A Survey of Data Center Network Architectures

Yang Liu, Jogesh K. Muppala, *Senior Member, IEEE*, Malathi Veeraraghavan, *Senior Member, IEEE*

***Abstract*—Large-scale data centers form the core infrastructure support for the ever expanding cloud based services. Thus the performance and dependability characteristics of data centers will have significant impact on the scalability of these services. In particular, the data center network needs to be agile and reconfigurable in order to respond quickly to ever changing application demands and service requirements. Significant research work has been done on designing the data center network topologies in order to improve the performance of data centers.**

In this paper, we present a survey of data center network designs and topologies that have published recently. We start with a discussion on various representative data center network topologies, and compare them with respect to several properties in order to highlight their advantages and disadvantages. Thereafter, we discuss several routing protocols designed for these topologies, and compare them based on various criteria: the basic algorithms to establish connections, the techniques used to gain better performance and the mechanisms for fault-tolerance. A good understanding of the state-of-the-art in data center networks would enable the design of future architectures in order to improve performance and dependability of data centers.

***Index Terms*—Data Center Network, Topology, Fault Tolerance**

I. Introduction

DATA center infrastructure design has recently been receiving significant research interest both from academia and industry, in no small part due to the growing importance of data centers in supporting and sustaining the rapidly growing Internet-based applications including search (e.g., Google, Bing), video content hosting and distribution (e.g., YouTube, NetFlix), social networking (e.g., Facebook, Twitter), and large-scale computations (e.g., data mining, bioinformatics, indexing). For example, the Microsoft Live online services are supported by a Chicago-based data center, which is one of the largest data centers ever built, spanning more than 700,000 square feet. In particular, cloud computing is characterized as the culmination of the integration of computing and data infrastructures to provide a scalable, agile and cost-effective approach to support the ever-growing critical IT needs (in terms of computation, storage, and applications) of both enterprises and the general public [1] [2].

Massive data centers providing storage form the core of the infrastructure for the cloud [2]. It is thus imperative that the data center infrastructure, including the data center networking is well designed so that both the deployment and maintenance of the infrastructure is cost-effective. With data availability and security at stake, the role of the data center is more critical than ever.

The topology of the network interconnecting the servers has a significant impact on the agility and reconfigurability of the data center infrastructure to respond to changing application demands and service requirements. Today, data center networks primarily use top of rack (ToR) switches that are interconnected through end of rack (EoR) switches, which are in turn connected via core switches. This approach leads to significant bandwidth oversubscription on the links in the network core [3], and prompted several researchers to suggest alternate approaches for scalable cost-effective network architectures. According to the reconfigurability of the topology after the deployment of the DCN, there are fixed architectures and flexible architectures. Fixed architectures can be further classified into to categories: the tree-based architectures such as fat tree [3] and Clos Network [4], and recursive topologies such as DCell [5], BCube [6]. Flexible architectures such as cThrough[7], Helios [8] and OSA [9] enables reconfigurability of their network topology. Every approach is characterized by its unique network architecture, routing algorithms, fault-tolerance and fault recovery approaches.

Our primary focus in this paper is on data center network topologies that have been proposed in research literature. We start with a discussion on various representative data center architectures, and compare them on several dimensions to highlight the advantages and disadvantages of the topology used by each architecture. Thereafter, we discuss the routing protocols designed for these topologies, and compare them based on various criteria, such as the algorithms used to gain better performance and the mechanisms for fault-tolerance. Our goal is to bring out the salient features of the different approaches such that these could be used as guidelines in constructing future architectures and routing techniques for data center networks. The focus of this survey is data center network topologies and related techniques. We notice that other researchers have done thorough surveys on other important issues about data center network, such as routing in data centers[10], and data center virtualization[11]. Kachris et al. did a survey focusing on optical interconnects for data centers in [12], which also mentioned some of the topologies that will be discussed in this paper. Wu et al. made comparisons of some existing DCN architectures in [13]. All the above surveys offer a rich set of work that has been done in the area, which helps to build up this paper.

This paper is organized as follows. First, we briefly review

Manuscript received April XX, XXXX; revised January XX, XXXX.

The work described in this paper has been supported by HK RGC under RGC-Competitive Earmarked Research Grant HKUST 617907

Yang Liu is with the Dept. of Computer Science and Engineering, The Hong Kong University of Science and Technology, Clear Water Bay, Kowloon, Hong Kong (e-mail: liuyangcse@cse.ust.hk).

Jogesh K. Muppala is with the Dept. of Computer Science and Engineering, The Hong Kong University of Science and Technology, Clear Water Bay, Kowloon, Hong Kong (corresponding author phone: +852 2358 6978; fax: +852 2358 1477; e-mail: muppala@cse.ust.hk).

Malathi Veeraraghavan is with the Charles L. Brown Department of Electrical and Computer Engineering, University of Virginia, Charlottesville, VA 22904, USA.

data centers and data center networks in Section II. Thereafter, we discuss the details of data center network topologies that are implemented using standard Ethernet switches in Section III. We review the different routing protocols in Section IV. We present a review of techniques to improve performance in Section VII. We then review fault-tolerant routing techniques implemented in different architectures to deal with link and switch failures in Section VIII. Finally we conclude the paper in Section IX.

Each of these sections is related to one or more layers of the OSI model [14]. We use a table I to show this relationship.

II. Background and Motivation

Two major network fabrics have been considered for implementing data center networks: Ethernet and InfiniBand (IBA) [15]. Other high-speed network fabrics such as Myrinet [16] may also be considered. In this survey, we primarily focus on data center networks that are based on Ethernet technologies. As such, InfiniBand and Myrinet is outside the scope of this survey. Interested readers may find more information about these technologies and their use in data centres in [15].

This section motivates the recent research interest in data center networking. It first explores the reason why off-the-shelf switching/routing technologies based on Ethernet and IP that are so widely used in campus, regional and wide-area networks are deemed insufficient for data center networks. Next, it lists a set of problems that form the focus of recent research proposals for data center networks.

Ethernet is commonly used in data center networks. The nodes can be configured to operate in Ethernet-switched mode or IP-routed mode. Pros and cons of these two modes of operation are discussed below. Scalability is especially a concern since data center networks increasingly connect 100K - 1 million servers.

