failure Detection and Recovery

Most previous fault-tolerant data center network architectures, such as PortLand [52] and F10 [53], mainly focus on link failures. According to a measurement study, however, switch failures account for 11.3% of the failure events in data centers and their impact is significantly more severe than link failures [120]. Therefore, ShareBackup aims to detect and recover both link and switch failures rapidly.

Failure detection and recovery are handled by a management entity, e.g. one or more dedicated machines running specific processes. For switch failures, we require switches to send keep-alive messages continuously to the management entity. After missing keep-alive messages from a switch for a pre-defined time period, the management entity allocates an available backup switch to failover to and reconfigures the circuit switches associated with the failure group. As shown in Figure 5.3(a), in these circuit switches, original connections to the failed switch should reconnect to the backup switch.

We adopt the rapid failure detection mechanism in F10 [53] for link failures, where switches keep sending packets to each other (or to hosts) to test the interface, data link, and forwarding engine. When a link is down, it takes time to determine which end has lost connectivity. For the purpose of fast failure recovery, the switches on both sides of the failed link are replaced. The cause of failure is analyzed later by the procedure in Section 5.3.3. The management entity gets notifications of link failures from switches and hosts, and reconfigures the circuit switches in the same way as it addresses switch failures on both ends. Figure 5.3(b) and 5.3(c) show examples of this approach.

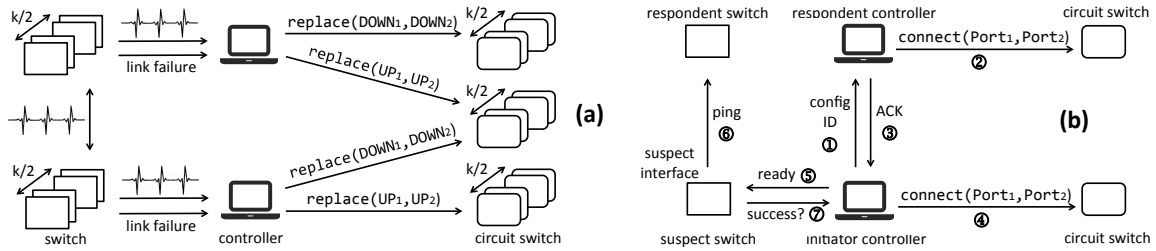


Figure 5.4 : Communication protocol in the control system. (a): Failure detection and recovery. (b): Diagnosis of link failure.

5.3.2 Distributed Network Controllers

ShareBackup requires the management entity to be implemented as distributed network controllers. A single controller is not capable of collecting frequent heartbeats from all the switches in the network. Like F10, the probing interval for failure detection can be as low as a few ms. For example, in a $k = 48$ fat-tree with 2880 switches, a heartbeat message every 5ms leads to 576k queries per second, which exceeds the capacity of a single controller. Distributed controllers also isolate the impact of controller failures to a small portion of the network. Controllers only store local state, so adding redundant controllers can further enhance resiliency with low state-exchange overhead. Finally, with distributed placement of network controllers, switches and circuit switches can be physically close to their controller, which effectively reduces the message latency of failure detection and recovery.

Figure 5.4(a) shows the failure detection and recovery process using distributed network controllers. Based on the layout of the ShareBackup architecture, an intuitive way of controller placement is to assign each failure group a dedicated controller. Each controller receives heartbeat messages from $\frac{k}{2}$ switches only. As shown in Figure 5.3, a failure group in the core layer corresponds to $\frac{k}{2}$ circuit switches beneath

it, while one in the edge and aggregation layers corresponds to two sets of $\frac{k}{2}$ circuit switches above and beneath it, so each controller reconfigures k circuit switches at maximum. The load for each controller is thus very light even for a large network. The controllers do not share state, so the communication among them is also minimal. Due to the simple functionality, controllers can be realized by low-cost, bare-minimum computing hardware, such as the Arduino [124] and Raspberry Pi [125] platforms. Multiple controllers can also reside on the same machine to realize different degrees of distribution/centralization.

Most circuit switches nowadays use the TL1 software interface to setup a connection, whose input and output ports should be specified explicitly, i.e. *connect(input_port, output_port)*. The network controller needs to maintain the current connections of the circuit switches so as to switch to new connections. In Figure 5.3(b) and 5.3(c), a circuit switch connects to two switches from the failure groups above and below, so it can be reconfigured by both controllers. After one controller updates the circuit configuration, the other is ignorant of the change. The other controller may later use outdated port information and mess up the connections. To address this problem, we change the interface function to *replace(old_port, new_port)* and free controllers from bookkeeping of the circuit switch configurations. After this change, network controllers reconfigure circuit switches by two parameters: the old port to the failed switch and the new port to the backup switch (the red round and black square ports in Figure 5.3), from which circuit switches resolve the new connections to change to. Requests from different controllers relate to ports on opposite sides of the circuit switch. In case of concurrent requests, the circuit switch reconfigures one side at a time, so the order of execution does not affect the end result.

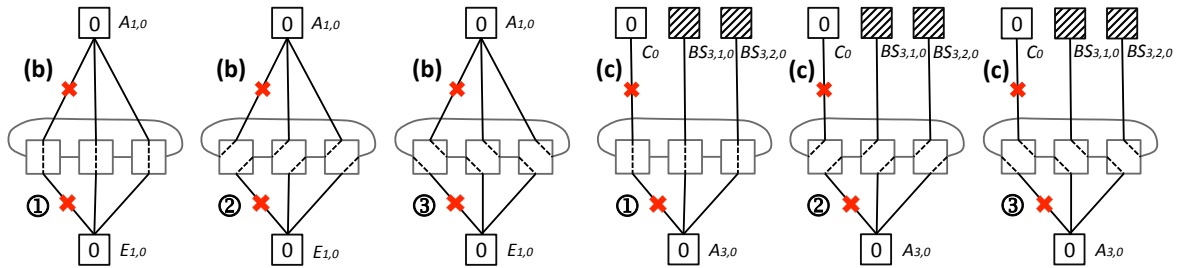


Figure 5.5 : Circuit switch configurations for diagnosis of link failures shown by examples (b) and (c) in Figure 5.3. Circuit switches in a Pod are chained up using the side ports. Only “suspect switches” on both sides of the failed link and some related backup switches are shown. Through configurations ①, ②, and ③, the “suspect interface” on both “suspect switches” associated with the failure can connect to 3 different interfaces on one or multiple other switches.

5.3.3 Offline Failure Diagnosis

Most link failures are due to malfunction of the network interface on one end. After both switches are replaced, we run failure diagnosis in the background to find which “suspect interface” (and the “suspect switch” it belongs to) has caused the problem. We chain up circuit switches in the same layer of a Pod as a ring through the side ports. Figure 5.5 shows the circuit switch configurations, through which the suspect interface on either end of the failed link can connect to 3 different interfaces, either on the same switch (as $A_{1,0}$, $E_{1,0}$, and C_0) or on different switches (as $A_{3,0}$).

The network controllers for the involved switches coordinate to change the circuit switch configurations and enforce the switches to exchange testing messages. A suspect interface that has connectivity in at least one configuration is redressed as healthy, so is the corresponding suspect switch. Because failure diagnosis only involves suspect switches already taken offline and backup switches not in use, this

process is completely independent of the functioning network.

Figure 5.4(b) illustrates the controller coordination process. The two suspect interfaces are tested one by one. Their corresponding controllers elect one to be the initiator and the other one as passive respondent. The initiator cycles through the configurations to test the suspect interface on its side, after which the initiator and the respondent reverse roles to test the other suspect interface. As shown in Figure 5.4(a), in our distributed control system, each controller is responsible for reconfiguring a small subset of circuit switches and is only allowed to control the ports on its own side. So, both controllers need to participate in the circuit setup. The respondent controller learns the target connections from the initiator controller via the configuration ID. Here, we use the original TL1 interface, i.e. *connect(input_port, output_port)*, to connect to the side ports. As an offline process, failure diagnosis can be preempted by failure recovery. It is paused if the involved backup switch, such as $BS_{3,1,0}$ and $BS_{3,2,0}$ in Figure 5.5(c), needs to be used when another failure happens. The initiator controller thus proceeds only after receiving confirmation that the respondent side is not being preempted at the moment. It will continue with the next configuration if reaching the ACK timeout. In the end, the tested interface pings the other end and terminates the diagnosis process if it has connectivity.

Failure diagnosis requires both sides have at least one healthy interface, so that both suspect interfaces can be tested. If this condition is not met, both suspect switches are considered faulty. Since all hosts are actively in use, the offline failure diagnosis is not supported between hosts and edge switches. We assume switches are at fault for link failures to hosts. If the problem is not fixed after replacing the switch, we mark the switch as healthy and trouble-shoot the host.

After a failed switch is repaired or a suspect switch is exonerated, it is unnecessary

to switch back to the original connectivity. Backup switches and regular switches are equal in functionality, so we keep the backup switch online and turn the replaced switch into a backup switch for future use. This design saves the reconfiguration overhead and avoids disruptions in the network. The network controller keeps track of the current backup switches in their failure groups.

