Chapter 6
Data Center Architecture, Operation, and Optimization

Kaiyang Liu, Aqun Zhao, and Jianping Pan

6.1 Introduction

The explosive growth of workloads driven by data-intensive applications, e.g., web search, social networks, and e-commerce, has led mankind into the era of big data [1]. According to the IDC report, the volume of data is doubling every 2 years and thus will reach a staggering 175 ZB by 2025 [2]. Data centers have emerged as an irreplaceable and crucial infrastructure to power this ever-growing trend.

As the foundation of cloud computing, data centers can provide powerful parallel computing and distributed storage capabilities to manage, manipulate, and analyze massive amounts of data. A special network, i.e., data center network (DCN), is designed to interconnect a large number of computing and storage nodes. In comparison with traditional networks, e.g., local area networks and wide area networks, the design of DCN has its unique challenges and requirements [3], which are summarized as follows:

Hyperscale: Currently, over 500 hyperscale data centers are distributed across the globe. We are witnessing the exponential growth of scale in modern data centers. For example, Range International Information Group located in Langfang, China, which is one of the largest data centers in the world, occupies 6.3 million square feet of space. A hyperscale data center hosts over a million servers spreading across

K. Liu (✉)
Department of Computer Science, University of Victoria, Victoria, BC, Canada

School of Computer Science and Engineering, Central South University, Changsha, China
e-mail: liukaiyang@uvic.ca

A. Zhao
School of Computer and Information Technology, Beijing Jiaotong University, Beijing, China

J. Pan
Department of Computer Science, University of Victoria, Victoria, BC, Canada

M. Toy (ed.), *Future Networks, Services and Management*,
https://doi.org/10.1007/978-3-030-81961-3_6

hundreds of thousands of racks [4]. Data centers at such a large scale put forward severe challenges on system design in terms of interconnectivity, flexibility, robustness, efficiency, and overheads.

Huge Energy Consumption: In 2018, global data centers consumed about 205 tWh of electricity, or 1% of global electricity consumed in that year [5]. It has been predicted that the electricity usage of data centers will increase about 15-fold by 2030 [6]. The huge energy consumption prompts data centers to improve the energy efficiency of the hardware and system cooling. However, according to the New York Times report [7], most data centers consume vast amounts of energy in an incongruously wasteful manner. Typically, service providers operate their facilities at maximum capacity to handle the possible bursty service requests. As a result, data centers can waste 90% or more of the total consumed electricity.

Complex Traffic Characteristics: Modern data centers have been applied to a wide variety of scenarios, e.g., Email, video content distribution, and social networking. Furthermore, data centers are also employed to run large-scale data-intensive tasks, e.g., indexing Web pages and big data analytics [8]. Driven by diversified services and applications, data center traffic shows complex characteristics, i.e., high fluctuation with the long-tail distribution. In fact, most of the flows are short flows, but most of the bytes are from long flows [9]. Short flows are processed before optimization decision takes place. Furthermore, data centers suffer from fragmentation with intensive short flows. It is a challenge to handle traffic optimization tasks in hyperscale data centers.

Tight Service-Level Agreement: The service-level agreement (SLA) plays the most crucial part in a data center lease, spelling out the performance requirements of services that data centers promise to provide in exact terms. It has been increasingly common to include mission-critical data center services in SLAs such as power availability, interconnectivity, security, response time, and delivery service levels. Considering the inevitable network failures, congestion, or even human errors, constant monitoring, agile failure recovery, and congestion control schemes are necessary to provide tight SLAs.

To solve these significant technical challenges above, DCNs have been widely investigated in terms of network topology [3], routing [10], load balancing [11], green networking [12], optical networking [13], and network virtualization [14]. This book chapter presents a systematic view of DCNs from both the architectural and operational principle aspects. We start with a discussion on the state-of-the-art DCN topologies (Sect. 6.2). Then, we examine various operation and optimization solutions in DCNs (Sect. 6.3). Thereafter, we discuss the outlook of future DCNs and their applications (Sect. 6.4). The main goal of this book chapter is to highlight the salient features of existing solutions which can be utilized as guidelines in constructing future DCN architectures and operational principles.

6.2 Data Center Network Topologies

Currently, the research on the DCN topologies can be divided into two types: switch centric schemes and server centric schemes. Some of the topologies originate from the interconnection networks in supercomputing for both categories. Furthermore, some researchers have introduced optical switching technology into the DCN and proposed some full optical and optical/electronic hybrid topologies. Others have introduced wireless technology into the DCN and proposed some wireless DCN topologies. Besides, in the real world, most service providers have built their production data centers with some specific topologies.

6.2.1 Switch-Centric Data Center Network Topologies

In the switch-centric DCN topologies, the network traffic is all routed and forwarded by the switches or routers. These topologies include Fat-tree [15], VL2 [16], Diamond [17], Aspen Trees [18], F10 [19], F^2 Tree [20], Scafida [21], Small-World [22], and Jellyfish [23]. In this section, some representative schemes are selected for introduction.

Fat-tree: Fat-tree is proposed by Al-Fares et al., which drew on the experience of Charles Clos et al. in the field of telephone networks 50 years ago [15]. A general Fat-tree model is a k-port n-tree topology [24]. In the data center literature, a special instance of it with $n = 3$ is usually adopted. In this Fat-tree topology, each k-port switch in the edge level is connected to $\frac{k}{2}$ servers. The remaining $\frac{k}{2}$ ports are connected to $\frac{k}{2}$ switches at the aggregation level. The $\frac{k}{2}$ aggregation level switches, $\frac{k}{2}$ edge level switches, and the connected servers form a basic cell of a Fat-tree, called a pod. At the core level, there are $\left(\frac{k}{2}\right)^2$ k-port switches, each connecting to each of the k pods. Figure 6.1 shows the Fat-tree topology with 4-port switches. The maximum number of servers in a Fat-tree with k-port switches is $\frac{k^3}{4}$.

Fat-tree has many advantages. Firstly, it eliminates the throughput limitation of the upper links of the tree structure and provides multiple parallel paths for communication between servers. Secondly, its horizontal expansion reduces the cost of building a DCN. Finally, this topology is compatible with Ethernet structure and IP-configured servers used in existing networks. However, the scalability of Fat-tree is limited by the number of switch ports. Another drawback is that it is not fault-tolerant enough and is very sensitive to edge switch failures. Finally, the number of switches needed to build Fat-tree is large, which increases the complexity of wiring and configuration.

VL2: VL2 is proposed by Greenberg et al. using a Clos Network topology in [16]. VL2 is also a multi-rooted tree. When deployed in DCNs, VL2 usually consists of three levels of switches: the Top of Rack (ToR) switches directly connected to servers, the aggregation switches connected to the ToR switches, and the intermediate switches connected to the aggregation switches. The number of the switches is determined by the number of ports on the intermediate switches and aggregation switches. If each of these switches has k ports, there will be k aggregation switches and $\frac{k}{2}$ intermediate switches. There is exactly one link between each intermediate switch and each aggregation switch. The remaining $\frac{k}{2}$ ports on each aggregation switch are connected to $\frac{k}{2}$ different ToR switches. Each of the ToR switches is connected to two different aggregation switches, and the remaining ports on the ToR switches are connected to servers. There are $\frac{k^2}{4}$ ToR switches because $\frac{k}{2}$ ToR switches are connected to each pair of aggregation switches. While intermediate switches and aggregation switches must have the same number of ports, the number of ports on a ToR switch is not limited. If k_{ToR} ports on each ToR switch are connected to servers, there will be $\frac{k^2}{4} \cdot k_{\text{ToR}}$ servers in the network. Figure 6.2 shows the VL2 topology with $k = 4$, $k_{\text{ToR}} = 2$.

The difference of VL2 with Fat-tree is that the topology between intermediate switches and aggregation switches forms a complete bipartite graph, and each ToR switch is connected to two aggregation switches. VL2 reduces the number of cables by leveraging higher speed switch-to-switch links, e.g., 10 Gbps for switch-to-switch links and 1 Gbps for server-to-switch links.

6.2.2 Server-Centric Data Center Network Topologies

In the server-centric DCN designs, the network topologies are constructed by recursion. The servers are not only computing devices but also routing nodes and will actively participate in packet forwarding and load balancing. These topologies avoid the bottleneck in the core switches through recursive design, and there are multiple disjoint paths between servers. Typical topologies include DCell [25], BCube [26],

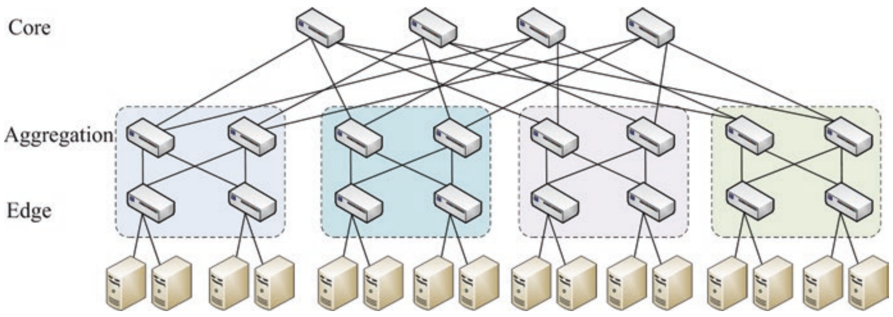

Fig. 6.1 The Fat-tree topology with $k = 4$ ports

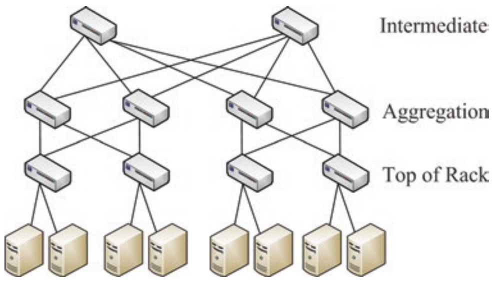

Fig. 6.2 The VL2 topology with $k = 4$ ports and $k_{\mathrm{ToR}} = 2$

FiConn [27], DPillar [28], MCube [29], MDCube [30], PCube [31], Snowflake [32], HFN [33], and so on.