The Ethernet 6-byte MAC addressing is flat, i.e., an interface can be assigned any address (typically by the manufacturer) without consideration of its topological location (i.e., the switch/router to which it is connected). On the other hand, IP addressing is hierarchical, which means the address assigned to an interface depends on its topological location. The advantage of hierarchical addressing is that addresses can be aggregated for use in forwarding tables, which makes the size of forwarding tables smaller than in networks with flat addressing. This makes IP-routed networks more scalable than Ethernet-switched networks, which is why the Internet in which multiple networks, each owned and operated by a different organization, uses IP routing. However, the disadvantage of hierarchical routing is that if a virtual machine (VM) is migrated to a different host, one of three approaches is required: (i) its IP address needs to change to reflect its new topological position, which means loss of live TCP connections as TCP, unlike SCTP [17], does not support dynamic address configuration, (ii) forwarding tables in all intermediate routers are updated with the /32 IPv4 (and /128 IPv6) address of the migrating machine (an inherently unscalable solution), or (iii) a solution such as mobile IP [18] is required with home agents registering foreign care-of addresses for the migrating machine and tunneling packets to the migrating machine at its new location. In contrast, in an Ethernet-switched network, the migrating machine can retain its IP address; what needs to change is the IP address to MAC address resolution stored at the senders. One solution to this problem noted in an IBM VM migration document [19] is as follows: “Once traffic from the Virtual Machine on the new node is sent again the ARP tables will get updated and communication will continue as expected.”

Networks with flat addressing, e.g., Ethernet-switched networks, require no address configuration. Server interfaces come ready for plug-n-play deployment with manufacturer-configured addresses. With any hierarchical addressing scheme, address configuration is required since addresses have to be assigned to server interfaces based on their topological locations. While this step can be automated, it is still a source of potential misconfiguration errors.

On the other hand, routing protocols, used to disseminate reachability information, are less feasible in networks with flat addressing. Thus, for example, Ethernet-switched networks have no explicit routing protocols to spread reachability information about the flat addresses of servers. Instead flooding and address learning are the methods used to slowly create forwarding table entries. Flooding brings in the specter of routing loops, and hence the need for the spanning tree protocol. Spanning tree protocol avoids loops by disabling certain links, but this is wasteful of network bandwidth resources. To improve latency and increase throughput, IP routing protocols such as OSPF are often augmented with Equal Cost MultiPath (ECMP) techniques [20] in which packet headers are hashed before one of multiple paths is selected to avoid out-of-sequence packets within individual flows. Therefore, IP-routed networks have the distinct advantage of more efficient routing when compared to Ethernet-switched networks. The solution of using Virtual LANs with multiple spanning trees has been standardized in IEEE 802.1Q [21] to address some of these issues. Drawbacks of this VLAN approach as noted by Kim et al. [22] include configuration overhead, limited control-plane scalability, and insufficient data-plane efficiency.

Given the above considerations, it appears that neither a completely flat addressing network (e.g., Ethernet-switched network) nor a completely hierarchical addressing network (e.g., IP-routed network) appears suitable for a highly scalable, agile network that supports host mobility. Interestingly, these features are required in both data center networks and in the wide-area Internet. Solutions, which are hybrid approaches combining both flat and hierarchical addressing schemes, proposed for both arenas are remarkably similar. In the wide-area, Locator-Identifier Split Protocol (LISP) [23] is a representative solution of using hierarchical addresses as locators in the core, with identifiers being flat addresses in the edges. Equivalent approaches in the data center network arena are IETF's TRansparent Interconnection of Lots of Links (TRILL) [24] and the Provider Backbone Bridging approach in IEEE's Shortest Path Bridging solution [21]. Tunneling is used with the original packets encapsulated in new packets whose outer headers carry the addresses used for packet forwarding through the core (e.g., IP-in-IP, TRILL or MAC-in-MAC). One can visualize this solution as a network-in-network, with the address space

TABLE I
LAYERS AND SECTIONS

Section No.	Topic	Layers				
		Physical	Data Link	Network	Transport	Application
III	Architectures	✓	✓			
IV	Routing Techniques		✓	✓		
V	Traffic Engineering				✓	
VII	Performance Enhancement				✓	
VIII	Fault-tolerant Routing		✓	✓		✓

used in the inner core network being different from that used in the outer network. Since both the core and the outer networks are connectionless packet-switched networks, the network-layer header[1] is repeated, one for forwarding within the core, and one for forwarding at the edges. A second set of solutions extend tunneling all the way to the servers. This approach is proposed in shim6 [25] for wide-area networks, and VL2 [4] for data center networks.

Whether or not these tunneling approaches are sufficient to meet the requirements of data center networks is not explored by most research publications. Instead researchers cite four main concerns for justifying work on new architectures, starting with the above-described issue of scalability: The main research concerns in data center network design include:

- How to maintain scalability so that data centers can meet future demands?
- How to achieve maximum throughput while minimizing the cost?
- How to guarantee data integrity and system availability when multiple failures occur in the system?
- How to enhance power efficiency so as to reduce operational cost and make data centers environmentally friendly?

In this paper, we choose to primarily concentrate on a representative subset of data center network architectures that have appeared in the literature. These include, Fixed topology networks based on Fat-tree based architectures as described in Al-Fares et al. [3], PortLand [26] and Hedera [27]; Clos network based architecture represented by VL2 [4]; recursive topologies such as DCell [5] and BCube [6] and Flexible architectures such as cThrough[7], Helios [8] and OSA [9]. Other noteworthy architectures that we do not explicitly cover in this survey, but are worth noting include FiConn [28], MDCube [29], and CamCube [30]. The next four sections describe these set of data center network architectures proposed by academic and industrial research laboratories. In most of these architectures, the starting point is a specific topology (Section III). By tying the addressing scheme to the topology, a routing protocol is no longer needed. Instead each node can run a predetermined algorithm to determine the output link on which to forward a packet based on its destination address (Section IV). Multipath routing and other techniques are often included to improve performance (Section VII). Also, as the regular structure of the topology will break down when failures occur, automated procedures are required to update forwarding tables (Section VIII).

III. DATA CENTER NETWORK TOPOLOGIES

Considering the cost and availability, high-speed Ethernet (1Gbps, 10Gbps) is most commonly used in constructing internal network of data centers [15]. However, although most data centers use Ethernet switches to interconnect the servers, there are still many different ways to implement the interconnections, leading to different data center network topologies. Each of these different topologies is characterized by different resource requirements, aiming to bring enhancements to the performance of data centers. In the following sections some representative topologies used in data centers will be discussed. If servers and switches are regarded as vertices, wires as edges, the topology of every data center network can be represented as a graph.