5.3.4 Live Impersonation of Failed Switch

Traffic will be redirected to the backup switch in the physical layer after a failed switch is replaced. The backup switch needs to impersonate the failed switch by using the same routing table. Fat-tree uses Two-Level Routing, where each switch has a pre-defined routing table with k entries [1]. To avoid the additional delay of inserting forwarding rules into the backup switch, we aim to preload the routing table and make the backup switch a hot standby. Regular switches recovered from failures can work as backup switches, so every switch needs to store the routing tables of all the switches in the failure group. The challenge is to resolve the conflicts between different routing tables.

In fat-tree, all the core switches and all the aggregation switches in the same Pod have the same routing table. Therefore, in the aggregation and core layers of our network, switches in a failure group only keep a common routing table. For in-bound traffic, edge switches in a Pod, also a failure group, have the same set of $\frac{k}{2}$ forwarding entries that match on the suffix of the end host addresses. For out-bound traffic, each of these edge switches has $\frac{k}{2}$ different entries. We use VLANs for differentiation. We first edit the original fat-tree routing tables by assigning every edge switch in the Pod a unique VLAN ID and adding it to the out-bound routing table entries. The edited routing tables from all the edge switches are then combined together and stored in

every switch in the failure group. A host knows which edge switch it should connect to, so it tags out-going packets with the VLAN ID of the edge switch. No matter what switches in the failure group are active, by matching the VLAN ID, packets can always refer to the correct routing table. This combined routing table from $\frac{k}{2}$ edge switches has $\frac{k}{2}$ in-bound entries and $\frac{k^2}{4}$ out-bound entries. This total number is within the TCAM capacity of commercial switches even for large-scale fat-tree networks. For instance, the table contains only 1056 entries for a $k = 64$ fat-tree with over 65k hosts.

5.4 Discussion

Layer-wise Partial Deployment: ShareBackup is especially powerful at the edge of the network. In today’s data centers, a host connects to one Top-of-Rack (ToR) switch only, and most fault-tolerant architectures fail to improve this condition [52–54]. If ToR or host link failures happen, hosts are disconnected and we have to rely on application frameworks to restart the task elsewhere. In this case, the job may be prolonged significantly or even crash due to loss of workers. In a fat-tree network, there are $\frac{k^2}{4}$ parallel paths in the core layer, but only $\frac{k}{2}$ in the aggregation layer. ShareBackup is more helpful in the aggregation layer, as rerouting may cause greater congestion with fewer paths to balance the load. Partial deployment is straightforward in ShareBackup thanks to the separate failure groups. We give a complete solution in this paper, but network operators have freedom to deploy backup switches in certain layers according to application requirements and monetary budget.

Cost Effectiveness: Redundancy incurs extra cost. We make key design decisions in ShareBackup to reduce cost. Concurrent failures are rare in data centers [120], so the ideal case is to have a single backup switch shared by the entire network. However, as discussed at the beginning of Section 5.2, this requires cascaded circuit

switches with high cost, insertion loss, and switching delay. As a compromise, we deploy low-cost circuit switches with short switching delay, e.g. electrical crosspoint switch or optical 2D-MEMS, in separate failure groups. A failure group needs at least 1 backup switch, so large groups help keep the backup pool small. Our targeted circuit switches have modest port count. As Figure 5.3 shows, we combine them to cover more switches and form larger failure groups. Our design achieves a reasonably low backup ratio at low circuit switch cost. The additional cabling cost is minimal in ShareBackup, because the circuit switches, either electrical or optical, are passive and do not require active elements on their end, e.g. optical transceiver or amplifier for copper. The layer-wise partial deployment can further reduces cost. A quantitative cost analysis with market price of devices is shown in Section 5.5.2.

Benefits to Network Management: When switches are routinely taken out for upgrade or maintenance, backup switches can neatly take their place to avoid downtime. Misconfigurations account for a large proportion of failures in data centers [120], and they are hard to reason and fix. ShareBackup can help mitigate the effect and diagnose the problem. The configurations of backup switches can be verified when they are idle. If a switch is misconfigured, it can failover to the backup switch whose configurations are guaranteed to be correct. Then complicated diagnosis can be executed offline. With judicious use of hardware, our offline failure diagnosis in Section 5.3.3 helps identify which interface has caused a link failure. In today’s data centers, failure diagnosis and repair are mostly handled manually and take hours at least. Even pioneering work like NetPilot takes 20 minutes only to mitigate failures [126]. As discussed in Section 5.5.3, with proper implementation of the control system, ShareBackup can repair failures in sub-ms. The failure diagnosis in Section 5.3.3 only involves simple communications between the network controllers,

switches, and circuit switches, so we expect the entire procedure to finish within seconds. This rapid automatic failure recovery and diagnosis is a breakthrough for data center management.

Extra Failures: Link failures are usually caused by faulty network interfaces. Although ShareBackup adds more cables, the number of network interfaces stays the same, so the network is unlikely to have significantly more link failures. Even if connectors to circuit switches fail, the failure recovery and diagnosis mechanisms will react to bypass the problematic connector and infer the cause of the error. Circuit switches are highly reliable passive physical-layer devices with bare-minimum control software. If the control software is unfunctional, the affected part of the network can still work under the current configuration. In the rare case that a circuit switch is completely down, the responsible controllers will receive a large number of link failure reports associated with the circuit switch in a short period of time. We can program controllers to stop failure recovery in such case and request for human intervention. As illustrated in Figure 5.3, each switch is connected to $\frac{k}{2}$ circuit switches. During the downtime of one circuit switch, each switch only loses $\frac{2}{k}$ capacity. Our distributed controllers are intrinsically robust, and it is simple to realize a fault-free control plane by redundancy.

Alternative Methods: An interesting question is whether PortLand and F10 will outperform ShareBackup if allowing the same deployment cost. Section 5.5.2 shows the cost of ShareBackup is only 6.7% more than fat-tree when $k = 48$. Tree networks are known to lack expandability. It is hard, if not impossible, to add only a small proportion of switches with the same port count. Even if more switches could be added, they would lock up bandwidth to fixed locations. Failures at highly unpredictable locations might still cause bandwidth loss. In contrast, ShareBackup

can move backup switches to wherever needed. Unstructured networks have been proposed for easier expansion [5, 127, 128], but the performance under failures is yet to be explored. Admittedly, these topologies have rich bandwidth and diverse paths, but the path length hugely varies, causing risk of path dilation. We can add switches to either provision bandwidth at the price of degraded performance under failures, or to provide guaranteed performance while keeping the backups idle most of the time. We choose the latter, and we believe shareable backup is an effective way to reduce the idle rate.

5.5 Architecture Properties

5.5.1 Capacity to Handle Failures

Switch failures: In a failure group, n backup switches are shared by $\frac{k}{2}$ switches. Thus, ShareBackup can handle n concurrent switch failures per failure group. In data centers, failures are independent; most devices have over 99.99% availability; and failures usually last for only a few minutes [120]. As a result, a small n is sufficient for a large-scale data center. For instance, let $n = 1$ for a $k = 48$ fat-tree with over 27k hosts, the backup ratio is $n/\frac{k}{2} = 4.17\%$, which is more than $400\times$ higher than the 0.01% switch failure rate.

Link failures: ShareBackup handles link failures as node failures. With failure diagnosis, we can identify the interface at fault, so we consume only one backup switch at the faulty end. For each failure group, ShareBackup can handle n independent link failures, which translates to up to kn link failures rooted at those n switches. Link failures are rare, and concurrent link failures are especially uncommon [120]. It is sufficient to target at a few link failures with a small n .

Circuit switch failures: Circuit switches are highly reliable. They are passive physical-layer devices with less than 10^{-12} bit-error rate [42, 72], and their bare-minimum control software for circuit reconfiguration receives infrequent requests only when switch and link failures happen. In the rare case that a circuit switch is down, switches connected to it will report link failures to their network controllers. If a controller receives a large number of link failure reports associated with one circuit switch in a short period of time, i.e. over a pre-defined threshold, it will stop failure recovery and request for human intervention. A rebooted circuit switch can get up-to-date circuit configurations from the controllers. Circuit switch port failures are sensed as link failures and handled by regular failure recovery.

Controller failures: ShareBackup uses distributed network controllers, so the failure of one controller only affects a small proportion of the network. If a failed switch cannot be replaced due to controller failure, other switches connected to it will take it as link failures. After a series of unsuccessful attempts to fix the problem, their controllers can notice the unusual condition and notify the network operator. A rebooted controller learns the current backup switches in its failure group from circuit switch configurations.

5.5.2 Cost Analysis

We calculate ShareBackup’s additional cost to fat-tree and compare to Aspen Tree [54] and 1:1 backup, which also add hardware to fat-tree to improve robustness. Table 5.2 lists the cost equations of these architectures and the market price of necessary devices. 1:1 backup requires twice as many switches as fat-tree, and the switch port count needs to be doubled. Assuming constant price of a switch port, the cost of 1:1 backup is $4\times$ that of fat-tree. Aspen Tree repurposes links between switches in adja-

Table 5.2 : Cost of compared architectures, where the data center uses electrical (E-DC) and optical (O-DC) transmissions respectively.