DCell: DCell topology was proposed by Guo et al. [25] in 2008 which uses lower-level structures as the basic cells to construct higher-level structures. DCell_0 is the lowest cell in DCell, which is composed of one n-port switch and n servers.[1] The switch is used to connect all of the n servers in the DCell_0. Let DCell_k be a level-k DCell. Firstly, DCell_1 is built from a few DCell_0s. Each DCell_1 has $(n + 1)$ DCell_0s, and each server of every DCell_0 in a DCell_1 is connected to a server in another DCell_0, respectively. Therefore, the DCell_0s are connected, with only one link between every pair of DCell_0s. A similar method is applied to build a DCell_k from a few DCell_{k-1}s. In a DCell_k, each server will finally have $k + 1$ links: the first link or the level-0 link connected to a switch when forming a DCell_0, and level-i link connected to a server in the same DCell_i but a different DCell_{i-1}. Suppose that each DCell_{k-1} has t_{k-1} servers, then a DCell_k will consist of t_k DCell_{k-1}s, and thus $t_{k-1} \cdot t_k$ servers. Obviously, we have $t_k = t_{k-1} \cdot (t_{k-1} + 1)$. Figure 6.3 shows a DCell_1 with $n = 4$.

DCell satisfies the basic requirement of DCNs such as scalability, fault tolerance, and increased network capacity. The main idea behind DCell not only depends on switches but also takes the advantage of the network interface card (NIC) deployed within servers to design the topology. The number of servers in a DCell increases double-exponentially with the number of server NIC ports. A level-3 DCell can support 3,263,442 servers with 4-port servers and 6-port switches. DCell also overcomes the constraint of a single point of failure as in tree-based topologies.

BCube: To overcome the issue of traffic congestion bottleneck and NIC installation, Guo et al. proposed a new hypercube-based topology known as BCube for shipping-container-based modular data centers [26]. It is also considered a module version of DCell. The most basic element of a BCube, which is named as BCube_0, is also the same as a DCell_0: n servers connected to one n-port switch. The main difference between BCube and DCell lies in how they scale up. BCube makes use of more switches when building higher-level structures. While building a BCube_1, n extra switches are used, connecting to exactly one server in each BCube_0. Consequently, a BCube_1 contains n BCube_0s and n extra switches, which means if the switches in the BCube_0s are considered, there are $2n$ switches in a BCube_1. In general, a BCube_k is built from n BCube_{k-1}s and n^k extra n-port switches. These

[1] Unlike Fat-tree and VL2, n is used to represent the number of ports in DCell and BCube.

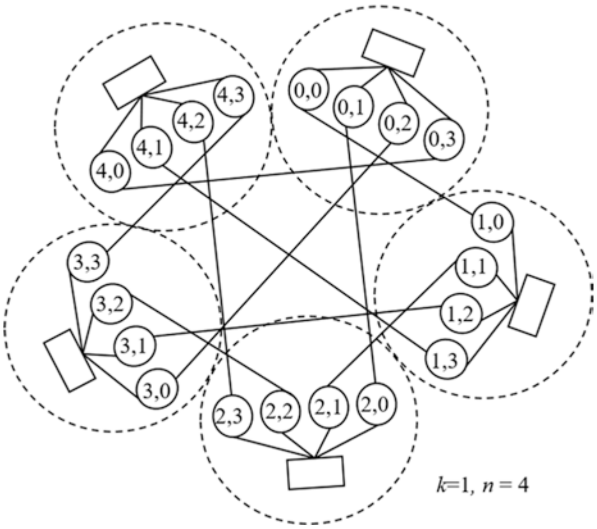

Fig. 6.3 The DCell topology with $n = 4$ ports

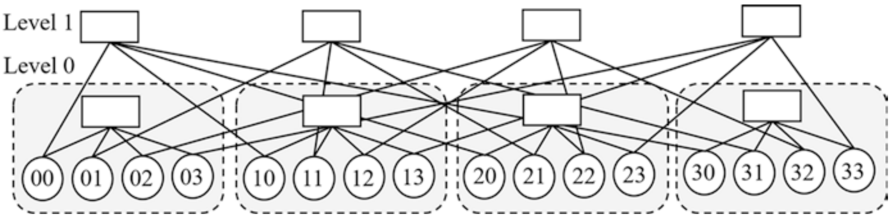

Fig. 6.4 The BCube topology with $n = 4$ ports

extra switches are connected to only one server in each BCube$_{k-1}$. In a level-k BCube, each level requires n^k n-port switches. BCube makes use of more switches when building higher-level structures, while DCell uses only level-0 n-port switches. However, both BCube and DCell require servers to have $k + 1$ NICs. The implication is that servers will be involved in switching more packets in DCell than in BCube. Figure 6.4 shows a BCube$_1$ with $n = 4$.

Just like DCell, the number of levels in a BCube depends on the number of ports on the servers. The number of servers in BCube grows exponentially with the levels, much slower than DCell. For example, when $n = 6$, $k = 3$, a fully constructed BCube can contain 1296 servers. Considering that BCube is designed for container-based data centers, such scalability is sufficient.

6.2.3 Data Center Network Topologies Originated from Interconnection Networks

Some of the DCN topologies originate from the interconnection network topologies, which have been applied to connect the multiple processors in the supercomputing domain. The typical examples include Clos [34], FBFLY [35], *Symbiotic* [36], Hyper-BCube [37], and so on, some of which are switch-centric topologies and the others are server-centric ones.

FBFLY: FBFLY [35] was proposed by Abts et al., which originates from the flattened butterfly, a cost-efficient topology for high-radix switches [38]. The FBFLY k-ary n-flat topology takes advantage of recent high-radix switches to create a scalable but low-diameter network. FBFLY is a multidimensional directed network, similar to a k-ary n-cube torus. Each high-radix switch with more than 64 ports interconnects servers and other switches to form a generalized multidimensional hypercube. A k-ary n-flat FBFLY is derived from a k-ary n-fly conventional butterfly. The number of supported servers is $N = k^n$ in both networks. The number of switches is $n \cdot k^{n-1}$ with port number $2k$ in the conventional butterfly and is $\frac{N}{k} = k^{n-1}$ with port number $n(k - 1) + 1$ in FBFLY. The dimension of FBFLY is $n - 1$. Figure 6.5 shows an 8-ary 2-flat FBFLY topology with 15-port switches. Here, c is the abbreviation for concentration, which means the number of servers and $c = 8$. Although it is similar to a generalized hypercube, FBFLY is more scalable and can save energy by modestly increasing the level of oversubscription. The size of FBFLY can scale from the original size of $8^4 = 4096$ to $c \cdot k^{n-1} = 6144$. The level of oversubscription is moderately raised from 1:1 to 3:2.

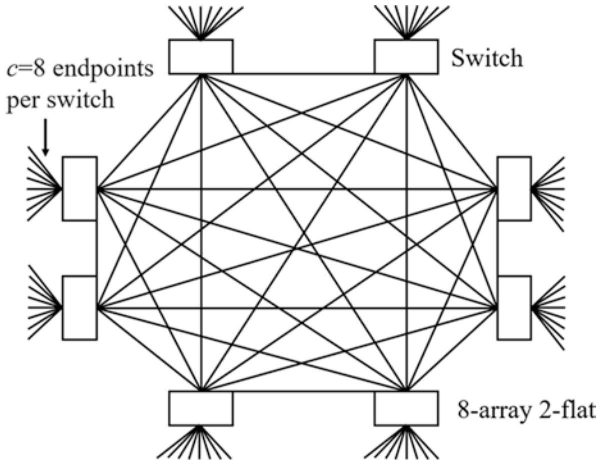

Fig. 6.5 The 8-ary 2-flat FBFLY topology

Symbiotic: Symbiotic [36] was proposed by Hussam et al. to build an easier platform for distributed applications in modular data centers. Communication between servers applies a k-ary 3-cube topology, which is also known as a direct-connect 3D torus and is employed by several supercomputers on the TOP500 list. This topology is formed by having each server directly connected to six other servers, and it does not use any switches or routers. In each dimension, k servers logically form a ring. Symbiotic can support up to k^3 servers. Each server is assigned an address, which takes the form of an (x, y, z) coordinate that indicates its relative offset from an arbitrary origin server in the 3D torus. They refer to the address of the server as the server coordinate, and, once assigned, it is fixed for the lifetime of the server.

6.2.4 Optical Data Center Network Topologies

The electronic data center architectures have several constraints and limitations. Therefore, with the ever-increasing bandwidth demand in data centers and the constantly decreasing price of optical switching devices, optical switching is envisioned as a promising solution for DCNs. Besides offering high bandwidth, optical networks have significant flexibility in reconfiguring the topology during operation. Such a feature is important considering the unbalanced and ever-changing traffic patterns in DCNs. Optical DCN can be classified into two categories, i.e., optical/electronic hybrid schemes such as c-Through [39] and Helios [40], and fully optical schemes such as optical switching architecture (OSA) [41].

c-Through: c-Through [39] is a hybrid network architecture that makes use of both electrical packet switching networks and optical circuit switching networks. Therefore, it is made of two parts: a tree-based electrical network part which maintains the connectivity between each pair of ToR switches and a reconfigurable optical network part which offers high bandwidth interconnection between some ToR electrical switches. Due to the relatively high-cost optical network and the high bandwidth of optical links, it is unnecessary and not cost-effective to maintain an optical link between each pair of ToR switches. Instead, c-Through connects each ToR switch to exactly one other ToR switch at a time. Consequently, the high-capacity optical links are offered to pairs of ToR switches transiently according to the traffic demand. The estimation of traffic between ToR switches and reconfiguration of the optical network is implemented by the control plane of the network. Figure 6.6 shows a c-Through network.

To configure the optical network part of c-Through, the traffic between ToR switches should be estimated. c-Through estimates the rack-to-rack traffic demands by observing the occupancy of the TCP socket buffer. Since only one optical link is offered to each ToR switch, the topology should be configured so that the most amount of estimated traffic can be satisfied. In [39], the problem is solved using the max-weight perfect matching algorithm [42]. The topology of the optical network is configured accordingly.