All the topologies discussed in this section can be classified into two categories: *fixed* and *flexible*. If the network topology cannot be modified after the network is deployed, we classify it as a fixed architecture; otherwise it is a flexible architecture. Sections III-A and III-B are about topologies for fixed architectures, and section III-C is about flexible architectures. Figure 1 gives a taxonomy of the different data center network topologies.

Furthermore, here we summarise the notations used in this section:

- n: The number of ports in a switch in an architecture.
- k: The number of ports in a server in an architecture.
- N: The total number of servers inside a data center network.

It should be noted that n and k may vary according to the position of the node.

A. Fixed Architectures I: Tree-based Topologies

Standard tree based architectures and their variants are widely used in designing data center networks [3], [26], [27].

1) Basic Tree: Basic tree topologies consist of either two or three levels of switches/routers, with the servers as leaves. In a 3-level topology, there is a *core* tier at the root of the tree, an *aggregation* tier in the middle, and an *edge* tier of switches connecting to the servers. In a 2-level topology, there is no *aggregation* tier. There are no links between switches in the same tier, or in nonadjacent tiers. Figure 2 shows a 3-level tree topology.

In a basic tree topology, the higher-tier switches need to support data communication among a large number of servers. Thus switches with higher performance and reliability are required in these tiers. The number of servers in a tree architecture is limited by the numbers of ports on the switches.

[1] Ethernet is considered Layer 2, though switching is defined in the OSI protocol reference model as a network-layer function.

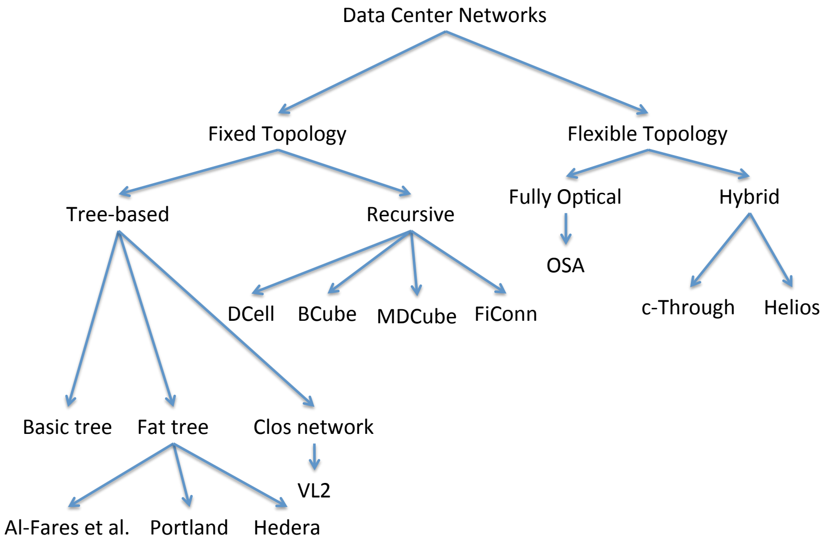

Fig. 1. A Taxonomy of Data Center Topologies

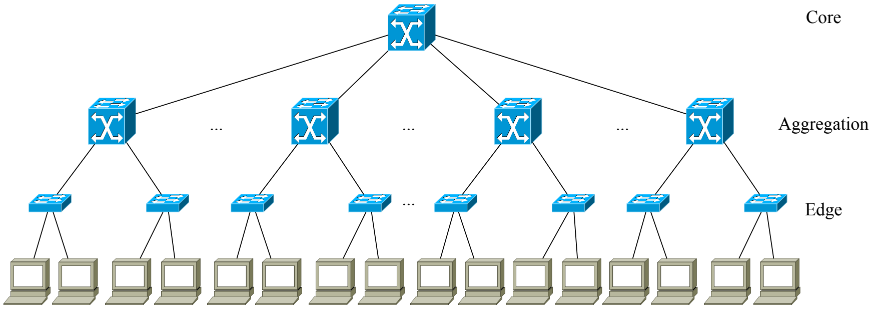

Fig. 2. A 3-level Tree Topology

2) Fat Tree: Fat tree is an extended version of a tree topology [31]. Fat trees have been applied as a topology for data centers by several researchers [3], [26].

A fat tree is a network based on a complete binary tree. Each n-port switch in the edge tier is connected to $\frac{n}{2}$ servers. The remaining $\frac{n}{2}$ ports are connected to $\frac{n}{2}$ switches in the aggregation level. The $\frac{n}{2}$ aggregation-level switches, the $\frac{n}{2}$ edge-level switches and the servers connected to the edge switches form a basic cell of a fat tree, which is called a *pod*. In the core level, there are $\left(\frac{n}{2}\right)^2$ n-port switches, each one connecting to each of the n pods. Figure 3 shows a fat tree topology with $n = 4$. Unlike in basic tree topology, all the 3 levels use the same kind of switches. High-performance switches are not necessary in the aggregate and core levels. The maximum number of servers in a fat tree with n-port switches is $\frac{n^3}{4}$ [3].

3) Clos Network: Greenberg et al. proposed a data center network architecture using a Clos Network topology in [4]. A Clos Network is also a multi-rooted tree. When deployed in data center networks, a Clos Network usually consists of three levels of switches: the ToR switches directly connected to servers, the aggregation switches connected to the ToR switches, and the intermediate switches connected to the aggregation switches. These three levels of switches are termed “input”, “middle”, and “output” siwtches in the original Clos terminology [32]. 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 n ports, there will be n aggregation switches and $\frac{n}{2}$ intermediate switches. There is exactly one link between each intermediate switch and each aggregation switch. The remaining $\frac{n}{2}$ ports on each aggregation switch is connected to $\frac{n}{2}$ different ToR switches. Each of the ToR switches is connected to 2 different aggregation switches, and the remaining ports on the ToR