Architecture	Cost		
Fat-tree	$\frac{5}{4}k^3b + \frac{k^3}{2}c$		
ShareBackup	$\frac{3}{2}k^2(\frac{k}{2} + n + 2)a + \frac{5}{2}k^2nb + \frac{5}{4}k^2nc + \text{fat-tree cost}$		
Aspen Tree	$\frac{k^3}{2}b + \frac{k^3}{4}c + \text{fat-tree cost}$		
1:1 Backup	$\frac{15}{4}k^3b + \frac{3}{2}k^3c + \text{fat-tree cost}$		

Variable	Meaning	Price	Notes
a	Per-port cost of circuit switches	\$3 E-DC \$10 O-DC	Electrical crosspoint switch [72] 2D MEMS optical switch [42]
b	Per-port cost of packet switches	\$60	\$3000 for a 48-port 10Gbps bare metal switch
c	Cost per link	\$81 E-DC \$40 O-DC	10m 10Gbps DAC [129] 10Gbps transceiver (\$16) \times 2 + 10m 10Gbps optical fiber (\$8) [129]

cent layers. The lower-layer switches can disconnect half of the upper-layer switches to duplicate connections to the other half. One more layer of switches are needed to connect the partitioned network, so there are $\frac{k^2}{2}$ more switches and $\frac{k^3}{4}$ more cables.

In ShareBackup, the cost of controllers is negligible. Because of the simple functionality, controllers can be realized by bare-minimum computing hardware, and multiple controllers can reside on one physical machine to further reduce cost. The circuit switches chained up in a group (Figure 5.5) are placed close to each other, so the cost of the very short inter-circuit-switch optical fibers is minimal. ShareBackup has $\frac{5}{2}k$

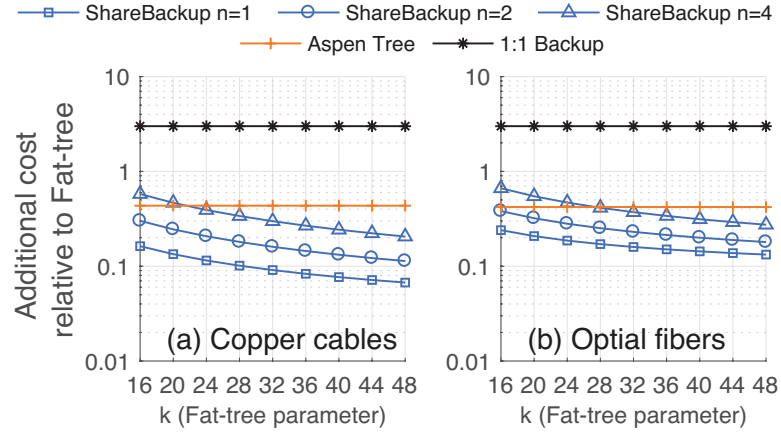


Figure 5.6 : Additional cost of ShareBackup, Aspen Tree, and 1:1 Backup relative to fat-tree at different network scales using market prices in Table 5.2

failure groups, each with n backup switches, and each Pod contains 3 sets of $\frac{k}{2}$ circuit switches with $(\frac{k}{2}+n+2)$ by $(\frac{k}{2}+n+2)$ ports. Thus, ShareBackup has $\frac{5}{2}kn$ more switches, $\frac{5}{4}k^2n$ more cables from these switches, and $\frac{3}{2}k^2(\frac{k}{2}+n+2)$ circuit switch ports. ShareBackup is less costly than Aspen Tree and 1:1 backup because it uses shareable backup and cheap circuit switches. As shown in Table 5.2, a is much cheaper than b and c , and n is a small constant in practice. Compared to Aspen Tree and 1:1 backup, ShareBackup reduces the power of k in b - and c -related terms, and the factor a limits the additional term to a relatively small value.

Figure 5.6 shows ShareBackup is multi-folds less expensive than 1:1 backup and Aspen Tree. For a fixed n , the relative additional cost of ShareBackup decreases as the network scales up, because the backup switches can be shared by more switches in the failure group. As discussed in Section 5.5.1, $n = 1$ is sufficient for a $k = 48$ fat-tree network. In this case, the additional cost of ShareBackup is merely 6.7% and 13.3% of the cost of fat-tree with copper cables and optical fibers respectively,

Table 5.3 : Performance characteristics of different network architectures

Architecture	No bandwidth loss?	No path dilation?	No upstream repair?
ShareBackup	✓	✓	✓
Fat-tree	×	✓	×
F10	×	×	✓
Aspen Tree	×	✓	✓/×

while Aspen Tree costs $6.5\times$ and $3.2\times$ as much. Even if n is increased to 4, which renders backup ratio as high as 16.7% for $k = 48$, ShareBackup is still cheaper than Aspen Tree. The cases where ShareBackup out-costs Aspen Tree show the flexibility of improving robustness by adding more backup switches.

5.5.3 Performance Characteristics

Scaling to large data centers with high robustness. The scalability of ShareBackup is determined by the port count of circuit switches, i.e. $(\frac{k}{2} + n + 2)$. 256-port electrical crosspoint switches are common place today [72], and 32-port 2D MEMS optical switches can be realized [42]. Even with the 32-port limit, we have $\frac{k}{2} + n + 2 = 32$, or $\frac{k}{2} + n = 30$. When $n = 1$, ShareBackup can support a $k = 58$ fat-tree network with over 48k hosts. The backup ratio is $n/\frac{k}{2} = 3.45\%$, which is significantly higher than the 0.01% switch failure rate. For a sizable $k = 48$ fat-tree with 27k hosts, n can reach 6, leading to a backup ratio as high as 25%. These parameters relating to scalability and robustness can be tuned to meet practical needs of the data center.

Recovering failures as fast as state of the art. F10 and Aspen Tree are state-of-the-art solutions for fast failure recovery [53, 54]. They reroute traffic locally

as soon as a switch detects a failure, so the recovery delay is the failure detector’s probing interval plus the time of redirecting packets to a different NIC interface. Rerouting requires change of at least one routing table entry. Using the standard SDN approach, it takes $\sim 1\text{ms}$ to modify a forwarding rule [112], e.g. changing the rule priority to match packets to a different interface. ShareBackup detects failures in a similar way, so we assume the same failure detection delay. Specifically, controllers probe switches for node failures, and switches probe each other for link failures and inform their controllers. With the failure information, a controller sends requests to circuit switches to reset circuits. The circuit reconfiguration delay is negligible, which is only 70ns for crosspoint switches [72] and $40\mu\text{s}$ for 2D MEMS [42]. Each failure group has a dedicated controller, so it can be placed close to the switches and the circuit switches. The communication channels are actively on because of probing. With efficient controller implementation, e.g. as a kernel module, the delay of switch-to-controller and controller-to-circuit-switch communications can be reduced to sub-ms level. Therefore, failure recovery in ShareBackup is as fast as that in F10 and Aspen Tree.

No bandwidth loss, no path dilation, and no upstream repair. Table 5.3 compares key features of failure-resilient architectures. Because ShareBackup replaces failed hardware completely, it does not have bandwidth loss. All other rerouting-based solutions have to cope with the remaining bandwidth resource, and we have demonstrated in Figure 5.1 that a single link or node failure can be disastrous to application performance. Fat-tree requires failure announcements to propagate multiple hops so rerouting can be performed upstream. To improve responsiveness, F10 reroutes traffic locally through longer paths, and Aspen Tree creates duplicate paths with extra hardware. As Figure 5.6 shows, the additional cost of Aspen Tree is high, so it pro-

vides the option of partial duplication that requires upstream repair. With shareable backup, our proposal replaces failed devices locally without path dilation at minimal additional cost to the network.

5.6 Summary

We introduce shareable backup as a novel solution to failure recovery in data center networks. It allows the entire network to share a small pool of backup devices. This proposal is grounded in three key observations. First, the traditional rerouting-based failure recovery is ineffective, because bandwidth loss from failures degrades application performance drastically. Therefore, failed devices should be replaced to restore bandwidth. Second, failures in data centers are rare but destructive [120], so it is desirable to seek cost-effective backup options. Third, the emergence of configurable data center network architectures promises feasibility of bringing backup devices online dynamically.

We find the effect of failures is magnified hugely on the application level: under failures, the number of impacted coflows is significantly greater than the number of impacted individual flows. As a result, we aim to restore bandwidth rapidly after failures. We design the ShareBackup prototype architecture based on fat-tree to realize the idea of replacing switches at runtime. We organize switches into failure groups and allow them to share one or more backup switches. Switches in the same failure group, as well as the backup switches, are connected to the same set of converter switches, so that they can be replaced by the backup switches when failed. Link failures are addressed as node failures on both ends, and we use offline failure diagnosis to understand the cause of problem and to recycle healthy switches. We use distributed network controllers to share the burden of failure detection and recov-

ery. For fast failure recovery, we support live impersonation of the failed switches on the control plane. Using market prices, the cost of ShareBackup is multi-fold lower than state-of-the-art failure-resilient architectures. It also provides more bandwidth compared to rerouting-based solutions.