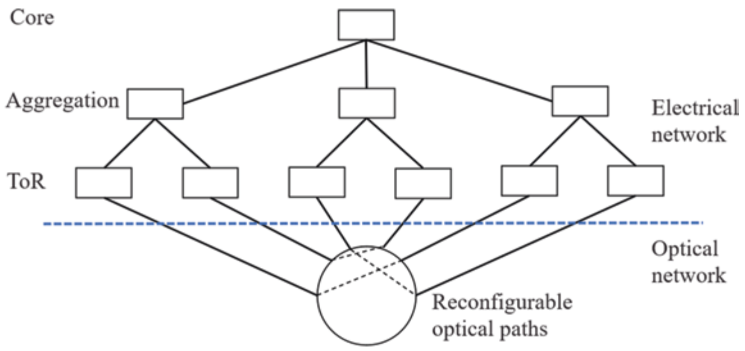

Fig. 6.6 The c-Through network topology [39]

Helios: Helios [40] is another hybrid network with both electrical and optical switches, which are organized as two-level multi-rooted ToR switches and core switches. Core switches consist of both electrical switches and optical switches to make full use of the two complementary techniques. Unlike c-Through, Helios uses the electrical packet switching network to distribute the bursty traffic, while the optical circuit switching part offers baseline bandwidth to the slow-changing traffic. On each of the pod switches, the uplinks are equipped with an optical transceiver. Half of the uplinks are connected to the electrical switches, while the other half are connected to the optical switch through an optical multiplexer. The multiplexer combines the links connected to it to be a "superlink" and enables flexible bandwidth assignment on this superlink.

Helios estimates bandwidth demand using the max-min fairness algorithm [43] among the traffic flows measured on pod switches, which allocates fair bandwidth for TCP flows. Similar to c-Through, Helios computes optical path configuration based on max-weighted matching. Unlike c-Through, both the electrical switches and the optical switches are dynamically configured based on the computed configuration. Helios decides where to forward traffic, the electrical network or the optical network.

OSA: Unlike optical/electronic hybrid schemes, OSA [41] explores the feasibility of a pure optical switching network, which means that it abandons the electrical core switches and use only optical switches to construct the switching core. The ToR switches are still electrical, converting electrical and optical signals between servers and the switching core. OSA allows multiple connections to the switching core on each ToR switch. However, the connection pattern is determined flexibly according to the traffic demand. Since the network does not ensure a direct optical link between each pair of racks with traffic demand, the controlling system constructs the topology to make it a connected graph, and ToR switches are responsible to relay the traffic between other ToR switches.

OSA estimates the traffic demand with the same method as Helios. However in OSA, there can be multiple optical links offered to each rack, and the problem can no longer be formulated as the max-weight matching problem. It is a multi-commodity flow problem with degree constraints, which is NP-hard. In OSA, the problem is simplified as a max-weight b-matching problem and approximately solved by applying multiple max-weight matching.

Compared with electronic switching, optical switching has potentially higher transmission speed, more flexible topology, and lower cooling cost, so it is an important research direction of DCNs. However, the existing optical DCN, such as OSA, still faces some problems. For example, the current design of OSA is for container data centers (i.e., small data centers that can be quickly deployed for edge computing applications in the IoT world), and its scale is limited. To design and build large-scale DCNs is very challenging from the perspective of both architecture and management. Also, the ToR switch used in OSA is the switch that supports optical transmission and electric transmission at the same time, which makes it difficult to be compatible with pure electric switch in traditional data centers.

6.2.5 Wireless Data Center Network Topologies

Wireless/wired hybrid topologies: Wireless technology has the flexibility to adjust the topologies without rewiring. Therefore, Ramachandran et al. introduced wireless technology into DCNs in 2008. Subsequently, Kandula et al. designed Flyways [44, 45], by adding wireless links between the ToR switches to alleviate the rack congestion problem to minimize the maximum transmission time. However, it is difficult for the wireless network to meet all the requirements of DCNs by itself, including scalability, high capacity, and fault tolerance. For example, the capacity of wireless links is often limited due to interference and high transmission load. Therefore, Cui et al. introduced wireless transmission to alleviate the congestion of the hotspot servers, which took wireless communication as a supplement to wired transmission, and proposed a heterogeneous Ethernet/wireless architecture, which is called WDCN [46]. In order not to introduce too many antennas and interfere with each other, Cui et al. regarded each rack as a wireless transmission unit (WTU). This design makes the rack not block the line of sight transmission.

The wireless link scheduling mechanism proposed by Cui et al. includes two parts: collecting traffic demand and link scheduling. A specific server in a WTU is designated as the unit head of the WTU. The unit head is responsible for collecting local traffic information and executing the scheduling algorithm. Each unit head is equipped with a control antenna, and all unit heads broadcast their traffic load in push mode through a common 2.4/5 GHz channel. Therefore, all units can get global traffic load distribution and can schedule the wireless link independently. After collecting the traffic demand information, the head server needs to allocate channels for wireless transmission. Cui et al. proposed a heuristic allocation method [46] to achieve this goal.

The application of wireless technology makes the DCN topologies no longer fixed and saves the complex wiring work, so it has a certain application prospect in the DCN environment. The introduction of Flyways and WDCN alleviates the bandwidth problem of hot servers, and achieved certain results, and made the traffic demand and wireless link scheduling become the focus of research. However, under the premise of providing enough bandwidth, the transmission distance of wireless technology is limited, which limits its deployment in large-scale data centers. Besides, WDCN uses the broadcast method to collect traffic demand, which makes it face the problems of clock synchronization and high communication overhead. Moreover, the measurement results show that the data center traffic is constantly changing, and this makes the location of the hot servers uncertain, which poses a greater challenge to topology adjustment.

All wireless DCN topologies: Based on the 60 GHz wireless communication technology, Shin et al. proposed a DCN with all wireless architecture [45, 47]. They aggregated the switch fabric to the server nodes and expected to arrange the server nodes to be closely connected, low stretch, and support failure recovery. To achieve this requirement, the network card of each server was replaced by Y-switch [47]. The servers are also arranged in a cylindrical rack, so that the communication channels between and within the racks can be easily established, and these connections together form a closely linked network topology.

The topology is modeled as a mesh of Cayley graphs [48]. When viewed from the top, connections within a story of the rack form a 20-node, degree-k Cayley graph, where k depends on the signal's radiation angle (Fig. 6.7). This densely connected graph provides numerous redundant paths from one server to multiple servers in the same rack and ensures strong connectivity. The transceivers on the exterior of the rack stitch together Cayley sub-graphs in different racks. There is great flexibility in how a data center can be constructed out of these racks, but they pick the simplest possible topology by placing the racks in rows and columns for ease of maintenance. Figure 6.7 illustrates an example of the two-dimensional connectivity of four racks in 2 by 2 grids: small black dots represent the transceivers and the lines indicate the connectivity. A Cayley graph sits in the center of each rack: lines coming out of the Cayley graphs are connections through the Y-switches. Relatively long lines connecting the transceivers on the exterior of the racks show the wireless inter-rack connections. Further, since the wireless signal spreads in a cone shape, a transceiver is able to reach other servers in different stories in the same or different racks.

There are still many problems in the scalability and performance of all wireless DCNs. First of all, the competition of the MAC layer greatly affects the performance of the system. Secondly, the performance of the wireless network is greatly affected by the number of network hops. Finally, the performance of multi-hop restricts the scalability of the Cayley data center. However, the advantages and continuous development of wireless technology make it possible to build small- and medium-sized data centers for specific applications.

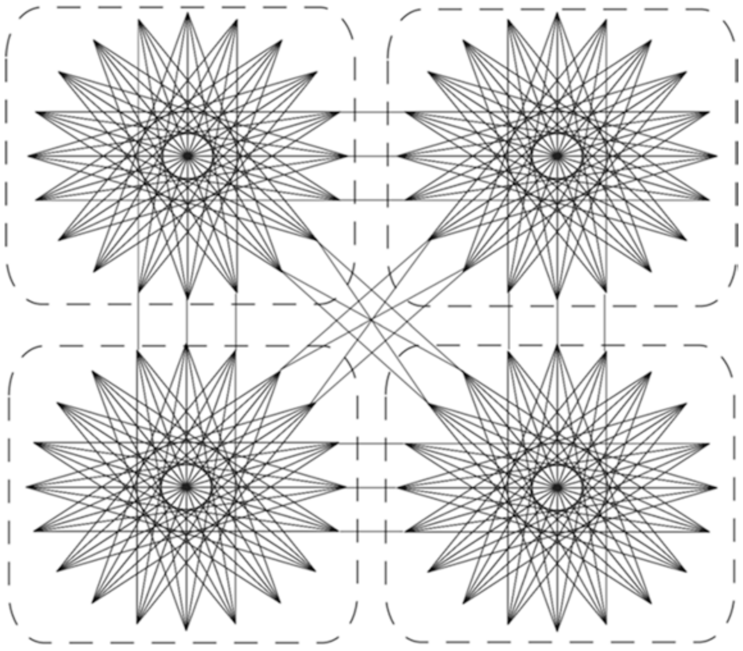

Fig. 6.7 The Cayley data center topology [47]

6.2.6 *Production Data Center Network Topologies*

Nowadays, production data centers have become indispensable for the service providers, such as Google, Facebook, Microsoft, Amazon, and Apple. Large IT companies built several production data centers to support their business. Others rented out to provide services to medium-sized and small-sized enterprises that cannot afford their own data centers.

Google's Jupiter: Singh et al. introduced Google's five generations of DCNs based on Clos topology in the last 10 years [49]. The newest-generation Jupiter is a 40G datacenter-scale fabric equipped with dense 40G capable merchant silicon. Centauri switch is employed as a ToR switch, which includes four switch chips. Four Centauris composed a Middle Block (MB) for use in the aggregation block. The logical topology of an MB is a two-stage network. Each ToR chip connects to 8 MBs with 2 × 10G links to form an aggregation block. Six Centauris are used to build a spine block. There are 256 spine blocks and 64 aggregation blocks in Jupiter.