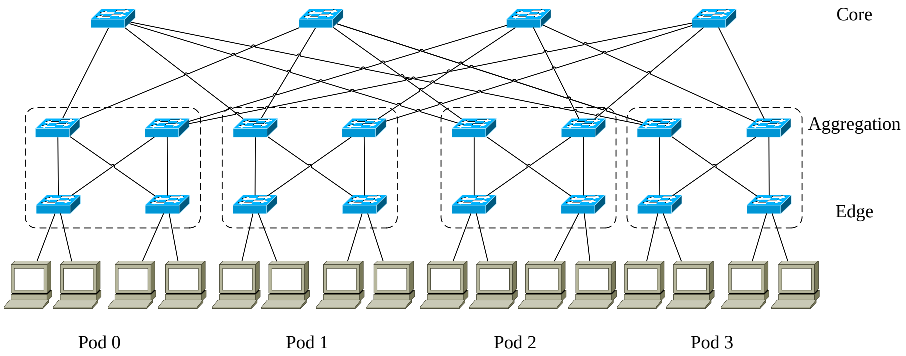

Fig. 3. A 3-level Fat Tree Topology

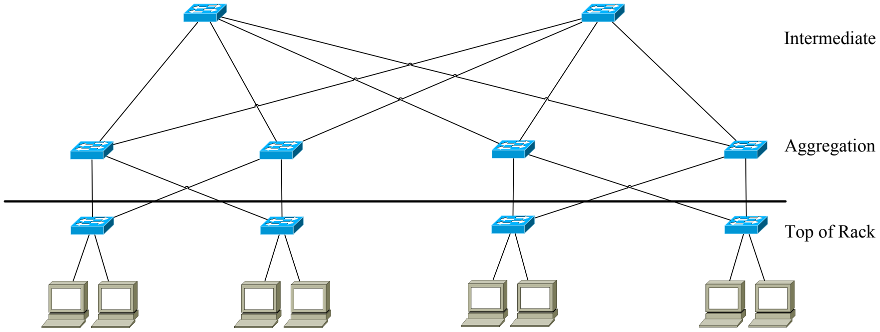

Fig. 4. A Clos Network Topology

switches are connected to servers. There are $\frac{n^2}{4}$ ToR switches because to each pair of aggregation switches, $\frac{n}{2}$ ToR switches are connected. 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 n_{ToR} ports on each TOR switch are connected to servers, there will be $\frac{n^2}{4} \times n_{ToR}$ servers in the network. Figure 4 shows a Clos Network with $n = 4$, $n_{ToR} = 2$.

B. Fixed Architectures II: Recursive Topologies

Tree-based topologies can scale up by inserting more levels of switches, while each server is connected to only one of the bottom level switches. The recursive topologies still have levels/tiers as in the tree-based topologies. However, recursive topologies use lower level structures as *cells* to build higher level structures, and the servers in recursive topologies may be connected to switches of different levels or even other servers. There are multiple network ports on the servers of recursive topologies, making them significantly different from the tree-based topologies. Graphs, rather than multi-rooted trees, are suitable to depict recursive architectures. Some representative recursive topologies will be discussed in the following sections.

1) DCell: DCell [5] is the first type of recursive data center network topology discussed in this paper. The most basic element of a DCell, which is called DCell$_0$, consists of n servers and one n-port switch. Each server in a DCell$_0$ is connected to the switch in the same DCell$_0$.

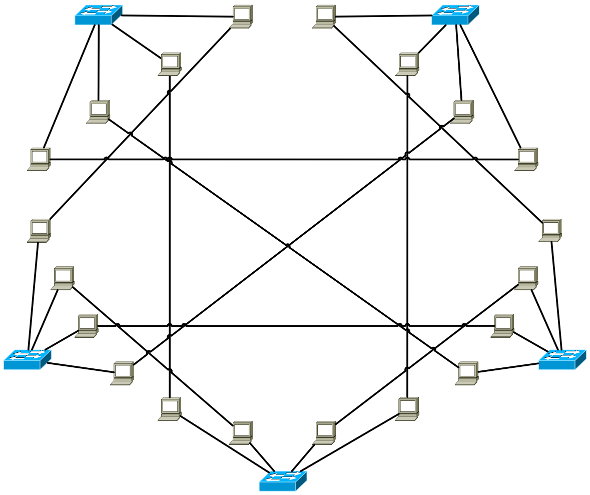

Fig. 5. Constructing a DCell$_1$ from DCell$_0$s with $n = 4$

Let DCell$_k$ be a level-k DCell. The first step is to construct a DCell$_1$ from several DCell$_0$s. Each DCell$_1$ has $n+1$ DCell$_0$s, and each server of every DCell$_0$ in a DCell$_1$ is connected to a server in another DCell$_0$, respectively. As a result, the DCell$_0$s are connected to each other, with exactly one link between every pair of DCell$_0$s. A similar procedure is used to construct a DCell$_k$ from several DCell$_{k-1}$s. In a DCell$_k$, each server will eventually 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$_{i-1}$. Assume that each DCell$_{k-1}$ has t_{k-1} servers, then a DCell$_k$ will consist of t_k DCell$_{k-1}$s, and consequently $t_{k-1} \times t_k$ servers. Obviously, we have $t_k = t_{k-1} \times (t_{k-1} + 1)$. Figure 5 shows a DCell$_1$ when $n = 4$. It can be seen that the number of servers in a DCell grows double-exponentially, and the total number of levels in a DCell is limited by the number of NICs on the servers in it. DCell can scale to very large number of servers using switches and servers with very few ports. For example, when $n = 6$, $k = 3$, a fully constructed DCell can comprise more than 3 million servers [5].

2) BCube: BCube [6] is a recursive topology specially designed for shipping container based modular data centers. Building a data center cluster in a 20- or 40-foot shipping container makes it highly portable. As demands change at different data centers, the whole cluster can be readily moved. While the deployment time is considerably shorter, the disadvantage of this environment is that due to operational and space constraints, once deployed in the field, it is difficult to service the cluster.

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 constructing higher level architecture. While constructing a BCube$_1$, n extra switches are used, connecting to exactly one server in each BCube$_0$. Therefore a BCube$_1$ contains n BCube$_0$s and n extra switches(If the switches in the BCube$_0$s are taken into consideration, there are totally $2n$ switches in a BCube$_1$). More generally, a BCube$_k$ is constructed from n BCube$_{k-1}$s and n^k extra n-port switches. These extra switches are connected to exactly one server in each BCube$_{k-1}$. In a level-k BCube, each level requires n^k switches (each of which is an n-port switch).

BCube makes use of more switches when constructing higher level structures, while DCell uses only level-0 n-port switches. Both however 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 shows a BCube$_1$ with $n = 4$.

Just like the 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].

C. Flexible Architectures

Recently, more and more researchers ([7], [8], [9]) have considered using optical switching technology to construct DCNs. Besides offering high bandwidth (up to Tbps per fiber with WDM techniques), optical networks have significant flexibility of reconfiguring the topology during operation. Such feature is important considering the unbalanced and ever-changing traffic patterns in DCNs. In this section, we introduce three of the flexible architectures: c-Through [7], Helios [8] and OSA [9].