The concept of shareable backup goes beyond our proposed failure recoverable fat-tree network. Like fat-tree, most data center network architectures have symmetric structures [11–13, 51]. Sharable backup is thus readily applicable to these networks, with different plans for partitioning failure groups. Non-uniform failure groups should also be explored, so that this idea can be extended to unstructured networks, such as Jellyfish [5], and we can have more backup on critical devices and less backup on unimportant ones. Several questions are worth further studies. First, a detailed failure diagnosis protocol between coordinating controllers need to be developed and evaluated. Second, the efficiency of the control plane depends on implementations of controllers, so the real-world performance is yet to be tested. Third, when backup switches are idle, they can be activated to add bandwidth to the network. How to make better use of backup switches to improve performance with guaranteed fault tolerance is an interesting research topic.

Chapter 6

Future Work

6.1 Joint Optimization of Network Topology and Traffic

In traditional data center networks, traffic is optimized against the network topology. The network has fixed topology, and traffic is engineered to take advantage of the available bandwidth resources. The optimization may happen at different layers, including routing [14, 15, 51, 52, 82], congestion-free data transportation [6, 17, 130–132], flow scheduling [16, 18–21, 83], and workload placement [26–29, 84]. The recent proposals of configurable network architectures have introduced the opposite way: optimizing the network topology against traffic [31–41, 46–49]. These works constantly monitor the traffic volume and change the network topology at fine granularity, e.g. on the scale of ms or μ s, to fit the traffic at runtime.

Other than these mainstream approaches, convertible network makes it possible to jointly optimize the network topology and traffic, giving potential for better transmission performance. The VM clustering algorithm in OmniSwitch makes decision for topology adaptation and traffic placement at the same. It heuristically changes the local topology and tentatively routes traffic to possible paths until the traffic is fully localized within the cabinet. Flat-tree suggests a suitable topology based on a high-level estimation of the communication properties and optimizes traffic using traditional methods on the chosen fixed topology. Recent studies also demonstrate the effectiveness of joint optimization. RotorNet can change the topology dynami-

cally. Instead of fully adapting the network topology to traffic demand, it routinely cycles through pre-defined topologies and shapes the traffic to a uniform pattern to fit the average topology. Kassing *et al.* uses the Xpander network, whose structure has been proved to provide rich bandwidth and easy expandability. Like RotorNet, this work also uniformizes the traffic, as uniform traffic is known to work best on the topology.

The joint optimization of topology and traffic has a very large design space, and many challenges arise. Most importantly, topology and traffic build on each other. On one hand, topology adjustment is to seek opportunities to optimize traffic. On the other hand, traffic optimization on the new topology changes characteristics of the traffic, which the topology update is based on in the first place. So, optimizing them iteratively may lead to a disruptive network that will never converge. Methods from optimization theory might be borrowed. However, these complicated procedures take much computation time, making them unsuitable for real-time decision making in a responsive data center network. Moreover, topology adjustment and traffic shaping are not without cost. Changing the topology frequently can match traffic accurately. Yet, since the network is disrupted during the transition period, too frequent a change will disrupt routing and transport protocols, as well as reduce the network utilization. Changing the traffic might increase the path length, thus wasting bandwidth in the network. In the huge design space, the above proposals only explored several design points that show decent performance in practice. Better designs require fundamental understanding of these inherent tradeoffs through formulation and theoretical analysis.

6.2 Combining Flat-tree and ShareBackup

Flat-tree and ShareBackup are different use cases of convertibility. Flat-tree changes the structure of the network dynamically to improve the transmission performance, while ShareBackup adds a small number of shareable backup switches to recover from failures. An interesting question is whether we can build a convertible network that achieves both goals, or combining flat-tree and ShareBackup.

This design is appealing for the cost advantage. ShareBackup requires additional hardware for redundancy. Yet, without adding more switches to the network, flat-tree can approximate random graph networks of different scales, which provide richer bandwidth than the original tree-structured network. If the extra bandwidth gained from topology conversion can be repurposed for failure recovery, we can reduce or even completely eliminate backup switches to save cost.

Despite the potential benefits, many open questions still remain. First, the placement of converter switches in flat-tree and ShareBackup are different. How should we place them in the new architecture to serve both purposes of bandwidth provisioning and fault tolerance? Second, ShareBackup requires distributed network controllers to constantly monitor the health of switches and links in the network, whereas a centralized controller is sufficient in flat-tree. How should we assign network controllers in the new architecture to distribute responsibility and minimize communication among them? Third, flat-tree has 3 operation modes, tree network, approximate local random graph, and approximate global random graph. How can we guarantee reliability in each individual topology? Last but not least, data center networks carry diverse applications, and people optimize their performance using different objectives according to their requirements, such as minimizing flow/job completion time and meeting deadlines. Failure recovery is yet another objective. How can we balance these equally

important goals and assign different priorities when necessary? These problems need to be addressed in future studies.

6.3 Convertibility for Power Saving

Besides improvement to transmission performance and fault tolerance, convertibility can play a broader role in network management, such as power saving. According to measurement studies [6–9], the average utilization of data center networks is low, and the network experiences busy/idle hours throughout the day. It would be desirable to automatically up/down-scale the network during busy/idle time. Specifically, some switches should be automatically shut down when the network utilization is low, and they should be turned on when needed. This is impossible traditionally, as the workload spread across the network requires all switches to be actively on, and compacting the workload to spare some switches involves very expensive VM migration. With the power of convertibility, the network can be partitioned to keep some switches functional, and traffic can be directed to these switches on the physical layer without shifting the workload around.

To enable this functionality, we should first monitor the power utilization of the switches and traffic demand in the network. Based on the collected information, a wiring plan can be formed to (1) minimize the number of switches being used and (2) have full coverage of all the traffic in the network. Unlike flat-tree and ShareBackup where topology change can be completed instantaneously, turning switches on and off takes non-negligible time. A straightforward solution is to convert the network topology at low frequency to improve the duty cycle and have a relatively stable network. A more sophisticated method is to leave a reasonable power head room with more active switches than necessary. The topology can evolve gradually given

the estimation of the power utilization in the near future, so that we can turn on/off switches ahead of time to combat the long delay. Machine learning based on statistical data of the past power utilization has potential to produce an accurate estimation. Which solution will be chosen in the end requires more understanding of operational features of specific data centers.

Chapter 7

Conclusion

This thesis introduces “convertible networks” as a new way to build data center networks and presents three prototype architectures as different use cases of this concept. A convertible network can switch between multiple topologies. This topology change is network-wide, and the network topology stays stable for a relatively long period of time throughout the life cycle of workloads. Using this approach, we achieve flexibility of the network structure and at the same time prevent frequent disruptions to traffic in the network.

Our exploration of convertible networks started from an industry project: OmniSwitch. The goal is to build a production-ready modular container with topological flexibility using the widely adopted Clos topology. In this project, we for the first time use small port-count converter switches to change the topology at low cost. Due to the limited port count of each individual converter switch, we deploy interleaving converter switches and Ethernet switches to provide convertibility and connectivity within the container. As universal building blocks of data centers, a number of OmniSwitch containers can be interconnected to form data center networks of different scales using different topologies. We give examples of mesh and tree networks, and numerous network structures are achievable with the rich permutation of converter switch configurations.

The convertibility facilitated by automatic rewiring of the converter switches can have many applications. For instance, we can direct traffic by converter switches

to desirable Ethernet switches within the container for traffic optimization, equip a spare Ethernet switch shareable to a set of active ones for efficient backup, allow partial population of devices for incremental expansion, and shut down some under-utilized switches to save power during idle hours. We demonstrate the potential of traffic optimization with the VM clustering case study. In the multi-tenant cloud environment, we want to localize traffic within the same VM cluster to make efficient use of the bandwidth. With convertibility, we can rewire the converter switches to bring VMs in the same cluster close to each other and thus reduce the hop count of transmission. We give a heuristic algorithm for this purpose and run simulations with a real data center workload for evaluations. Compared to state-of-the-art solutions, OmniSwitch reduces the average path length significantly and services more bandwidth using minimal computation time. Small converter switches are proven to provide similar convertibility to a high port-count counterpart.

OmniSwitch demonstrates on the conceptual level the feasibility of hardware implementation and potential applications of convertibility. However, it does not provide specific instructions of how to construct a convertible network. In the following two projects, we propose flat-tree and ShareBackup as two concrete architecture and system designs of convertible data center networks. They both base on the intuitions from OmniSwitch and target towards traffic optimization and efficient backup respectively.

OmniSwitch optimizes the network topology opportunistically within the container according to requirements of cloud tenants, yet a more fundamental questions is: what network topologies are desirable for data center traffic? We find Clos and random graph networks have complementary characteristics: Clos has low implementation complexity in practice and has good performance for inter-rack traffic; while

random graph is suitable for uniform network-wide traffic, and local random graphs can be constructed to optimize for more localized traffic. In the flat-tree project, we seek to convert the topology of the entire network between these options.

To convert between these completely different topologies, we flatten the tree structure of Clos network to approximate random graphs of different scales. Specifically, we relocate servers to different switches and diversify the connects between switches by reconfiguring the converter switches. Like OmniSwitch, flat-tree also places small converter switches in a distributed manner. To ease deployment, we package these converter switches into Pods and only connect adjacent Pods for neighbor-to-neighbor wiring. Using regular wiring patterns between Pods and core switches, we effectively approximate randomness in the network core and at the same time obtain low wiring complexity.