Facebook's next-generation data center fabric: After the "four-post" architecture [51], Facebook proposed its next-generation data center fabric [50]. As is shown in Fig. 6.8, the basic building block of this data center fabric is the server pod, which is composed of 4 fabric switches and 48 rack switches. The network scale can be extended by increasing the number of server pods. To implement building-wide

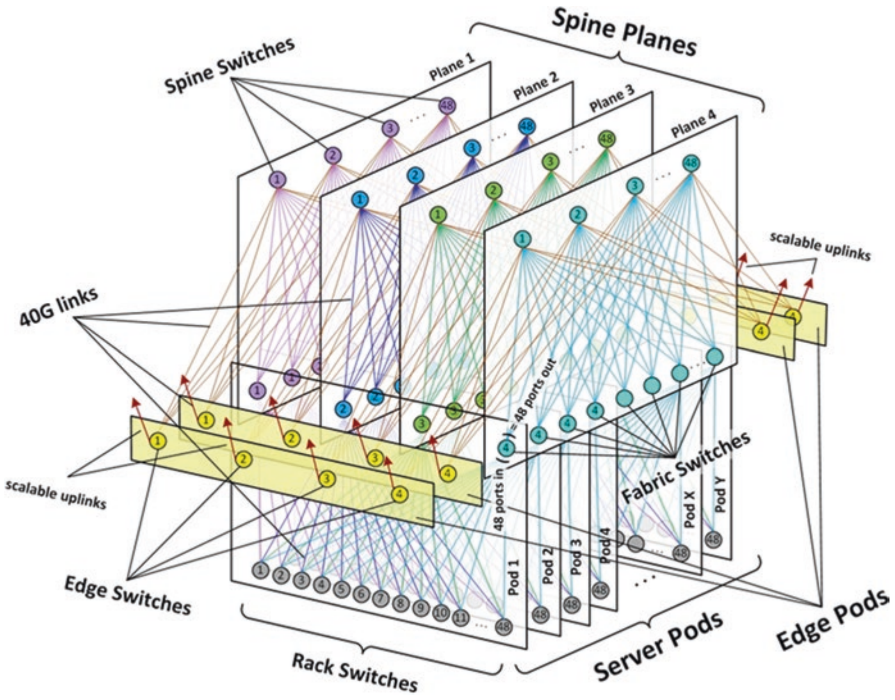

Fig. 6.8 The Facebook data center fabric topology [50]

connectivity, 4 independent planes have been created, each scalable up to 48 spine switches within a plane. Each fabric switch in each server pod connects to each spine switch within its local plane.

Both Google's Jupiter and Facebook's next-generation data center fabrics are based on switch-centric DCN topologies, so they have similar characteristics which are those of switch-centric topologies.

6.3 Operations and Optimizations in Data Center Networks

In a dynamic environment characterized by demand uncertainties, high energy consumption, low average resource utilization, and ever-changing technologies, how to optimize the performance and efficiency of data center operations is a continuous challenge. This section presents a series of operation and optimization schemes that are implemented to enable secure, on-demand, and highly automated services in data centers.

6.3.1 Data Forwarding and Routing

On the basis of a given DCN topology, data forwarding and routing schemes determine how data packets are delivered from a sender to a receiver. Due to the unique features of DCNs, data forwarding and routing are not the same as other networks, e.g., Internet or LAN.

1. Hyperscale data centers have hundreds of thousands switches that interconnect millions of servers. Operating at such a large scale exerts great pressure on forwarding and routing schemes. Unlike the Internet, the sender already knows the topology of DCNs. So the sender also knows the number of available paths to the receiver. How to use the regularities of the DCN topology to scale up forwarding and routing is a critical challenge.
2. People deploy data centers to deliver communication- and computation-intensive services, such as web searching, video content distribution, and big data analytics on a large scale. All these applications are delay sensitive. Major cloud providers, e.g., Amazon and Google, have reported that a slight increase in the service latency may cause observable fewer user accesses and thus a considerable revenue loss [52]. This means forwarding and routing schemes should be highly efficient with less introduced overheads when handling intensive data requests.
3. Servers in data centers may experience downtime frequently [53]. All these failures should be transparent to the client. Therefore, agility has become increasingly important in modern DCNs as there is a requirement that any servers can provide services to any kinds of applications.[2] Facing network crash or server failure, data forwarding and routing schemes should be able to route traffic using alternate paths without interrupting the running application.

Currently, data forwarding and routing schemes have been extensively investigated for DCNs [15, 16, 25, 26, 36, 43, 54–56], which can be classified into different categories based on different criteria. A summary of important data forwarding and routing schemes is presented in Table 6.1, which shows the comparison of these schemes based on five criteria which are briefly described as follows:

Topology: Is the scheme designed for a particular data center topology or can be applied to a generic network?

Implementation: Is the scheme performed in a distributed or centralized manner?

Structure: Which component is involved in packet forwarding with the proposed routing scheme? Typically, forwarding occurs at switches. But in several designs, servers perform forwarding between their equipped network interface cards (NICs).

[2] If a specific number of servers are fixed for specific applications without agility, data centers may operate at low resource utilization with growing and variable demands.

Table 6.1 Summary of data forwarding and routing schemes in DCNs

Scheme	Topology	Implementation	Structure	Traffic	Stack
M. Al-Fares [15]	Fat-tree	Centralized	Switch-centric	Unicast	L3
DCell [25]	Custom	Distributed	Server-centric	Unicast	L3
BCube [26]	Hypercube	Distributed	Server-centric	Unicast	L3
VL2 [16]	Clos	Distributed	Switch-centric	Unicast	L2.5
PortLand [54]	Fat-tree	Centralized	Switch-centric	Multicast	L2
Hedera [43]	Fat-tree	Centralized	Switch-centric	Multicast	L3
Symbiotic [36]	CamCube	Distributed	Switch-centric	Multicast	L3
XPath [55]	Generic	Centralized	Switch-centric	Unicast	L3
FatPaths [56]	Custom	Distributed	Switch-centric	Unicast	L2
PBARA [57, 58]	Generic	Distributed	Both	Unicast	L3

Traffic: Can the scheme support unicast or multicast?

Stack: Is the scheme performed at Layer-2 (Ethernet) or Layer 2.5 (shim layer) or Layer-3 (network) of the TCP/IP stack?

Then, we introduce the data forwarding and routing schemes in detail.

Fat-tree Architecture by M. Al-Fares et al. [15]: This work presented Fat-tree topology, which uses interconnected identical commodity Ethernet switches for full end-to-end bisection bandwidth. A Fat-tree contains k pods and $\left(\frac{k}{2}\right)^2$ core switches. Each pod is assigned with $\frac{k}{2}$ edge switches and $\frac{k}{2}$ aggregation switches. Each edge switch is connected with $\frac{k}{2}$ servers. A novel addressing scheme is designed to specify the position of a switch or a server in the Fat-tree. The core and edge/aggregation switches are addressed in the form of $10.\ n.\ j.\ i,\ i, j \in \left[1, \frac{k}{2}\right]$, and 10.pod.switch.1, $pod, switch \in [0, k-1]$, respectively. The $\frac{k}{2}$ servers connected to an edge switch are addressed as 10.pod.switch.server, $server \in \left[2, \frac{k}{2}+1\right]$. In Fat-tree, all inter-pod traffic is transmitted through the core switches. For load balancing, the proposed data forwarding and routing scheme evenly allocates the traffic among core switches. To achieve this goal, a centralized controller is deployed to maintain two-level forwarding tables at all routers. In this way, the data forwarding is simplified to a static two-level table lookup process, with no need to execute the scheme when a packet arrives. However, how to handle failure is not provided in this work. Another limitation is that the wiring cost is high due to the Fat-tree topology.

DCell [25]: As the first recursive DCN topology, DCell addresses its server with a $(k + 1)$-tuple $\{a_k, a_{k-1}, \ldots, a_1, a_0\}$, where a_k indicates the server belongs to which DCell$_k$, and a_0 denotes the number of servers in DCell$_0$. Let t_k denotes the number of servers in DCell$_k$. Each server is assigned with a unique ID $uid_k = a_0 + \sum_{j=1}^{k} a_j \times t_{j-1}$.

The server in DCell$_k$ can be recursively defined as $[a_k, uid_k]$. Then, DCellRouting, a divide-and-conquer-based routing scheme, is proposed with the consideration of server failures. We assume that the sender and receiver belong to different DCell$_{k-1}$ but are in the same DCell$_k$. The link $\{n_1, n_2\}$ between two DCell$_{k-1}$ can be obtained. The sub-paths are: from the sender to n_1, from n_1 to n_2, and from n_2 to the receiver. We repeat the above process until all sub-paths are direct links. The shortcoming is that DCell cannot guarantee the shortest path can be obtained.

BCube [26]: As a recursive topology, the server in BCube is similarly identified with $\{a_k, \ldots, a_1, a_0\}$. We have $a_i \in [0, n-1]$ as BCube$_0$ contains n servers and BCube$_k$ contains n BCube$_k$. The server address is defined as $\text{baddr} = \sum_{i=0}^{k} a_i \times n^i$. Similarly, the switch is identified as $\{l, s_{k-1}, \ldots, s_0\}$, where l denotes the level of switch, $s_j \in [0, n-1], j \in [0, k-1], 0 \le l \le k$. With this design, servers connected to the same switch only have a single-digit difference in the addresses. For data forwarding, the relay node will calculate the difference in the addresses and change one digit in each step. Theoretical analysis shows that a data packet needs to be transmitted with $k + 1$ switches and k servers at most in routing. Compared with DCell and Fat-tree, BCube provides better network throughput and fault tolerance performance. However, the scalability of BCube is limited. For example, with $k = 3$, $n = 8$, only 4096 servers are supported.