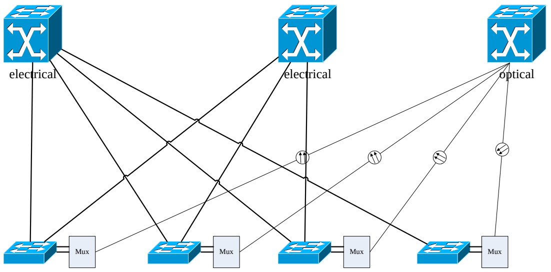

Fig. 8. Helios Network

1) c-Through: c-Through [7] is a hybrid network architecture which makes use of both electrical packet switching network and optical circuit switching network. The hybrid network of c-Through, or the so-called HyPaC (Hybrid Packet and Circuit) Network, consists of two parts: a tree-based electrical network which maintains connectivity between each pair of ToR switches, and a reconfigurable optical network which offers high bandwidth interconnection between certain racks. 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 racks; instead c-Through connects each rack to exactly one other rack at a time. As a result, the high capacity optical links are offered to pairs of racks transiently according to the traffic demand. The estimation of traffic between racks and reconfiguration of the optical network is accomplished by the control plane of the system. Figure 7 shows a c-Through network (servers which are directly connected to ToR switches are not shown).

To configure the optical part of c-Through network, the traffic between racks should be known. c-Through estimates the rack-to-rack traffic demands by observing the occupancy of socket buffer. Since only one optical link is offered to each rack, the topology should be configured such that the most amount of estimated traffic can be satisfied. In [7], the problem is solved using the maximum-weight perfect matching algorithm (Edmonds' Algorithm) [33]. The topology of the optical network is configured accordingly.

2) Helios: Helios [8] is another hybrid network with both electrical and optical switches. Helios is a 2-level multi-rooted tree of core and pod (ToR) switches. Core switches can be either electrical or optical, so as 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 a optical transceiver. Half of the uplinks are connected to the electrical switches, while the other half are connected to the optical switch through a optical multiplexer. The multiplexer combines the links connected to it to be a "superlink" and enables flexible bandwidth assignment on this superlink. Figure 8 shows a Helios network topology (the transceivers are not shown).

Helios uses an algorithm from Hedera [27] to estimate

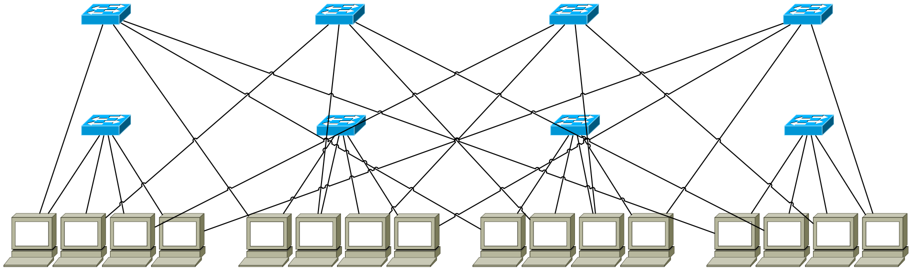

Fig. 6. Constructing a BCube_1 from BCube_0s with $n = 4$

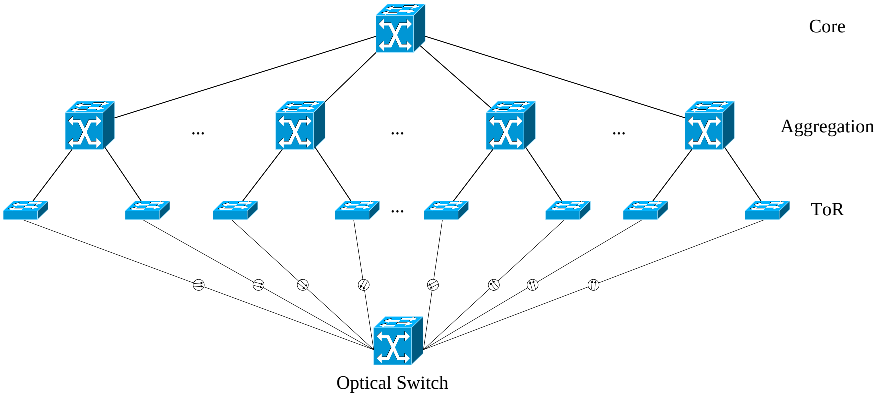

Fig. 7. c-Through Network

traffic demand between racks, which allocates fair bandwidth for TCP flows. The goal of the configuration of optical network is the same as c-Through: to maximize the traffic demand that can be satisfied. Helios also formulates the optimization problem as maximum-weight matching and solves it using Edmonds' Algorithm.

3) OSA: OSA [9] is 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 (the typical amount of such links is 4). However, the connection pattern is determined flexibly according to the traffic demand. Since the network does not ensure direct optical link between each pair of rack 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 maximum-weight matching problem. It is a multi-commodity flow problem with degree constraints, which is NP-hard. In [9], the problem is simplified as a maximum-weight b-matching problem [34], and approximately solved by applying multiple maximum-weight matchings.

D. Comparisons Of Topologies

1) Parameter Comparison: Table II presents a comparison of some of the parameters of the topologies introduced earlier. The flexible architectures are not shown because they do not have a fixed topology. The following parameters are used when comparing the different topologies:

- *Degree of the servers*: The (average) number of network ports on the servers in the data center. For the tree-based topologies, only one port is needed on each server. However, in the recursive topologies, the number of ports vary according to the levels required. Considering the data shown in table III, k does not need to be large to scale up. On the other hand, since the servers are not specifically designed for switching, and the actual bandwidth that a server can offer is also limited by the speed of storage, it will be meaningless to add more and more ports on servers.
- *Diameter*: The longest of the shortest paths between two servers in a data center. A smaller diameter leads to more effective routing, and lower transmission latency in practice. For most topologies the diameter grows logarithmically with the number of servers.
- *Number of Switches*: It is assumed that all switches are the same in each topology except Clos Network. This assumption also stands for tables III, IV and V. BCube uses the largest amount of switches, which may lead to higher cost. The basic tree architecture uses the fewest switches because there is little hardware redundancy in the architecture.
- *Number of Wires*: This metric shows the number of wires required when deploying a data center. Similar to the