A control system is indispensable for flat-tree. In a random graph network, multi-path routing and congestion control are crucial to exploiting the path diversity for high-throughput transmission. However, multi-path routing generates an enormous number of states, exceeding the capacity of commercial switches. In flat-tree, we design state aggregation strategies including an architecture-specific addressing and source routing schemes. As a result, we perform multi-path routing on the ingress/egress switch level and transit switches in the network become stateless. Simulation results show that existing routing and transport protocols combined with our aggregation schemes can balance between high network utilization and fair bandwidth sharing among flows.

We explore the performance of flat-tree using simulations with real data center traffic and a testbed implementation of the system. Flat-tree has similar average path length as random graphs and the traffic throughput is indistinguishable. We

also observe flat-tree can optimize for diverse workloads with different topology options. This result validates the necessity of convertibility: we can choose the right network topology for each workload through conversion. In our testbed experiments, topology conversion takes tens of ms and the traffic can recover in 2.5s, meaning real-time topology change is feasible. We run Hadoop and Spark jobs on the testbed under different topology modes. The core bandwidth is increased by 27.6% by converting the topology from Clos to approximate random graph. With this improvement, the end-to-end reading time of the applications is reduced by around 10%, meaning the physical-layer topology change does bring benefits to the application-layer performance.

OmniSwitch supports 1: N efficient backup in the container, yet it is still unclear how to enable it in the entire network. For example, if two OmniSwitch containers are connected to each other, two layers of converter switches are connected back to back with duplicate functionalities; and we don't yet know how to handle link failures. We design the ShareBackup architecture to further explore how convertibility can be used to enhance fault tolerance in the network.

First of all, we need to understand the characteristics of failures in data center networks. We find from literature that failures are rare but disruptive in production data centers [44]. Our own experiments also validate the result. We observe that the effect of failures is magnified hugely on the application level: under failures, the number of impacted coflows is significantly greater than the number of impacted individual flows. Based on the evidence, a network ought to recover from failures immediately after they happen, and a small backup ratio is sufficient in practice. Therefore, we create the concept of “shareable backup”, which is to create a small pool of backup switches that can be shared by the entire network.

ShareBackup realizes this concept on the fat-tree network, which is the most widely adopted data center network architecture. We organize switches into failure groups and allow them to share one or more backup switches. Switches in the same failure group, as well as the backup switches, are connected to the same set of converter switches, so that they can be replaced by the backup switches when failed. This approach can recover node failures if backup switches are available. For link failures, we can replace switches on both ends of the link. To enable fast failure recovery, we take the switches offline instantly and then run failure diagnosis to understand the cause of problem and to recycle healthy switches.

OmniSwitch and flat-tree both use a single network controller to reconfigure the converter switches. In ShareBackup, monitoring the status of the switches constantly for failure detection is a heavy burden, so one controller is not enough. To address this problem, we assign distributed network controllers to each failure group. Because the ultimate goal is to recover failure fast enough to be transparent to applications, we cannot afford to change forwarding rules at runtime, which takes several milliseconds. Therefore, we support live impersonation of the failed switches on the control plane. We pre-set routing tables of all the edge switches in the failure group into each switch and differentiate them by VLANs. The end hosts set the VLAN ID according to the edge switch the host should connect to. Even if the edge switch has been replaced by the backup switch, by matching the VLAN ID, the switch knows which routing table to enable. In this way, we resolve conflicts of different routing tables and make backup switches as hot standbys. We derive formula for the cost of ShareBackup. Using market prices, the cost of ShareBackup is multi-fold lower than state-of-the-art failure-resilient architectures. We also analyze the qualitative properties of ShareBackup. Compared to rerouting-based solutions, It provides more bandwidth after failures,

does not use longer paths, and repairs the problem locally without failure propagation among switches.

Bibliography

- [1] M. Al-Fares, A. Loukissas, and A. Vahdat, “A Scalable, Commodity Data Center Network Architecture,” in *SIGCOMM '08*, (Seattle, Washington, USA), pp. 63–74, August 2008.
- [2] “Introducing data center fabric, the next-generation Facebook data center network, <https://code.facebook.com/posts/360346274145943/introducing-data-center-fabric-the-next-generation-facebook-data-center-network/>.”
- [3] A. Singh, J. Ong, A. Agarwal, G. Anderson, A. Armistead, R. Bannon, S. Bov-ing, G. Desai, B. Felderman, P. Germano, A. Kanagala, J. Provost, J. Simmons, E. Tanda, J. Wanderer, U. Hölzle, S. Stuart, and A. Vahdat, “Jupiter Rising: A Decade of Clos Topologies and Centralized Control in Google’s Datacenter Network,” in *SIGCOMM '15*, (London, United Kingdom), pp. 183–197, ACM, August 2015.
- [4] A. Singla, *Designing Data Center Networks for High Throughput*. Ph.D. Thesis, University of Illinois at Urbana-Champaign.
- [5] A. Singla, C.-Y. Hong, L. Popa, and P. B. Godfrey, “Jellyfish: Networking Data Centers Randomly,” in *NSDI '12*, (San Jose, California, USA), pp. 1–14, April 2012.
- [6] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan, “DCTCP: Efficient Packet Transport for the

- Commoditized Data Center,” in *SIGCOMM'10*, August 2010.
- [7] T. Benson, A. Anand, A. Akella, and M. Zhang, “Understanding Data Center Traffic Characteristics,” *SIGCOMM CCR*, vol. 40, no. 1, pp. 92–99, January 2010.
- [8] P. Bodík, I. Menache, M. Chowdhury, P. Mani, D. A. Maltz, and I. Stoica, “Surviving Failures in Bandwidth-constrained Datacenters,” in *SIGCOMM '12*, (Helsinki, Finland), pp. 431–442, August 2012.
- [9] S. Kandula, S. Sengupta, A. Greenberg, P. Patel, and R. Chaiken, “The Nature of Data Center Traffic,” in *IMC '09*, (Chicago, Illinois, USA), pp. 202–208, November 2009.
- [10] A. Roy, H. Zeng, J. Bagga, G. Porter, and A. C. Snoeren, “Inside the Social Network’s (Datacenter) Network,” in *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, SIGCOMM '15, (London, United Kingdom), pp. 123–137, August 2015.
- [11] C. Guo, H. Wu, K. Tan, L. Shi, Y. Zhang, and S. Lu, “DCell: A Scalable and Fault-Tolerant Network Structure for Data Centers,” in *SIGCOMM '08*, (Seattle, Washington, USA), pp. 75–86, August 2008.
- [12] C. Guo, G. Lu, D. Li, H. Wu, X. Zhang, Y. Shi, C. Tian, Y. Zhang, and S. Lu, “BCube: A High Performance, Server-centric Network Architecture for Modular Data Centers,” in *SIGCOMM '09*, (Barcelona, Spain), pp. 63–74, August 2009.
- [13] J. H. Ahn, N. Binkert, A. Davis, M. McLaren, and R. S. Schreiber, “HyperX: Topology, Routing, and Packaging of Efficient Large-scale Networks,” in *SC '09*, (Portland, Oregon, USA), pp. 41:1–41:11, November 2009.

- [14] C. Kim, M. Caesar, and J. Rexford, “Floodless in SEATTLE: A Scalable Ethernet Architecture for Large Enterprises,” in *SIGCOMM '08*, SIGCOMM '08, (Seattle, WA), pp. 3–14, 2008.
- [15] J. Mudigonda, P. Yalagandula, M. Al-Fares, and J. C. Mogul, “SPAIN: COTS Data-center Ethernet for Multipathing over Arbitrary Topologies,” in *NSDI'10*, (San Jose, CA), pp. 18–33, 2010.
- [16] M. Al-fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat, “Hedera: Dynamic Flow Scheduling for Data Center Networks,” in *NSDI '10*, (San Jose, CA), 2010.
- [17] D. Wischik, C. Raiciu, A. Greenhalgh, and M. Handley, “Design, implementation and evaluation of congestion control for multipath tcp,” in *Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation*, NSDI'11, (Berkeley, CA, USA), pp. 99–112, USENIX Association, 2011.
- [18] T. Benson, A. Anand, A. Akella, and M. Zhang, “MicroTE: Fine Grained Traffic Engineering for Data Centers,” in *CoNEXT '11*, (Tokyo, Japan), pp. 8:1–8:12, ACM, 2011.
- [19] M. Chowdhury, M. Zaharia, J. Ma, M. I. Jordan, and I. Stoica, “Managing Data Transfers in Computer Clusters with Orchestra,” in *SIGCOMM '11*, (Toronto, Ontario, Canada), pp. 98–109, 2011.
- [20] M. Chowdhury and I. Stoica, “Coflow: A Networking Abstraction for Cluster Applications,” in *HotNets-XI*, (Redmond, WA), pp. 31–36, 2012.
- [21] M. Chowdhury, Y. Zhong, and I. Stoica, “Efficient Coflow Scheduling with Varys,” in *SIGCOMM '14*, (Chicago, IL), pp. 443–454, 2014.