VL2 [16]: VL2 is a flexible and scalable architecture based on the Clos topology, which provides uniform high capacity among servers in the data center. For flexible addressing, VL2 assigns IP addresses based on the actual service requirements. To achieve this goal, two types of IP addresses are designed, i.e., location-specific IP addresses (LA) for network devices (e.g., switches and interfaces) and application-specific IP addresses (AA) for applications. LA is used for routing, which varies if the physical location of the server changes or VMs migrate. In contrast, AA is assigned to an application, which remains unchanged. Each AA is mapped to an LA. A centralized directory system is deployed to maintain the mapping information. VL2 designs a new 2.5 shim layer into the classic network protocol stack to answer the queries from users. The shim layer replaces AA with LA when sending packets and replaces LA back to AA when receiving packets. The shortcoming of VL2 is that it does not provide the bandwidth guarantee for real-time applications.

PortLand [54] and Hedera [43]: PortLand is a scalable, fault-tolerant layer 2 forwarding and routing protocol for data center networks. Observing that data centers are with known baseline topologies and growth models, PortLand adopts a centralized fabric manager to store the network configuration. A 48-bit hierarchical Pseudo MAC (PMAC) addressing scheme is proposed for efficient routing and forwarding. Each host (physical or virtual) is assigned with a unique PMAC, encoding the physical location and the actual MAC (AMAC) address. For routing, when an edge switch receives a data packet, it sends the Address Resolution Protocol (ARP) request for IP to MAC mapping. The request is sent to the fabric manager for its PMAC. After receiving the PMAC address, the edge switch creates an ARP reply and sends it to the source node for further routing. As an extension work, Hedera implementation augments the PortLand routing and fault tolerance protocols. Hedera tries to maximize the utilization of bisection bandwidth with the least scheduling overhead in DCNs. Flow information from constituent switches is collected, which is used to compute nonconflicting paths. This instructs switches to reroute traffic accordingly. The shortcoming is that Portland and Hedera only maintain a single fabric manager, which is at risk of malicious attacks.

Symbiotic [36]: Symbiotic routing allows applications to implement their routing services in the CamCube topology. CamCube [59] directly connects a server with several other servers without any switches or routers, forming a 3D torus topology which uses content addressable network (CAN). Symbiotic routing provides a key-based routing service, where the key space is a 3D wrapped coordinate space. Each server is assigned an (x, y, z) coordinate which determines the location of the server. With the key-based routing, all packets are directed toward the receiver. Each application is assigned with a unique ID on each server. When a data packet arrives at the server, the kernel determines which application the packet belongs to and then queues the packet for execution. The limitation of symbiotic routing is that it requires a CamCube topology; thus, it may not be applicable to other widely used data center network topologies.

XPath [55]: By using existing commodity switches, XPath explicitly controls the routing path at the flow level for well-structured network topologies, e.g., Fat-tree, BCube, and VL2. To address the scalability and deployment challenges of routing path control schemes, e.g., source routing [60], MPLS [61], and OpenFlow [62], XPath preinstalls all desired paths between any two nodes into IP TCAM tables of commodity switches. As the number of all possible paths can be extremely large, a two-step compression algorithm is proposed, i.e., path set aggregation and path ID assignment for prefix aggregation, compressing all paths to a practical number of routing entries for commodity switches. The limitation of XPath is that it needs to contact the centralized controller for every new flow in the DCN to obtain the corresponding path ID.

FatPaths [56]: FatPaths is a simple, generic, and robust routing architecture for low-diameter networks with Ethernet stacks. Currently, many low-diameter topologies, e.g., Slim Fly [63], Jellyfish [23], and Xpander [64], have been designed to improve the cost-performance trade-off when compared with the most commonly used Clos topologies. For example, Slim Fly is 2× more cost- and power-efficient than Fat-tree and Clos while reducing the latency by about 25%. However, traditional routing schemes are not suitable for low-diameter topologies. As only one shortest path exists between any pair of servers, limiting data flow to the shortest path does not utilize the path diversity, which may incur network congestion. FatPaths improves the path diversity of low-diameter topologies by using both minimal and non-minimal paths. Then, the transport layer is redesigned based on new advances for Fat-tree topology [65], removing all TCP performance issues for ultimately low latency and high bandwidth. Furthermore, flowlet switching is used to prevent packet reordering in TCP, ensuring simple and effective load balancing.

PBARA [57, 58]: Port-based addressing and routing architecture (PBARA) applies a kind of port-based source-routing address for forwarding, which renders the table-lookup operation unnecessary in the switches. By leveraging the characteristics of this addressing scheme and the regularity of the DCN topologies, a simple and efficient routing mechanism is proposed, which is completely distributed among the servers, without switch involvement, control message interaction, and topology information storage. The load-balancing and fault-tolerance mechanisms have been designed based on the addressing and routing schemes. The PBARA architecture places all the control functions in servers and keeps the switches very simple. The switches can really realize the goal of no forwarding tables, state maintenance, or configuration, and can implement high-speed packet forwarding through hardware. It may lead to the innovation of the switch architecture. Therefore, it can improve the forwarding performance and reduce the cost and energy consumption of the entire DCN. The PBARA architecture is originally proposed based on the Fat-tree topology [57] and then is extended to the other DCN topologies including F10, Facebook's next-generation data center fabric and DCell [58], which shows that it can be used in a generic DCN topology.

6.3.2 Traffic Optimization

The characteristics of data traffic determine how network protocols are designed for efficient operation in DCNs. Most workloads in data centers fall into two categories: online transaction processing (OLTP) and batch processing. Typically, OLTP is characterized by different kinds of queries that need to analyze massive data sets, e.g., web search. The characteristics of OLTP traffic can be summarized as follows:

Patterns: Most of the data exchanging traffic happens within servers on the same rack. To be more specific, the probability of exchanging traffic for server pairs inside the same rack is 22× higher than that for server pairs in different racks [66]. The inter-rack traffic is incurred by the distributed query processing, e.g., MapReduce applications, where a server pushes or pulls data to many servers across the cluster. For example, the server either does not talk to any server outside its rack or talks to about 1–10% of servers outside its rack [66].

Congestion: Data center experiences network congestion frequently. For example, 86% links observe congestion lasting at least 10 s and 15% of them observe congestion lasting at least 100 s [66]. Furthermore, most congestions are short-lived. Over 90% of congestions are no longer than 2 s while the longest lasted for 382 s [66].

Flow Characteristics: Most of the flows are short flows, but most of the bytes are from long flows [67].

Batch processing is the execution of jobs, e.g., big data analytics, that can run without end user interaction. Each job consists of one or more tasks, which use a direct acyclic graph (DAG) as the execution plan if task dependencies exist. Further, each task creates one or multiple instances for parallel execution. In a recently released Alibaba trace [68, 69], the percentages of jobs with single task and tasks with single instance are over 23% and 43%, respectively. Furthermore, 11% jobs only create one instance. In addition, 99% jobs have less than 28 tasks and 95% tasks have less than 1000 instances. The complexity of batch jobs brings more pressure on the data center operation. The system requires a decent amount of expenses in the beginning, which needs to be trained to understand how to schedule the traffic of batch jobs. Furthermore, debugging these systems can be tricky.

Traffic optimization in DCNs has been explored extensively, which mainly focuses on load balancing and congestion control. The main goal of traffic optimization in DCNs is the flow scheduling to improve network utilization and other quality of service (QoS) parameters (e.g., delay, jitter, data loss, etc.), which can be categorized into link-based and server-based scheduling [11].

6.3.2.1 Link-Based Scheduling

Link-based scheduling schemes aim to balance traffic among links, which contain two crucial procedures, i.e., congestion information collection and path selection.

Congestion Information Collection: Existing scheduling schemes use explicit congestion notification (e.g., FlowBender [70] and CLOVE [71]), data sending rate (e.g., Hedera [43], CONGA [72], HULA [73], and Freeway [74]), or queue length at switch (DeTail [75] and Drill [76]) to represent the congestion information.

Path Selection: With the congestion information, the scheduling schemes determine the paths of data flows. The static ECMP [77] scheme randomly hashes the flows to one of the equal-cost paths, which is convenient to be deployed in DCNs. However, this random scheme cannot properly schedule large flows, which easily incurs network congestions. Some schemes, e.g., CONGA [72], HULA [73], DeTail [75], and Freeway [74], forward flows to the least congested paths. CONGA [72] is a distributed load balancing scheme which obtains the global congestion information among leaf switches. The TCP flows are split into flowlets to achieve fine-grained load balancing. The flowlet is assigned to the least congested path based on the congestion table maintained by each leaf switch. However, due to the limited size of switch memory, CONGA is not scalable to large topologies as only a small amount of congestion information can be maintained. To address this limitation, HULA [73] only tracks congestion for the best path to the destination in each switch. Furthermore, HULA is designed for programmable switches without requiring custom hardware. DeTail [75] presents a new cross-layer network stack to reduce the long tail of flow completion times (FCT). In the link layer, a combined input/output queue is used to sort packets based on their priority in each switch. In the network layer, local egress queue occupancies indicate network congestion. In the transport layer, the out-of-order packets are processed. In the application layer, the flow priorities are determined based on the deadlines. However, as the network stack is modified, DeTail may not be suitable for traditional hardware. Freeway [74] divides the traffic into long-lived elephant flows and latency-sensitive mice flows. According to the path utilization, the shortest paths are dynamically divided into low-latency paths for mice flows and high-throughput paths for elephant flows. The limitation of these schemes is that the precise path utilization is hard to be obtained in real time.

To relieve the requirement of precise path utilization information, some other methods, e.g., Hedera [43], Multipath TCP (MPTCP) [78], and Drill [76], forward flows to the less congested paths. Hedera [43] is a centralized scheme for passive load balancing. Hedera detects large flows in DCNs[3] and assigns them to the paths with enough resources for the maximization of path utilization. However, Hedera is not optimal as the large flows which do not reach the rate limitation may not be scheduled to proper paths. To improve throughput, MPTCP [78] establishes multiple TCP connections between servers by using different IP addresses or ports. Data flows are stripped into subflows for parallel transmission. However, MPTCP may increase the FCT with more additional TCP connections. With no need to collect global congestion information, Drill [76] uses local queue occupancies at each switch and randomized algorithms for load balancing. When a data packet arrives, DRILL randomly chooses two other available ports and compares them with the port which transmits the last packet. The port with the smallest queue size is selected. Drill is a scalable scheme for large-scale data center network topologies as (1) no extra overhead is introduced, and (2) there is no need to modify the existing hardware and protocols.