TABLE II
SUMMARY OF PARAMETERS

	Tree-based Architecture			Recursive Architecture	
	Basic Tree	Fat Tree [3]	Clos Network [4]	DCell [5]	BCube [6]
Degree of Servers	1	1	1	$k+1$	$k+1$
Diameter	$2\log_{n-1}N$	6	6	$2^{k+1}-1$	$\log_n N$
No. of Switches	$\frac{n^2+n+1}{n^3}N$	$\frac{5N}{n}$	$\frac{3}{2}n+\frac{n^2}{4}$	$\frac{N}{n}$	$\frac{N}{n}\log_n N$
No. of Wires	$\frac{n}{n-1}(N-1)$	$N\log_{\frac{n}{2}}\frac{N}{2}$	$N+\frac{4N}{n_{ToR}}$	$\left(\frac{k}{2}+1\right)N$	$N\log_n N$
No. of Servers	$(n-1)^3$	$\frac{n^3}{4}$	$\frac{n^2}{4}\times n_{ToR}$	$\geq\left(n+\frac{1}{2}\right)^{2^k}-\frac{1}{2}$, $\leq(n+1)^{2^k}-1$	n^{k+1}

[i] Typically k is smaller for DCell because it needs smaller k to connect the same number of servers compared to other recursive architectures.

condition of the number of switches, BCube uses the most wires, and then DCell. However, it should be noted that the number of wires only shows the wiring complexity; it does not show the accurate wiring cost, because not all the architectures use homogeneous wires. E.g., the typical bandwidth of a link between an aggregation switch and a core switch in fat-tree is 10 Gbps, while it is 1Gbps on a link between a server and an edge switch.

- *Number of Servers*: All the metrics above are counted under the same number of servers (N), while this one shows the scalability of different architectures with the same n and k (for recursive architectures only). It is assumed in this row that the tree-based architectures use 3-level structure. Considering the data in table III, it is no doubt that DCell scales up much faster than other architectures for the number of servers in DCell grows double-exponentially with k.

2) Performance Comparison: Table IV shows the comparisons of some performance metrics of different topologies.

- *Bandwidth*: The first four rows of table IV show the bandwidth that the topologies can offer under different traffic patterns. "One-to-one" means the maximum bandwidth that the topology can offer when one arbitrary server sends data to another arbitrary server, and so on so forth. "All-to-all" bandwidth means every server establishes a flow to all the other servers. Each kind of these traffic pattern is meaningful for different applications. E.g., one-to-several traffic occurs when the file system is making replicas, one-to-all traffic occurs when updating some software on all the servers, and all-to-all traffic is very common in MapReduce.

  Here in table IV the bandwidths are expressed as the number of links, which implies that each link in a data center has the same bandwidth. It is shown in the table that one-to-one, one-to-several and one-to-all bandwidths are in fact limited by the number of ports on the servers, or the degree of servers. The basic tree topology offers the smallest all-to-all bandwidth, limited by the number of ports of the switch at the root. Since fat tree introduces redundant switches, it can offer as many edge-disjoint paths as the number of servers.
- *Bisection Width*: The minimum number of wires that must be removed when dividing the network into two equal sets of nodes. The larger it is, the better fault tolerance ability a topology will have. A similar metric, the bisection bandwidth is also widely used, which means the minimum collective bandwidth of all the wires that must be removed to divide the network into two equal sets of nodes. Basic tree has the smallest bisection width, which means that it is the most vulnerable of all.

Another important property, the relationship between all-to-all bandwidth and the ratio of faulty components is not shown in the table because of lack of testing data for all the topologies. A graceful degradation of performance implies that when more and more components fail in the data center, performance reduces slowly without dramatic falls. When building a new data center, it is usual that a partial topology will be built first, and more components are added according to future need. This partial network can be viewed as a network with a lot of failures. Building a partial network with a topology of graceful degradation means that people can get expected performance with fewer components. [6] gives a comparison between fat tree, DCell and BCube on this metric with simulation results.

3) Hardware Redundancy of Data Center Network Topologies: The hardware redundancies discussed here are based on the topology of data center network. Some data center network architectures, such as PortLand and VL2, which requires more reliability for their centralized management system, are not considered here.

Table V shows the hardware redundancy offered by different topologies. Before discussing the table, some definitions need to be clarified:

Definition 3.1 (Node-disjoint Paths): The minimum of the total number of paths that share no common intermediate nodes between any arbitrary servers.

Definition 3.2 (Edge-disjoint Paths): The minimum of the total number of paths that share no common edges between any arbitrary servers.

Definition 3.3 (f-fault tolerance): [35] A network is f-fault tolerant if for any f failed components in the network, the network is still connected.

Definition 3.4 (Redundancy Level): [35] A network has a redundancy level equal to r if and only if, after removing any set of r links, it remains connected, and there exists a set of $r+1$ links such that after removing them, the network is no longer connected.

In the table, the redundancy levels of different components are shown respectively, which reflects the influence of different

TABLE III
NUMBER OF SERVERS

n	Tree-based Architecture			k	Recursive Architecture	
	Basic Tree	Fat Tree [3]	Clos Network [4]		DCell [5]	BCube [6]
4	64	16	8	2	420	64
				3	$176,820$	256
				4	$> 3 \times 10^{10}$	$1,024$
6	216	54	36	2	$1,806$	216
				3	$3,263,442$	$1,296$
				4	$> 10^{13}$	$7,776$
8	512	128	96	2	$5,256$	512
				3	$27,630,792$	$4,096$
				4	$> 7 \times 10^{14}$	$32,768$
16	$4,096$	$1,024$	896	2	$74,256$	$4,096$
				3	$> 5 \times 10^{9}$	$65,536$
48	$110,592$	$27,648$	$26,496$	2	$5,534,256$	$110,592$

TABLE IV
PERFORMANCE SUMMARY

	Tree-based Architecture			Recursive Architecture	
	Basic Tree	Fat Tree [3]	Clos Network [4]	DCell [5]	BCube [6]
One-to-one	1	1	1	$k+1$	$k+1$
One-to-several	1	1	1	$k+1$	$k+1$
One-to-all	1	1	1	$k+1$	$k+1$
All-to-all	n	N	$\frac{2N}{n_{ToR}}$	$> \frac{N}{2^k}$	$\frac{n}{n-1}(N-1)$
Bisection Width	$\frac{n}{2}$	$\frac{N}{2}$	$\frac{N}{n_{ToR}}$	$> \frac{N}{4 \log_n N}$	$\frac{N}{2}$

TABLE V
HARDWARE REDUNDANCY SUMMARY

			Tree-based Architecture			Recursive Architecture	
			Basic Tree	Fat Tree	Clos	DCell	BCube
Node-disjoint Paths			1	1	1	$k+1$	$k+1$
Edge-disjoint Paths			1	1	1	$k+1$	$k+1$
Redundancy Level	Switch	Edge/ToR	0	0	0	k	k
		Aggregation	0	$\frac{n}{2}-1$	1		
		Core/Intermediate	0	$\frac{n^2}{4}-1$	$\frac{n}{2}-1$		
	Server		-	-	-	k	k

[i] n_{core} is the number of core switches in the tree.

failures.