- [22] M. Alizadeh, S. Yang, M. Sharif, S. Katti, N. McKeown, B. Prabhakar, and S. Shenker, “pFabric: Minimal Near-optimal Datacenter Transport,” in *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*, SIGCOMM ’13, (Hong Kong, China), pp. 435–446, August 2013.
- [23] M. Chowdhury and I. Stoica, “Efficient Coflow Scheduling Without Prior Knowledge,” in *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, SIGCOMM ’15, (London, United Kingdom), pp. 393–406, August 2015.
- [24] H. Zhang, L. Chen, B. Yi, K. Chen, M. Chowdhury, and Y. Geng, “CODA: Toward Automatically Identifying and Scheduling Coflows in the Dark,” in *Proceedings of the 2016 Conference on ACM SIGCOMM 2016 Conference*, SIGCOMM ’16, (Florianopolis, Brazil), pp. 160–173, August 2016.
- [25] L. Chen, K. Chen, W. Bai, and M. Alizadeh, “Scheduling Mix-flows in Commodity Datacenters with Karuna,” in *Proceedings of the 2016 Conference on ACM SIGCOMM 2016 Conference*, SIGCOMM ’16, (Florianopolis, Brazil), pp. 174–187, August 2016.
- [26] C. Guo, G. Lu, H. J. Wang, S. Yang, C. Kong, P. Sun, W. Wu, and Y. Zhang, “SecondNet: A Data Center Network Virtualization Architecture with Bandwidth Guarantees,” in *CoNEXT ’10*, (Philadelphia, PA), pp. 15:1–15:12, 2010.
- [27] H. Ballani, P. Costa, T. Karagiannis, and A. Rowstron, “Towards Predictable Datacenter Networks,” in *SIGCOMM ’11*, (Toronto, Ontario, Canada), pp. 242–253, 2011.
- [28] L. Popa, A. Krishnamurthy, S. Ratnasamy, and I. Stoica, “FairCloud: Sharing

- the Network in Cloud Computing,” in *HotNets-X*, (Cambridge, MA), pp. 22:1–22:6, 2011.
- [29] J. Lee, Y. Turner, M. Lee, L. Popa, S. Banerjee, J.-M. Kang, and P. Sharma, “Application-driven Bandwidth Guarantees in Datacenters,” in *SIGCOMM ’14*, (Chicago, IL), pp. 467–478, 2014.
- [30] A. Verma, L. Pedrosa, M. Korupolu, D. Oppenheimer, E. Tune, and J. Wilkes, “Large-scale Cluster Management at Google with Borg,” in *Proceedings of the Tenth European Conference on Computer Systems, EuroSys ’15*, (Bordeaux, France), pp. 18:1–18:17, 2015.
- [31] G. Wang, D. G. Andersen, M. Kaminsky, K. Papagiannaki, T. S. E. Ng, M. Kozuch, and M. Ryan, “c-Through: Part-time Optics in Data Centers,” in *SIGCOMM ’10*, (New Delhi, India), pp. 327–338, August 2010.
- [32] N. Farrington, G. Porter, S. Radhakrishnan, H. H. Bazzaz, V. Subramanya, Y. Fainman, G. Papen, and A. Vahdat, “Helios: A Hybrid Electrical/Optical Switch Architecture for Modular Data Centers,” in *SIGCOMM ’10*, (New Delhi, India), pp. 339–350, August 2010.
- [33] M. Ghobadi, R. Mahajan, A. Phanishayee, N. Devanur, J. Kulkarni, G. Ranade, P.-A. Blanche, H. Rastegarfar, M. Glick, and D. Kilper, “ProjecToR: Agile Reconfigurable Data Center Interconnect,” in *Proceedings of the 2016 Conference on ACM SIGCOMM 2016 Conference*, SIGCOMM ’16, (Florianopolis, Brazil), pp. 216–229, August 2016.
- [34] D. Halperin, S. Kandula, J. Padhye, P. Bahl, and D. Wetherall, “Augmenting Data Center Networks with Multi-gigabit Wireless Links,” in *Proceedings*

- of the ACM SIGCOMM 2011 Conference*, SIGCOMM '11, (Toronto, Ontario, Canada), pp. 38–49, August 2011.
- [35] X. Zhou, Z. Zhang, Y. Zhu, Y. Li, S. Kumar, A. Vahdat, B. Y. Zhao, and H. Zheng, “Mirror Mirror on the Ceiling: Flexible Wireless Links for Data Centers,” in *Proceedings of the ACM SIGCOMM 2012 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, SIGCOMM '12, (Helsinki, Finland), pp. 443–454, August 2012.
- [36] N. Hamedazimi, Z. Qazi, H. Gupta, V. Sekar, S. R. Das, J. P. Longtin, H. Shah, and A. Tanwer, “FireFly: A Reconfigurable Wireless Data Center Fabric Using Free-space Optics,” in *Proceedings of the 2014 ACM Conference on SIGCOMM*, SIGCOMM '14, (Chicago, Illinois, USA), pp. 319–330, August 2014.
- [37] K. Chen, A. Singla, A. Singh, K. Ramachandran, L. Xu, Y. Zhang, X. Wen, and Y. Chen, “OSA: An Optical Switching Architecture for Data Center Networks with Unprecedented Flexibility,” in *NSDI '12*, (San Joes, CA), April 2012.
- [38] K. Chen, X. Wen, X. Ma, Y. Chen, Y. Xia, C. Hu, and Q. Dong, “WaveCube: A Scalable, Fault-tolerant, High-performance Optical Data Center Architecture,” in *2015 IEEE Conference on Computer Communications (INFOCOM)*, pp. 1903–1911, April 2015.
- [39] G. Porter, R. Strong, N. Farrington, A. Forencich, P. Chen-Sun, T. Rosing, Y. Fainman, G. Papen, and A. Vahdat, “Integrating Microsecond Circuit Switching into the Data Center,” in *SIGCOMM '13*, (Hong Kong, China), pp. 447–458, August 2013.
- [40] Y. J. Liu, P. X. Gao, B. Wong, and S. Keshav, “Quartz: A New Design Element

for Low-latency DCNs,” in *SIGCOMM '14*, (Chicago, Illinois, USA), pp. 283–294, August 2014.

- [41] “Plexxi, <http://www.plexxi.com/>.”
- [42] M. C. Wu, O. Solgaard, and J. E. Ford, “Optical MEMS for Lightwave Communication,” *Journal of Lightwave Technology*, vol. 24, pp. 4433–4454, December 2006.
- [43] M. Fokine, L. E. Nilsson, Å. Claesson, D. Berlemont, L. Kjellberg, L. Krumenacher, and W. Margulis, “Integrated Fiber Mach–Zehnder Interferometer for Electro-Optic Switching,” *Optics Letters*, vol. 27, pp. 1643–1645, September 2002.
- [44] P. Gill, N. Jain, and N. Nagappan, “Understanding Network Failures in Data Centers: Measurement, Analysis, and Implications,” in *Proceedings of the ACM SIGCOMM 2011 Conference*, SIGCOMM '11, (Toronto, Ontario, Canada), pp. 350–361, ACM, Aug. 2011.
- [45] H. Abu-Libdeh, P. Costa, A. Rowstron, G. O’Shea, and A. Donnelly, “Symbiotic Routing in Future Data Centers,” in *SIGCOMM '10*, (New Delhi, India), pp. 51–62, August 2010.
- [46] H. Wang, Y. Xia, K. Bergman, T. S. E. Ng, S. Sahu, and K. Sripanidkulchai, “Rethinking the Physical Layer of Data Center Networks of the Next Decade: Using Optics to Enable Efficient *-cast Connectivity,” *SIGCOMM Comput. Commun. Rev.*, vol. 43, pp. 52–58, July 2013.
- [47] Y. Xia, T. S. E. Ng, and X. Sun, “Blast: Accelerating High-Performance Data

- Analytics Applications by Optical Multicast,” in *INFOCOM '15*, (Hong Kong, China), pp. 1930–1938, April 2015.
- [48] N. Farrington, G. Porter, S. Radhakrishnan, H. H. Bazzaz, V. Subramanya, Y. Fainman, G. Papen, and A. Vahdat, “Helios: A Hybrid Electrical/Optical Switch Architecture for Modular Data Centers,” in *SIGCOMM '10*, (New Delhi, India), p. 339, Aug. 2010.
- [49] G. Wang, D. G. Andersen, M. Kaminsky, K. Papagiannaki, T. S. E. Ng, M. Kozuch, and M. Ryan, “c-Through: Part-time Optics in Data Centers,” in *SIGCOMM '10*, (New Delhi, India), p. 327, Aug. 2010.
- [50] K. Chen, A. Singla, A. Singh, K. Ramachandran, L. Xu, Y. Zhang, X. Wen, and Y. Chen, “OSA: An Optical Switching Architecture for Data Center Networks with Unprecedented Flexibility,” in *NSDI '12*, (San Joes, CA, USA), April 2012.
- [51] A. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. A. Maltz, P. Patel, and S. Sengupta, “VL2: A Scalable and Flexible Data Center Network,” in *Proceedings of the ACM SIGCOMM 2009 Conference on Data Communication*, SIGCOMM '09, (New York, NY, USA), pp. 51–62, ACM, 2009.
- [52] R. Niranjan Mysore, A. Pamboris, N. Farrington, N. Huang, P. Miri, S. Radhakrishnan, V. Subramanya, and A. Vahdat, “PortLand: A Scalable Fault-tolerant Layer 2 Data Center Network Fabric,” in *Proceedings of the ACM SIGCOMM 2009 Conference on Data Communication*, SIGCOMM '09, (New York, NY, USA), pp. 39–50, ACM, 2009.