Some other schemes, e.g., DRB [79] and CLOVE [71], adopt the weighted-round-robin method to achieve load balancing. By utilizing the characteristics of Fat-tree and VL2 topologies, DRB [79] evenly distributes flows among available paths. For each packet in a data flow, the sender selects one of the core switches as the bouncing switch and transmits the data packet to the receiver through that switch. DRB selects the bouncing switch by digit-reversing the IDs of core switches, ensuring that no two successive data packets pass the same path. CLOVE [71] uses ECN and in-band network telemetry (INT) [80] to detect the congestion information and calculate the weight of paths. Then, flowlets are assigned to different paths by rotating the source ports in a weighted round-robin manner.

With the blooming of machine learning techniques in solving complex online optimization problems, recent research efforts proposed learning-based schemes for automatic traffic optimization. CODA [81] utilized an unsupervised clustering scheme to identify the flow information with no need for application modification. AuTO [9] is a two-level deep reinforcement learning (DRL) framework to solve the scalability problem of traffic optimization in DCNs. Due to the non-negligible computation delay, current DRL systems for production data centers (with more than 10^5 servers) cannot handle flow-level traffic optimization as short flows are gone before the decisions come back. AuTO mimics the peripheral and central nervous

[3]The large flows are detected if their rates are larger than 10% of the link capacity.

systems in animals. Peripheral systems are deployed at end-hosts to make traffic optimization decisions locally for short flows. A central system is also deployed to aggregate the global traffic information and make individual traffic optimization decisions for long flows, which are more tolerant of the computation delay. Iroko [82] analyzes the requirements and limitations of applying reinforcement learning in DCNs and then designs an emulator which supports different network topologies for data-driven congestion control.

6.3.2.2 Server-Based Scheduling

Server-based scheduling schemes are designed to balance loads between servers for the improvement of system throughput, resource utilization, and energy efficiency. From the viewpoint of computation, server-based scheduling can be classified into Layer-4- and Layer-7-based schemes. Layer-4 load balancing schemes work at the transport layer, e.g., Transmission Control Protocol (TCP) and User Datagram Protocol (UDP). Without considering application information, the IP address and port information are used to determine the destination of the traffic. Ananta [83] is a Layer-4 load balancer that contains a consensus-based reliable controller and several software multiplexers (Muxes) for a decentralized scale-out data plane. The Mux splits all incoming traffic and realizes encapsulation functionality in software. Google Maglev [84] provides distributed load balancing that uses consistent hashing to distribute packets across the corresponding services. However, software load balancers suffer high latency and low capacity, making them less than ideal for request-intensive and latency-sensitive applications. For further performance improvement, Duet [85] embedded the load balancing functionality into switches and achieved low latency, high availability, and scalability at no extra cost. SilkRoad [86] implements a fast load balancer in a merchant switching ASIC, which can scale to ten million connections simultaneously by using hashing to maintain per-connection state management.

In contrast, Layer-7 load balancers operate at the highest application layer, which are aware of application information to make more complex and informed load balancing decisions. Traditional Layer-7 load balancers are either dedicated hardware middleboxes [87] or can run on virtual machines (VMs) [88]. The key problem of Layer-7 load balancers is that when a load balancing instance fails, the TCP flow state for the client-server connections is lost, which breaks the data flows. To solve this problem, Yoda [89] keeps per-flow TCP state information with a distributed store.

As discussed above, the load balancing can be actively achieved by scheduling computing tasks into appropriate servers. From another viewpoint of storage, traffic optimization can also be passively achieved by optimizing the storage location with data replica placement and erasure code schemes in DCNs. By creating full data copies at storage nodes near end users, data replication can reduce the data service latency with good fault tolerance performance. An intuitive heuristic is hash—hash data and replicas to data centers so as to optimize for load balancing, which has been widely adopted in the distributed storage systems today, such as HDFS [90]. Nevertheless, this simple heuristic is far from ideal as it overlooks the skewness of data requests. Yu et al. [91] designed a hypergraph-based framework for associated data placement (ADP), achieving low data access overheads and load balancing among geo-distributed data centers. However, the centralized ADP is not effective enough in terms of the running time and computation overhead, making it slow to react to the real-time changes in workloads. Facebook Akkio [92] is a data migration scheme, which adapts to the changing data access patterns. To improve the scalability of the solution for petabytes of data, Akkio groups the related data with similar access locality into a migration unit. DataBot [93] is a reinforcement learning-based scheme which adaptively learns optimal data placement policies, reducing the latency of data flows with no future assumption about the data requests. The limitation of data replication is that it suffers from high bandwidth and storage costs with the growing number of replicas.

With erasure codes, each data item is coded into K data chunks and R parity chunks. The original data item can be recovered via the decoding process from any K out of $K + R$ chunks. Compared with replication, erasure codes can lower the bandwidth and storage costs by an order of magnitude while with the same or better level of data reliability. EC-Cache [94] provides a load-balanced, low-latency caching cluster that uses online erasure coding to overcome the limitations of data replication. Hu et al. [95] designed a novel load balancing scheme in coded storage systems. When the original storage node of the requested data chunk becomes a hotspot, degraded reads[4] are proactively and intelligently launched to relieve the burden of the hotspot. Due to the non-negligible decoding overhead, erasure codes may not be suitable for data-intensive applications.

6.4 Future Data Center Networks and Applications

According to the prediction of Oracle Cloud, 80% of all enterprises plan to move their workloads to the cloud data centers [96]. The amount of stored and processed data continues to increase, from 5G and Internet of Things (IoT) devices to emerging technologies, e.g., artificial intelligence, augmented reality, and virtual reality. These new technologies are dramatically reshaping the data center in order to meet the rising demands. However, Forbes reported that only 29% engineers said their data centers can meet the current needs [97]. Here, we list several future data center trends that let network infrastructure meet the ultimate challenges of the upcoming days.

Low Latency: From Milliseconds to Microseconds and Nanoseconds. Currently, a significant part of the communication traffic is within DCNs. Network latency can affect the performance of delay-sensitive applications, e.g., web search, social networks, and key-value stores in a significant manner. The latencies for current

[4]The action of parity chunk retrieval for decoding is defined as degraded read.

DCNs are in the order of milliseconds to hundreds of microseconds, which use (1) the mainstream Hadoop and HDD/SSD as the storage solution, (2) TCP as the communication protocol, and (3) statistical multiplexing as the communication link sharing mechanism. The latencies are planned to be reduced by an order of magnitude to microseconds or even nanoseconds with the evolving of future data center architectures. The storage access latencies have been reduced to tens of microseconds by all using NVMe SSD or even tens of nanoseconds with the emerging storage class memory (SCM). Through network virtualization, the data center is virtualized to a distributed resource pool for scalable all-IP networks, improving the resource utilization. Furthermore, TCP introduces many extra overheads. This means the processors need to spend a lot of time in managing network transfers for data-intensive applications, reducing the overall performance. In contrast, remote direct memory access (RDMA) allows servers to exchange data in the memory without involving either one's processor, cache, or operating system. RDMA is the future of data center storage fabrics to achieve low latency.

In-Network Computing. The newly emerged programmable network devices (e.g., switches, network accelerators, and middleboxes) and the continually increasing traffic motivate the design of in-network computing. In future DCNs, the computing will not start and end at the servers but will be extended into the network fabric. The aggregation functions needed by the data-intensive applications, e.g., big data, graph processing, and stream processing, have the features that make it suitable to be executed in programmable network devices. The total amount of data can be reduced by arithmetic (add) or logical function (minima/maxima detection) that can be parallelized. By offloading computing tasks onto the programmable network devices, we can (1) reduce network traffic and relieve network congestion, (2) serve user requests on the fly with low service latency, and (3) reduce the energy consumption of running servers. How to enable in-network computing inside commodity data centers with complex network topologies and multipath communication is a challenge. The end-to-end principle which has motivated most of the networking paradigms of the past years is challenged when in-network computing devices are inserted on the ingress-egress path.

Data Center Automation. With the explosive growth of traffic and the rapid expansion of businesses in data centers, manual monitoring, configuration, troubleshooting, and remediation are inefficient and may put businesses at risk. Data center automation means the process of network management, e.g., configuration, scheduling, monitoring, maintenance, and application delivery, can be executed without human administration, which increases the operational agility and efficiency. Massive history traces have been accumulated during the operation of data centers. Machine learning, which gives computers the ability to learn from history, is a promising solution to realize data center automation. The purposes of the learning-based data center automation are to (1) provide insights into network devices and servers for automatic configurations, (2) realize adaptive data forwarding and routing according to network changes, (3) automate all scheduling and monitoring tasks, and (4) enforce data center to operate in agreement with standards and policies.

High Reliability and Availability in the Edge. Providing highly available and reliable services has always been an essential part of maintaining customer satisfaction and preventing potential revenue losses. However, downtime is the enemy of all data centers. According to the Global Data Center Survey report in the year 2018, 31% of data center operators reported they experienced a downtime incident or severe service degradation [98]. The time to full recovery for most outages was 1–4 h, with over a third reporting a recovery time of 5 h or longer. The downtime in data centers is costly. It has been reported that about $285 million have been lost yearly due to failures [99]. According to the global reliability survey in 2018, 80% of businesses required a minimum uptime of 99.99% [100]. To achieve high reliability and availability, equipment redundancy is widely utilized in the data center industry. Compared with redundancy in hyperscale data centers, providing services at the edge of the network is attracting increasing attention. In the not-too-distant future, edge data centers are likely to explode as people continue to offload their computing and storage tasks from end devices to centralized facilities. With data being captured from so many different sources, edge data centers are going to become as common as streetlights to ensure high reliability and availability. The hyperscale data centers may work together with edge computing to meet the computing, storage, and latency requirements, which creates both opportunities and threats to the design of the existing system.