The numbers of node-disjoint paths and edge-disjoint paths are the same in each of the topology. This is because there is exactly one link between each pair of directly connected nodes (server to server, server to switch, switch to switch) in these topologies. If redundant links are added making the topology a multigraph, there will be more edge-disjoint paths than node-disjoint paths. The number of node-disjoint paths shows the ability of a system to deal with node failures, while the number of edge-disjoint paths shows the ability of dealing with link failures. A node failure can have a more severe impact than a link failure, because it usually leads to several link failures.

The three tree-based topologies have fewer number of node-disjoint paths and edge-disjoint paths compared to DCell and BCube. The main reason for this is that tree-based topologies and FiConn place fewer ports on the servers. Each server is connected to an edge switch with single connection in tree-based topologies. Fewer ports on servers makes the connections between servers and switches vulnerable to failures. For example, in the tree-based topologies, if one edge switch fails, every server connected to it will be separated from the remaining of the data center. This also explains why there is no redundancy of edge switches.

DCell and BCube, however, have more node-disjoint paths and edge-disjoint paths because of the multiple ports on their servers, which results in higher redundancy level. Unless each of the node-disjoint paths has a failure on it, two servers will always remain connected. It should be noted that the redundancy level of DCell or BCube is decided by the number of ports on servers.

This table does not show the difference between the failure rate of different components. For example, in the basic tree topology, designers tend to use more expensive and reliable switches as the core switches, which have a much lower failure rate compared to the cheaper edge switches.

E. Potential New Topologies

Most of the topologies discussed above have their prototypes in the area of supercomputing or parallel computing. e.g., fat tree [31] was first proposed for supercomputing, and BCube is closely related to Hypercube [36], which was designed for parallel computing. This is because supercomputing and parallel computing share many design goals in common with data center networks, such as large amount of computing nodes, requirement of high bandwidth, and large amount of data transmission.

The taxonomy that we propose is not the only approach for classifying the data center network topologies. Zhang et al. [38] proposes a taxonomy by classifying data center networks into two categories, *hierarchical* topologies and *flat* topologies, which correspond to the tree-based topologies and recursive topologies in our survey respectively. Lucian Popa et al. classify the architectures into *switch-only* and *server-only* topologies in [39].

IV. Addressing, Forwarding and Routing in Data Center Network Architectures

Given a network topology, routing techniques and protocols define how data packets are delivered from a source to a destination within a data center. The *orchestration* of the network in order to achieve the goals of data delivery among the servers is what we discuss next in this section. *Addresses* are used to uniquely identify interfaces (or nodes) within a data center network. *Forwarding* is the mechanism by which packets are forwarded from one link to another at switches/routers. Typically this action is based on destination address carried in a packet header. The *data-plane protocol* used for packet forwarding specifies the addressing format used. Also of interest is the *node at which forwarding actions* take place. In most cases, these actions occur at switches/routers, but in several of the architectures compared here, servers perform packet forwarding between the several NICs with which they are equipped. Such forwarding in servers can be implemented in software or special hardware. Finally, the *routing mechanisms* used to determine the next-hop node, whether executed on a packet-by-packet basis, or used in conjunction with routing protocols for a pre-configuration of forwarding tables, are presented for each architecture. The focus of this section is on unicast, single-path routing, while multi-path routing will be discussed in section VII. Now we present the details of the addressing, forwarding and routing mechanisms used in the different network architectures described in section III.

A. Fat-Tree Architecture of Al-Fares et al. [3]

At first glance, it appears that this architecture proposes a direct use of IP routers in a fat-tree topology, and specifies a particular address assignment scheme that leverages the regular topology eliminating the need for a routing protocol such as OSPF. In other words, a simple algorithm can be executed to determine the output port based on the destination IP address given that the algorithm leverages the structured manner in which addresses are assigned.

However, on closer examination, one sees that the nodes at which packet forwarding occurs cannot be off-the-shelf IP routers performing standard longest-prefix matching operations. Instead a new router design (Al-Fares et al. [3] refer to the nodes as *switches* even though packet forwarding action is at the IP layer. The term *router* is hence more appropriate) is required for a two-level lookup. Therefore, even though the data-plane protocol is unchanged, i.e., servers generate IP datagrams carried in Ethernet frames, new routers need to be designed, implemented, and deployed to realize this architecture.

As for the routing mechanism, a centralized controller is envisioned to create the two-level forwarding tables at all the routers in the fat tree so that the forwarding action is reduced to a two-level table lookup rather than execution of the algorithm on a packet-by-packet basis. Mention is made of needing distributed routing protocols to handle failures, but no details are provided.

1) Addressing: Al-Fares et al. [3] make use of the dotted decimal notation of IP addresses to specify the position of a node(either a server or a switch) in the tree. As described in III-A2, a fat tree has n pods, with n switches in each pod, while $\dfrac{n}{2}$ servers are connected to each of the $\dfrac{n}{2}$ edge switches in a pod. The reserved private $10.0.0.0/8$ block is allocated to the nodes. The pod switches, including edge switches and aggregation switches are addressed in the form $10.pod.switch.1$, where $pod \in [0, n-1]$ denotes the number of the pod that a switch belongs to, and $switch \in [0, n-1]$ denotes the position of a switch in a pod. The $\dfrac{n^2}{4}$ core switches are addressed as $10.k.j.i$, where $i, j \in \left[1, \frac{n}{2}\right]$.

The $\dfrac{n}{2}$ servers connected to edge switch $10.pod.switch.1$ is addressed as $10.pod.switch.ID$, where $ID \in \left[2, \frac{n}{2}+1\right]$ denotes the position of the server in the subnet connected to the switch. With such an addressing scheme, n can be as large as 256, supporting as many as 4.19M servers.

2) Forwarding and Routing: In the fat tree architecture shown in section 3, the core switches need to transmit all the inter-pod traffic. If hop count is used as the metric of evaluating a route, there will be $\left(\dfrac{n}{2}\right)^2$ shortest routes between two servers in different pods, each route with 6 hops: 3 hops from the source up to a core switch, and 3 hops from the core switch down to the destination. The main goal of routing in fat-tree is to distribute the traffic evenly among these core switches.