- [53] V. Liu, D. Halperin, A. Krishnamurthy, and T. Anderson, “F10: A Fault-Tolerant Engineered Network,” in *Presented as part of the 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13)*, (Lombard, IL), pp. 399–412, USENIX, 2013.
- [54] M. Walraed-Sullivan, A. Vahdat, and K. Marzullo, “Aspen Trees: Balancing Data Center Fault Tolerance, Scalability and Cost,” in *Proceedings of the Ninth ACM Conference on Emerging Networking Experiments and Technologies*, CoNEXT ’13, (New York, NY, USA), pp. 85–96, ACM, 2013.
- [55] S. Kini, S. Ramasubramanian, A. Kvalbein, and A. F. Hansen, “Fast recovery from dual link failures in IP networks,” in *INFOCOM 2009, IEEE*, pp. 1368–1376, IEEE, 2009.
- [56] A. Kvalbein, A. F. Hansen, S. Gjessing, and O. Lysne, “Fast IP network recovery using multiple routing configurations,” in *in INFOCOM 2006. 25th IEEE International Conference on Computer Communications. Proceedings*, Citeseer, 2006.
- [57] S. Lee, Y. Yu, S. Nelakuditi, Z.-L. Zhang, and C.-N. Chuah, “Proactive vs reactive approaches to failure resilient routing,” in *INFOCOM 2004. Twenty-third Annual Joint Conference of the IEEE Computer and Communications Societies*, vol. 1, IEEE, 2004.
- [58] A. Li, X. Yang, and D. Wetherall, “Safeguard: safe forwarding during route changes,” in *Proceedings of the 5th international conference on Emerging networking experiments and technologies*, pp. 301–312, ACM, 2009.
- [59] M. Motiwala, M. Elmore, N. Feamster, and S. Vempala, “Path splicing,” in

- ACM SIGCOMM Computer Communication Review*, vol. 38, pp. 27–38, ACM, 2008.
- [60] W. Xu and J. Rexford, *MIRO: multi-path interdomain routing*, vol. 36. ACM, 2006.
- [61] X. Yang and D. Wetherall, “Source selectable path diversity via routing deflections,” in *ACM SIGCOMM Computer Communication Review*, vol. 36, pp. 159–170, ACM, 2006.
- [62] K. Lakshminarayanan, M. Caesar, M. Rangan, T. Anderson, S. Shenker, and I. Stoica, “Achieving convergence-free routing using failure-carrying packets,” *ACM SIGCOMM Computer Communication Review*, vol. 37, no. 4, pp. 241–252, 2007.
- [63] S. S. Lor, R. Landa, and M. Rio, “Packet re-cycling: eliminating packet losses due to network failures,” in *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, p. 2, ACM, 2010.
- [64] N. Kushman, S. Kandula, D. Katabi, and B. M. Maggs, “R-bgp: Staying connected in a connected world,” USENIX, 2007.
- [65] J. Liu, A. Panda, A. Singla, B. Godfrey, M. Schapira, and S. Shenker, “Ensuring connectivity via data plane mechanisms.,” in *NSDI*, pp. 113–126, 2013.
- [66] B. Yang, J. Liu, S. Shenker, J. Li, and K. Zheng, “Keep forwarding: Towards k-link failure resilient routing,” in *INFOCOM, 2014 Proceedings IEEE*, pp. 1617–1625, IEEE, 2014.

- [67] B. Stephens and A. L. Cox, “Deadlock-free local fast failover for arbitrary data center networks,” in *Computer Communications, IEEE INFOCOM 2016-The 35th Annual IEEE International Conference on*, pp. 1–9, IEEE, 2016.
- [68] B. Stephens, A. L. Cox, and S. Rixner, “Scalable multi-failure fast failover via forwarding table compression,” in *Proceedings of the Symposium on SDN Research*, p. 9, ACM, 2016.
- [69] L. Schiff, M. Borokhovich, and S. Schmid, “Reclaiming the brain: Useful openflow functions in the data plane,” in *Proceedings of the 13th ACM Workshop on Hot Topics in Networks*, p. 7, ACM, 2014.
- [70] M. Borokhovich, L. Schiff, and S. Schmid, “Provable data plane connectivity with local fast failover: Introducing openflow graph algorithms,” in *Proceedings of the third workshop on Hot topics in software defined networking*, pp. 121–126, ACM, 2014.
- [71] M. Reitblatt, M. Canini, A. Guha, and N. Foster, “FatTire: Declarative Fault Tolerance for Software-defined Networks,” in *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, HotSDN ’13, (Hong Kong, China), pp. 109–114, ACM, Aug. 2013.
- [72] S. Legtchenko, N. Chen, D. Cletheroe, A. Rowstron, H. Williams, and X. Zhao, “XFabric: A Reconfigurable In-Rack Network for Rack-Scale Computers,” in *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)*, (Santa Clara, CA), pp. 15–29, USENIX Association, 2016.
- [73] “40G short range transceiver.”

- [74] J. R. Hamilton, “An Architecture for Modular Data Centers,” in *CIDR’07*, (Asilomar, California, USA), pp. 306–313, 2007.
- [75] “B. Canney, IBM Portable Modular Data Center Overview, <https://www.ibm.com/us-en/marketplace/prefabricated-modular-data-center>.”
- [76] “HP Performance Optimized Datacenter, <https://www.hpe.com/us/en/integrated-systems/pods.html>.”
- [77] “SGI ICE Cube, <http://www.sgi.com/pdfs/4160.pdf>.”
- [78] M. Kozhevnikov, N. Basavanhally, J. Weld, Y. Low, P. Kolodner, C. Bolle, R. Ryf, A. Papazian, A. Olkhovets, F. Pardo, *et al.*, “Compact 64 x 64 micromechanical optical cross connect,” *IEEE Photonics Technology Letters*, vol. 15, no. 7, pp. 993–995, 2003.
- [79] M. Yano, F. Yamagishi, and T. Tsuda, “Optical mems for photonic switching—compact and stable optical crossconnect switches for simple, fast, and flexible wavelength applications in recent photonic networks,” *IEEE Journal of Selected Topics in Quantum Electronics*, vol. 11, no. 2, pp. 383–394, 2005.
- [80] S. Kandula, S. Sengupta, A. Greenberg, P. Patel, and R. Chaiken, “The Nature of Data Center Traffic: Measurements & Analysis,” in *IMC ’09*, (Chicago, IL), pp. 202–208, 2009.
- [81] T. Benson, A. Anand, A. Akella, and M. Zhang, “Understanding Data Center Traffic Characteristics,” *SIGCOMM CCR*, vol. 40, pp. 92–99, Jan. 2010.

- [82] B. Stephens, A. Cox, W. Felter, C. Dixon, and J. Carter, “PAST: Scalable Ethernet for Data Centers,” in *CoNEXT '12*, (Nice, France), pp. 49–60, 2012.
- [83] M. Chowdhury, S. Kandula, and I. Stoica, “Leveraging Endpoint Flexibility in Data-Intensive Clusters,” in *SIGCOMM '13*, (Hong Kong, China), pp. 231–242, 2013.
- [84] X. Meng, V. Pappas, and L. Zhang, “Improving the Scalability of Data Center Networks with Traffic-aware Virtual Machine Placement,” in *INFOCOM'10*, (San Diego, CA), pp. 1154–1162, 2010.
- [85] W. Voorsluys, J. Broberg, S. Venugopal, and R. Buyya, “Cost of Virtual Machine Live Migration in Clouds: A Performance Evaluation,” in *CloudCom '09*, (Beijing, China), pp. 254–265, 2009.
- [86] H. Liu, C. F. Lam, and C. Johnson, “Scaling Optical Interconnects in Datacenter Networks Opportunities and Challenges for WDM,” in *HOTI '10*, (Mountain View, CA), pp. 113–116, 2010.
- [87] M. Schlansker, Y. Turner, J. Tourrilhes, and A. Karp, “Ensemble Routing for Datacenter Networks,” in *ANCS '10*, (La Jolla, CA), pp. 23:1–23:12, 2010.
- [88] P. Bodík, I. Menache, M. Chowdhury, P. Mani, D. A. Maltz, and I. Stoica, “Surviving Failures in Bandwidth-Constrained Datacenters,” in *SIGCOMM '12*, (Helsinki, Finland), pp. 431–442, 2012.
- [89] N. G. Duffield, P. Goyal, A. Greenberg, P. Mishra, K. K. Ramakrishnan, and J. E. van der Merive, “A Flexible Model for Resource Management in Virtual Private Networks,” in *SIGCOMM '99*, (Cambridge, MA), pp. 95–108, 1999.