References

1. Y. Mansouri, A.N. Toosi, R. Buyya, Data storage management in cloud environments: taxonomy, survey, and future directions. ACM Comput. Surv. **50**(6), 1–51 (2018)
2. D. Reinsel, J. Gantz, J. Rydning, The digitization of the world from edge to core, IDC white paper (2018), [Online]: https://www.seagate.com/em/en/our-story/data-age-2025/
3. W. Xia, P. Zhao, Y. Wen, H. Xie, A survey on data center networking (DCN): infrastructure and operations. IEEE Commun. Surv. Tuts. **19**(1), 640–656 (2017)
4. T. Ye, T.T. Lee, M. Ge, W. Hu, Modular AWG-based interconnection for large-scale data center networks. IEEE Trans. Cloud Comput. **6**(3), 785–799 (2018)
5. N. Jones, How to stop data centres from gobbling up the world's electricity. Nature **561**(7722), 163–167 (2018)
6. A.S.G. Andrae, T. Edler, On global electricity usage of communication technology: trends to 2030. Challenges **6**(1), 117–157 (2015)
7. Power, pollution and the internet (2012), [Online]: https://www.nytimes.com/2012/09/23/technology/data-centers-waste-vast-amounts-of-energy-belying-industry-image.html
8. T. Benson, A. Akella, D.A. Maltz, Network traffic characteristics of data centers in the wild, in *ACM IMC*, 2010, pp. 267–280
9. L. Chen, J. Lingys, K. Chen, F. Liu, AuTO: scaling deep reinforcement learning for datacenter-scale automatic traffic optimization, in *ACM SIGCOMM* (2018), pp. 191–205
10. K. Chen, C. Hu, X. Zhang, K. Zheng, Y. Chen, A.V. Vasilakos, Survey on routing in data centers: insights and future directions. IEEE Netw. **25**(4), 6–10 (2011)
11. J. Zhang, F.R. Yu, S. Wang, T. Huang, Z. Liu, Y. Liu, Load balancing in data center networks: a survey. IEEE Commun. Surv. Tuts. **20**(3), 2324–2352 (2018)

12. E. Baccour, S. Foufou, R. Hamila, A. Erbad, Green data center networks: a holistic survey and design guidelines, in IEEE IWCMC, 2019, pp. 1108–1114
13. C. Kachris, I. Tomkos, A survey on optical interconnects for data centers. IEEE Commun. Surv. Tuts. **14**(4), 1021–1036 (2012)
14. M.F. Bari, R. Boutaba, R. Esteves, L.Z. Granville, M. Podlesny, M.G. Rabbani, Q. Zhang, M.F. Zhani, Data center network virtualization: a survey. IEEE Commun. Surv. Tuts. **15**(2), 909–928 (2013)
15. M. Al-Fares, A. Loukissas, A. Vahdat, A scalable, commodity data center network architecture, in *ACM SIGCOMM*, 2008, pp. 63–74
16. A. Greenberg, J.R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, S. Sengupta, VL2: a scalable and flexible data center network, in *ACM SIGCOMM*, 2009, pp. 51–62
17. Y. Sun, J. Chen, Q. Liu, W. Fang, Diamond: an improved fat-tree architecture for large-scale data centers. J. Commun. **9**(1), 91–98 (2014)
18. M. Walraed-Sullivan, A. Vahdat, K. Marzullo, Aspen trees: balancing data center fault tolerance, scalability and cost, in *ACM CoNEXT*, 2013, pp. 85–96
19. V. Liu, D. Halperin, A. Krishnamurthy, T. Anderson, F10: a fault-tolerant engineered network, in *USENIX NSDI*, 2013, pp. 399–412
20. G. Chen, Y. Zhao, D. Pei, D. Li, Rewiring 2 links is enough: accelerating failure recovery in production data center networks, in *IEEE ICDCS*, 2015, pp. 569–578
21. L. Gyarmati, T.A. Trinh, Scafida: a scale-free network inspired data center architecture. ACM SIGCOMM Comput. Commun. Rev. **40**(5), 4–12 (2010)
22. J.Y. Shin, B. Wong, E.G. Sirer, Small-world datacenters, in *ACM SOCC*, 2011, pp. 1–13
23. A. Singla, C.Y. Hong, L. Popa, P.B. Godfrey, Jellyfish: networking data centers randomly, in *USENIX NSDI*, 2012, pp. 225–238
24. X.Y. Lin, Y.C. Chung, T.Y. Huang, A multiple LID routing scheme for fat-tree-based InfiniBand networks, in *IEEE IPDPS*, 2004, p. 11
25. C. Guo, H. Wu, K. Tan, L. Shi, Y. Zhang, S. Lu, DCell: a scalable and fault-tolerant network structure for data centers, in *ACM SIGCOMM*, 2008, pp. 75–86
26. C. Guo, G. Lu, D. Li, H. Wu, X. Zhang, Y. Shi, C. Tian, Y. Zhang, S. Lu, BCube: a high performance, server-centric network architecture for modular data centers, in *ACM SIGCOMM*, 2009, pp. 63–74
27. D. Li, C. Guo, H. Wu, K. Tan, Y. Zhang, S. Lu, FiConn: using backup port for server interconnection in data centers, in *IEEE INFOCOM*, 2009, pp. 2276–2285
28. Y. Liao, D. Yin, L. Gao, DPillar: scalable dual-port server interconnection for data center networks, in *IEEE ICCCN*, 2010, pp. 1–6
29. C. Wang, C. Wang, Y. Yuan, Y. Wei, MCube: a high performance and fault-tolerant network architecture for data centers, in *IEEE ICCDA*, 2010, pp. 423–427
30. H. Wu, G. Lu, D. Li, C. Guo, Y. Zhang, MDCube: a high performance network structure for modular data center interconnection, in *ACM CoNEXT*, 2009, pp. 25–36
31. L. Huang, Q. Jia, X. Wang, S. Yang, B. Li, PCube: improving power efficiency in data center networks, in *IEEE CLOUD*, 2011, pp. 65–72
32. X. Liu, S. Yang, L. Guo, S. Wang, H. Song, Snowflake: a new-type network structure of data center. Chin. J. Comput. **34**(1), 76–85 (2011)
33. Z. Ding, D. Guo, X. Liu, X. Luo, G. Chen, A MapReduce-supported network structure for data centers. Concurr. Comput. Pract. Exp. **24**(12), 1271–1295 (2012)
34. Y. Liu, J.K. Muppala, M. Veeraraghavan, D. Lin, M. Hamdi, *Data center network topologies: current state-of-the-art* (Springer International Publishing, Cham, 2013), pp. 7–14
35. D. Abts, M.R. Marty, P.M. Wells, P. Klausler, H. Liu, Energy proportional datacenter networks. ACM SIGARCH Comput. Archit. News **38**(3), 338–347 (2010)
36. H. Abu-Libdeh, P. Costa, A. Rowstron, G. O'Shea, A. Donnelly, Symbiotic routing in future data centers, in *ACM SIGCOMM*, 2010, pp. 51–62
37. D. Lin, Y. Liu, M. Hamdi, J. Muppala, Hyper-BCube: a scalable data center network, *IEEE ICC*, 2012, pp. 2918–2923

38. J. Kim, W.J. Dally, D. Abts, Flattened butterfly: a cost-efficient topology for high-radix networks. ACM SIGARCH Comput. Archit. News **35**(2), 126–137 (2007)
39. G. Wang, D. Andersen, M. Kaminsky, K. Papagiannaki, T. Ng, M. Kozuch, M. Ryan, c-Through: part-time optics in data centers. ACM SIGCOMM Comput. Commun. Rev. **40**(4), 327–338 (2010)
40. N. Farrington, G. Porter, S. Radhakrishnan, H. Bazzaz, V. Subramanya, Y. Fainman, G. Papen, A. Vahdat, Helios: a hybrid electrical/optical switch architecture for modular data centers. ACM SIGCOMM Comput. Commun. Rev. **40**(4), 339–350 (2010)
41. K. Chen, A. Singla, A. Singh, K. Ramachandran, L. Xu, Y. Zhang, X. Wen, Y. Chen, OSA: an optical switching architecture for data center networks with unprecedented flexibility, in *USENIX NSDI*, 2012, pp. 239–252
42. J. Edmonds, Paths, trees, and flowers. Can. J. Math. **17**(3), 449–467 (1965)
43. M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, A. Vahdat, Hedera: dynamic flow scheduling for data center networks, in *USENIX NSDI*, 2010, pp. 89–92
44. J.P.S. Kandula, P. Bahl, Flyways to de-congest data center networks, in *ACM HotNets*, 2009, pp. 1–6
45. D. Halperin, S. Kandula, J. Padhye, P. Bahl, D. Wetherall, Augmenting data center networks with multi-gigabit wireless links, in *ACM SIGCOMM*, 2011, pp. 38–49
46. Y. Cui, H. Wang, X. Cheng, B. Chen, Wireless data center networking. IEEE Wirel. Commun. **18**(6), 46–53 (2011)
47. J.Y. Shin, E.G. Sirer, H. Weatherspoon, D. Kirovski, *On the Feasibility of Completely Wireless Data Centers. Technical report* (Cornell University, Ithaca, 2011) [Online]: http://hdl.handle.net/1813/22846
48. A. Cayley, On the theory of groups. Am. J. Math. **11**(2), 139–157 (1889)
49. A. Singh, J. Ong, A. Agarwal, G. Anderson, A. Armistead, R. Bannon, S. Boving, G. Desai, B. Felderman, P. Germano et al., Jupiter rising: a decade of Clos topologies and centralized control in Google's datacenter network, in *ACM SIGCOMM*, 2015, pp. 183–197
50. A. Andreyev, Introducing data center fabric, the next-generation Facebook data center network (2014), [Online]: https://code.facebook.com/posts/360346274145943/
51. N. Farrington, A. Andreyev, Facebook's data center network architecture, in *IEEE OI*, 2013, pp. 49–50
52. Q. Pu, G. Ananthanarayanan, P. Bodik, S. Kandula, A. Akella, P. Bahl, I. Stoica, Low latency geo-distributed data analytics, in *ACM SIGCOMM*, 2015, pp. 421–434
53. O. Khan, R.C. Burns, J.S. Plank, W. Pierce, C. Huang, Rethinking erasure codes for cloud file systems: minimizing I/O for recovery and degraded reads, in *USENIX FAST*, 2012, pp. 20–33
54. R. Mysore, A. Pamboris, N. Farrington, N. Huang, P. Miri, S. Radhakrishnan, A. Vahdat, Portland: a scalable fault-tolerant layer 2 data center network fabric, in *ACM SIGCOMM*, 2009, pp. 39–50
55. S. Hu, K. Chen, H. Wu, W. Bai, C. Lan, H. Wang, C. Guo, Explicit path control in commodity data centers: design and applications, in *USENIX NSDI*, 2015, pp. 15–28
56. M. Besta, M. Schneider, K. Cynk, M. Konieczny, E. Henriksson, S. Di Girolamo, T. Hoefler, FatPaths: routing in supercomputers, data centers, and clouds with low-diameter networks when shortest paths fall short. *arXiv preprint arXiv:1906.10885* (2019)
57. A. Zhao, Z. Liu, J. Pan, M. Liang, A simple, cost-effective addressing and routing architecture for fat-tree based datacenter networks, in *IEEE INFOCOM Workshop on DCPerf*, 2017, pp. 36–41
58. A. Zhao, Z. Liu, J. Pan, M. Liang, A novel addressing and routing architecture for cloud-service datacenter networks. IEEE Trans. Serv. Comput. (2019). https://doi.org/10.1109/TSC.2019.2946164
59. P. Costa, A. Donnelly, G. O'Shea, A. Rowstron, CamCube: a key-based data center, *Technical Report MSR TR-2010-74*, Microsoft Research, 2010
60. C.A. Sunshine, Source routing in computer networks. ACM SIGCOMM Comput. Commun. Rev. **7**(1), 29–33 (1977)

61. E. Rosen, A. Viswanathan, R. Callon, Multiprotocol label switching architecture, RFC 3031 (2001)
62. N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, J. Turner, OpenFlow: enabling innovation in campus networks. ACM SIGCOMM Comput. Commun. Rev. **38**(2), 69–74 (2008)
63. M. Besta, T. Hoefler, Slim fly: a cost effective low-diameter network topology, in *IEEE SC*, 2014, pp. 348–359
64. A. Valadarsky, M. Dinitz, M. Schapira, Xpander: unveiling the secrets of high-performance datacenters, in *ACM HotNets*, 2015, pp. 1–7
65. M. Handley, C. Raiciu, A. Agache, A. Voinescu, A.W. Moore, G. Antichi, M. Wójcik, Re-architecting datacenter networks and stacks for low latency and high performance, in *ACM SIGCOMM*, 2017, pp. 29–42
66. S. Kandula, S. Sengupta, A. Greenberg, P. Patel, R. Chaiken, The nature of data center traffic: measurements & analysis, in *ACM IMC*, 2009, pp. 202–208
67. M. Alizadeh, A. Greenberg, D.A. Maltz, J. Padhye, P. Patel, B. Prabhakar, M. Sridharan, Data center TCP (DCTCP), in *ACM SIGCOMM*, 2010, pp. 63–74
68. Alibaba Cluster Trace, 2018. [Online]: https://github.com/alibaba/clusterdata
69. Q. Liu, Z. Yu, The elasticity and plasticity in semi-containerized co-locating cloud workload: a view from Alibaba trace, in *ACM SoCC*, 2018, pp. 347–360
70. A. Kabbani, B. Vamanan, J. Hasan, F. Duchene, FlowBender: flow-level adaptive routing for improved latency and throughput in datacenter networks, in *ACM CoNEXT*, 2014, pp. 149–160
71. N. Katta, A. Ghag, M. Hira, I. Keslassy, A. Bergman, C. Kim, J. Rexford, CLOVE: congestion-aware load balancing at the virtual edge, in *ACM CoNEXT*, 2017, pp. 323–335
72. M. Alizadeh, T. Edsall, S. Dharmapurikar, R. Vaidyanathan, K. Chu, A. Fingerhut, G. Varghese, CONGA: distributed congestion-aware load balancing for datacenters, in *ACM SIGCOMM*, 2014, pp. 503–514
73. N. Katta, M. Hira, C. Kim, A. Sivaraman, J. Rexford, HULA: scalable load balancing using programmable data planes, in *ACM SOSR*, 2016, pp. 1–12
74. W. Wang, Y. Sun, K. Zheng, M.A. Kaafar, D. Li, Z. Li, Freeway: adaptively isolating the elephant and mice flows on different transmission paths, in *IEEE ICNP*, 2014, pp. 362–367
75. D. Zats, T. Das, P. Mohan, D. Borthakur, R. Katz, DeTail: reducing the flow completion time tail in datacenter networks, in *ACM SIGCOMM*, 2012, pp. 139–150
76. S. Ghorbani, Z. Yang, P.B. Godfrey, Y. Ganjali, A. Firoozshahian, Drill: micro load balancing for low-latency data center networks, in *ACM SIGCOMM*, 2017, pp. 225–238
77. C.E. Hopps, Multipath issues in unicast and multicast next-hop selection, RFC 2991 (2000)
78. C. Raiciu, S. Barre, C. Pluntke, A. Greenhalgh, D. Wischik, M. Handley, Improving datacenter performance and robustness with multipath TCP, in *ACM SIGCOMM*, 2011, pp. 266–277
79. J. Cao, R. Xia, P. Yang, C. Guo, G. Lu, L. Yuan, Y. Zheng, H. Wu, Y. Xiong, D. Maltz, Per-packet load-balanced, low-latency routing for clos-based data center networks, in *ACM CoNEXT*, 2013, pp. 49–60
80. C. Kim, A. Sivaraman, N. Katta, A. Bas, A. Dixit, L.J. Wobkeret, In-band network telemetry via programmable dataplanes, in *ACM SIGCOMM*, 2015
81. H. Zhang, L. Chen, B. Yi, K. Chen, M. Chowdhury, Y. Geng, CODA: toward automatically identifying and scheduling coflows in the dark, in *ACM SIGCOMM*, 2016, pp. 160–173
82. F. Ruffy, M. Przystupa, I. Beschastnikh, Iroko: a framework to prototype reinforcement learning for data center traffic control, in *NIPS*, 2018
83. P. Patel, D. Bansal, L. Yuan, A. Murthy, A. Greenberg, D.A. Maltz, R. Kern, H. Kumar, M. Zikos, H. Wu, C. Kim, N. Karri, Ananta: cloud scale load balancing, in *ACM SIGCOMM*, 2013, pp. 207–218
84. D.E. Eisenbud, C. Yi, C. Contavalli, C. Smith, R. Kononov, E. Mann-Hielscher, A. Cilingiroglu, B. Cheyney, W. Shang, J.D. Hosein, Maglev: a fast and reliable software network load balancer, in *USENIX NSDI*, 2016, pp. 523–535

85. R. Gandhi, H.H. Liu, Y.C. Hu, G. Lu, J. Padhye, L. Yuan, M. Zhang, Duet: cloud scale load balancing with hardware and software, in *ACM SIGCOMM*, 2015, pp. 27–38
86. R. Miao, H. Zeng, C. Kim, J. Lee, M. Yu, Silkroad: making stateful layer-4 load balancing fast and cheap using switching asics, in *ACM SIGCOMM*, 2017, pp. 15–28
87. F5 load balancer, 2020, [Online]: http://www.f5.com
88. HAProxy load balancer, 2020, [Online]: http://haproxy.1wt.eu
89. R. Gandhi, Y. C. Hu, M. Zhang, Yoda: a highly available layer-7 load balancer, in *EuroSys*, 2016, p. 21
90. HDFS Architecture Guide, 2019, [Online]: https://hadoop.apache.org/
91. B. Yu, J. Pan, A framework of hypergraph-based data placement among geo-distributed datacenters. IEEE Trans. Serv. Comput. **13**(3), 395–409 (2020)
92. M. Annamalai, K. Ravichandran, H. Srinivas, I. Zinkovsky, L. Pan, T. Savor, D. Nagle, M. Stumm, Sharding the shards: managing datastore locality at scale with Akkio, in *USENIX OSDI*, 2018, pp. 445–460
93. K. Liu, J. Wang, Z. Liao, B. Yu, J. Pan, Learning-based adaptive data placement for low latency in data center networks, in *IEEE LCN*, 2018, pp. 142–149
94. K.V. Rashmi, M. Chowdhury, J. Kosaian, I. Stoica, K. Ramchandran, EC-cache: load-balanced, low-latency cluster caching with online erasure coding, in *USENIX OSDI*, 2016, pp. 401–417
95. Y. Hu, Y. Wang, B. Liu, D. Niu, C. Huang, Latency reduction and load balancing in coded storage systems, in *ACM SoCC*, 2017, pp. 365–377
96. Oracle's Top 10 Cloud Predictions, 2019, [Online]: https://questoraclecommunity.org/learn/blogs/oracles-2019-top-10-cloud-predictions/
97. The Data Center of The Future, 2020, [Online]: https://www.forbes.com/sites/insights-vertiv/2020/01/22/the-data-center-of-the-future/
98. Data Center Industry Survey Results, 2018, [Online]: https://uptimeinstitute.com/2018-data-center-industry-survey-results
99. B. Snyder, J. Ringenberg, R. Green, V. Devabhaktuni, M. Alam, Evaluation and design of highly reliable and highly utilized cloud computing systems. J Cloud Comput. **4**(1), 1–16 (2015)
100. 80% of businesses now require uptime of 99.99% from their cloud service vendors, 2018, [Online]: https://www.techrepublic.com/article/80-of-businesses-now-require-uptime-of-99-99-from-their-cloud-service-vendors/