- [90] K. LaCurts, J. C. Mogul, H. Balakrishnan, and Y. Turner, “Cicada: Introducing Predictive Guarantees for Cloud Networks,” in *HotCloud’14*, (Philadelphia, PA), pp. 14–19, 2014.
- [91] S. Even, A. Itai, and A. Shamir, “On the Complexity of Time Table and Multi-commodity Flow Problems,” in *SFCS ’75*, (Washington, DC), pp. 184–193, 1975.
- [92] J. Y. Yen, “Finding the K Shortest Loopless Paths in a Network,” *Management Science*, vol. 17, no. 11, pp. 712–716, 1971.
- [93] Y. Xia, M. Schlansker, T. S. E. Ng, and J. Tourrilhes, “Enabling Topological Flexibility for Data Centers Using OmniSwitch,” in *HotCloud ’15*, (Santa Clara, CA), July 2015.
- [94] Y. Xia and T. S. E. Ng, “Flat-tree: A Convertible Data Center Network Architecture from Clos to Random Graph,” in *Proceedings of the 15th ACM Workshop on Hot Topics in Networks*, HotNets ’16, (Atlanta, GA), pp. 71–77, November 2016.
- [95] C. Hopps, “Analysis of an Equal-Cost Multi-Path Algorithm,” *RFC 2992*, 2000.
- [96] A. Ford, C. Raiciu, M. Handley, S. Barre, and J. Iyengar, “Architectural Guidelines for Multipath TCP Development,” *RFC 6182*, 2011.
- [97] A. Ford, C. Raiciu, M. Handley, and O. Bonaventure, “TCP Extensions for Multipath Operation with Multiple Addresses,” *RFC 6824*, 2013.
- [98] M. Scharf and A. Ford, “Multipath TCP (MPTCP) Application Interface Considerations,” *RFC 6897*, 2013.

- [99] “MultiPath TCP - Linux Kernel implementation, <http://multipath-tcp.org/pmwiki.php/Main/HomePage>.”
- [100] R. M. Ramos, M. Martinello, and C. E. Rothenberg, “SlickFlow: Resilient source routing in Data Center Networks unlocked by OpenFlow,” in *38th Annual IEEE Conference on Local Computer Networks*, pp. 606–613, October 2013.
- [101] S. A. Jyothi, M. Dong, and P. B. Godfrey, “Towards a Flexible Data Center Fabric with Source Routing,” in *Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research, SOSR '15*, (Santa Clara, California), pp. 10:1–10:8, June 2015.
- [102] M. Soliman, *Exploring Source Routing as an Alternative Routing Approach in Wide Area Software-Defined Networks*. PhD thesis, Carleton University Ottawa, 2015.
- [103] C. Filsfils, S. Previdi, B. Decraene, S. Litkowski, and R. Shakir, “Segment Routing Architecture,” *IETF Draft: draft-ietf-spring-segment-routing-04*, 2016.
- [104] “Segment Routing: Prepare Your Network for New Business Models White Paper,” *Cisco Technology White Paper*, 2015.
- [105] “Segment Routing and Path Computation Element,” *Nokia Technology White Paper*.
- [106] E. Rosen, A. Viswanathan, and R. Callon, “Multiprotocol Label Switching Architecture,” *RFC 3031*, 2001.

- [107] M. Soliman, B. Nandy, I. Lambadaris, and P. Ashwood-Smith, “Source Routed Forwarding with Software Defined Control, Considerations and Implications,” in *Proceedings of the 2012 ACM Conference on CoNEXT Student Workshop*, CoNEXT Student ’12, (Nice, France), pp. 43–44, December 2012.
- [108] Y. Chiba, Y. Shinohara, and H. Shimonishi, “Source Flow: Handling Millions of Flows on Flow-based Nodes,” in *Proceedings of the ACM SIGCOMM 2010 Conference*, SIGCOMM ’10, (New Delhi, India), pp. 465–466, August 2010.
- [109] “OpenFlow Switch Specification, Version 1.3.0,” *Open Networking Foundation*, 2012.
- [110] “Glimmerglass 80x80 MEMS switch, <http://electronicdesign.com/article/test-and-measurement/3d-mems-based-optical-switch-handles-80-by-80-fibe.aspx>.”
- [111] L. J. Hornbeck, “Digital Light Processing for high-brightness high-resolution applications,” 1997.
- [112] K. He, J. Khalid, A. Gember-Jacobson, S. Das, C. Prakash, A. Akella, L. E. Li, and M. Thottan, “Measuring Control Plane Latency in SDN-enabled Switches,” in *Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research*, SOSR ’15, (Santa Clara, California), pp. 25:1–25:6, 2015.
- [113] C. Rotsos, N. Sarrar, S. Uhlig, R. Sherwood, and A. W. Moore, “OFLOPS: An Open Framework for Openflow Switch Evaluation,” in *Proceedings of the 13th International Conference on Passive and Active Measurement*, PAM’12, (Berlin, Heidelberg), pp. 85–95, Springer-Verlag, 2012.
- [114] T. Leighton and S. Rao, “Multicommodity Max-flow Min-cut Theorems and

- Their Use in Designing Approximation Algorithms,” *J. ACM*, vol. 46, no. 6, pp. 787–832, November 1999.
- [115] “MPTCP simulator, <http://nrg.cs.ucl.ac.uk/mptcp/implementation.html>.”
- [116] W. Dally and B. Towles, *Principles and Practices of Interconnection Networks*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2003.
- [117] “Coflow-Benchmark, <https://github.com/coflow/coflow-benchmark>.”
- [118] “Facebook Network Analytics Data Sharing, <https://www.facebook.com/groups/1144031739005495/>.”
- [119] “Tez, <https://tez.apache.org/>.”
- [120] P. Gill, N. Jain, and N. Nagappan, “Understanding Network Failures in Data Centers: Measurement, Analysis, and Implications,” in *Proceedings of the ACM SIGCOMM 2011 Conference*, SIGCOMM ’11, (New York, NY, USA), pp. 350–361, ACM, 2011.
- [121] Y. Xia, X. S. Sun, S. Dzinamarira, D. Wu, X. S. Huang, and T. S. E. Ng, “A Tale of Two Topologies: Exploring Convertible Data Center Network Architectures with Flat-tree,” in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM ’17, (New York, NY, USA), pp. 295–308, ACM, 2017.
- [122] M. Schlansker, M. Tan, J. Tourrilhes, J. R. Santos, and S.-Y. Wang, “Configurable optical interconnects for scalable datacenters,” in *Optical Fiber Communication Conference and Exposition and the National Fiber Optic Engineers Conference (OFC/NFOEC), 2013*, pp. 1–3, IEEE, 2013.

- [123] D. Zhuo, M. Ghobadi, R. Mahajan, K.-T. Förster, A. Krishnamurthy, and T. Anderson, “Understanding and Mitigating Packet Corruption in Data Center Networks,” in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM ’17, (Los Angeles, CA), pp. 362–375, ACM, 2017.
- [124] “Arduino, <https://www.arduino.cc>.”
- [125] “Raspberry Pi, <https://www.raspberrypi.org>.”
- [126] X. Wu, D. Turner, C.-C. Chen, D. A. Maltz, X. Yang, L. Yuan, and M. Zhang, “NetPilot: Automating Datacenter Network Failure Mitigation,” in *Proceedings of the ACM SIGCOMM 2012 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, SIGCOMM ’12, (Helsinki, Finland), pp. 419–430, August 2012.
- [127] S. Kassing, A. Valadarsky, G. Shahaf, M. Schapira, and A. Singla, “Beyond Fat-trees Without Antennae, Mirrors, and Disco-balls,” in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM ’17, (Los Angeles, CA, USA), pp. 281–294, ACM, 2017.
- [128] A. Valadarsky, G. Shahaf, M. Dinitz, and M. Schapira, “Xpander: Towards Optimal-Performance Datacenters,” in *Proceedings of the 12th International on Conference on Emerging Networking EXperiments and Technologies*, CoNEXT ’16, (Irvine, California, USA), pp. 205–219, ACM, 2016.
- [129] “FS.COM, <http://www.fs.com/>.”
- [130] B. Vamanan, J. Hasan, and T. Vijaykumar, “Deadline-aware Datacenter TCP (D2TCP),” in *Proceedings of the ACM SIGCOMM 2012 Conference on Applica-*

tions, Technologies, Architectures, and Protocols for Computer Communication, SIGCOMM '12, (Helsinki, Finland), pp. 115–126, ACM, 2012.

- [131] C. Wilson, H. Ballani, T. Karagiannis, and A. Rowtron, “Better Never Than Late: Meeting Deadlines in Datacenter Networks,” in *Proceedings of the ACM SIGCOMM 2011 Conference*, SIGCOMM '11, (Toronto, Ontario, Canada), pp. 50–61, ACM, 2011.

- [132] M. Handley, C. Raiciu, A. Agache, A. Voinescu, A. W. Moore, G. Antichi, and M. Wójcik, “Re-architecting Datacenter Networks and Stacks for Low Latency and High Performance,” in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM '17, (Los Angeles, CA, USA), pp. 29–42, ACM, 2017.