Al-Fares et al. [3] propose a 2-level routing table to solve this problem. Unlike the common routing table in which one entry relates several IP address with one port on the switch, every entry in this 2-level routing table may potentially have a pointer to a small secondary table. The entries of the secondary table are in the form of $(suffix, port)$, while the entries in the primary table are like $(prefix, port)$. Not all the entries of the primary table have a pointer to a secondary entry, while one secondary entry may be pointed to by multiple primary entries. A primary entry is *terminating* if it points to no secondary entry. When routing a packet, the destination IP address will be first checked to match the longest prefix in the primary table. If the result primary entry points to a secondary table, the destination IP will be checked again to match the longest suffix, and the output port will be finally decided.

In such a two-level routing scheme, the pod switches have terminating prefixes for the servers in that pod, and a default /0 prefix with a secondary table for addresses outside the pod. As a result, in a pod switch, the entries in the primary table will distinguish destination servers that do not belong to the same pod, and the entries in a secondary table directs packets to different paths so as to get a balanced flow distribution, e.g., the aggregation switches will transfer packets with different suffixes through different core switches so that each core switch will receive the same number of packets in all-to-all traffic. For the core switches, the routing tables are simpler. They are assigned terminating prefixes for all network addresses, so that they can correctly direct every packet to the pod where the destination server is. Because one core switch has exactly one link to an aggregation switch in each pod, and one aggregation switch has exactly one link to every edge switch in its pod, when a packet arrives at a core switch, the remaining of its shortest path to the destination is unique. Traffic diffusion only happens at the first half of a path, i.e., before a packet reaches a core switch.

It should be noted that in such a routing scheme, the original routing table is generated according to the architecture, not by any autonomous route discovering mechanism.

B. PortLand and Hedera: Layer-2 Routing Fabric for Tree-based Topologies

PortLand [26] offers layer-2 addressing, forwarding and routing in multi-rooted tree based data centers. Hedera [27] is based on PortLand, and its implementation augments PortLand routing and fault tolerant protocols. Unlike the routing schemes we have discussed, PortLand/Hedera use a centralized solution to manage the whole fabric.

1) Addressing in PortLand: PortLand uses a 48-bit hierarchical Pseudo MAC (PMAC) addressing scheme to realize forwarding, routing and VM migration. A unique PMAC is assigned to each machine (physical or virtual), encoding the location of it. For example, in the fat tree architecture, all the computers inside the same pod will have the same prefix in the PMAC addresses. The machine itself, however, remains unmodified. It still keeps its actual MAC (AMAC) address. The ARP requests received by the fabric manager will return the PMAC of an IP address. The address in a packet header will remain PMAC until it reaches the edge switch of the machine, where the mapping between the destination's PMAC and AMAC are stored.

2) Fabric Manager: The fabric manager of PortLand is a user process running on a dedicated computer inside the data center. It is responsible for assisting ARP resolution, fault tolerance, multicast and other functions. Using a centralized structure can help to reduce the complexity of routing, while reducing robustness at the same time. To keep robustness, the amount of centralized information kept at the fabric manager is limited, and there is no need of administrator configuration for the manager. Besides, one or more replicas of the fabric manager are kept with asynchronous update.

3) Proxy-based ARP: Address Resolution Protocol (ARP) offers the mapping from IP addresses to MAC addresses in Ethernet. However, in PortLand, since PMAC is used, standard ARP does not work. In PortLand, the ARP requests sent by a computer are intercepted by the edge switch connecting it, and the edge switch will forward the requests to the fabric manager. The fabric manager will look up in its PMAC-IP table, and if there is a matched entry, it will return the result to the edge switch, which will finally create an ARP reply to the requesting computer.

In this scenario it is required that one machine is connected to only one switch or the so-called edge switch, which is not possible in recursive architectures where there are multiple ports on computers. In that case, an ARP request may reach multiple switches, while being handled by other computers during transmission. To make the PMAC and Proxy-based ARP mechanisms work, the computers in recursive architectures should be transparent to ARP requests. Besides, each ARP request should be labeled with the requesting computer's own address and a timestamp, to avoid multiple replies to the same request.

It is possible that the fabric manager does not have a matched entry for a request, for example, in the case of collapse recovery. When this happens, the fabric manager will have to launch a broadcast to retrieve the mapping. The machine owning the target IP will reply its AMAC, which will be translated to PMAC by the connecting switch.

PortLand offers an extra characteristic for more convenient VM migration. While the AMAC will remain the same no matter how a VM migrates, its PMAC and IP address will change according to its physical position. After migration is completed, a VM will send a ARP message to the fabric manager reporting its new IP. The fabric manager will then send an ARP invalidation message to the VM's previous switch. For those machines that are still attempting to communicate with the previous PMAC of the migrated VM, this previous switch will send a unicast ARP message to each of these machines to announce the new PMAC and IP of the migrated VM.

4) Location Discovery and Packet Forwarding: PortLand uses a Location Discovery Protocol (LDP) to discover the locations of the switches, so as to make packet forwarding and routing more effective. A PortLand switch will only forward data packets after its location is established. It will periodically send a Location Discovery Message (LDM) through all its ports. A LDM contains the ID of the switch and its location information. LDMs help switches to find their neighboring switches, and locate themselves by checking the number and state of switches they are connecting. Finally the fabric manager will assign a number corresponding to the location to each of the switches.

Once the location of the switches is established by means of LDP, they can use the location information to forward packets. For example, in the fat tree architecture, a core switch can learn pod number from aggregation switches connected to it. When a packet comes, it just need to inspect the bits corresponding to pod number in the PMAC of the packet, and then it can decide which port to use to send the packet.

LDP offers an automatic mechanism for the switches to discover the location by themselves, and thus eliminates the necessity of manual configuration. This is meaningful for initial construction of the data center, and collapse recovery.

5) Routing in Portland/Hedera: The routing scheme discussed in IV-A is based on layer-3 or network layer. However, layer-2 routing is also an alternative. Both approaches have advantages and disadvantages. The main reason for using layer-2 routing is the requirement to support Virtual Machines (VMs). Using VMs helps to make the hardware and network details transparent to the users, and can migrate the physical location of data or applications without affecting the users. However, since one computer may host multiple VMs, and a VM may migrate from one computer to another, using layer-3 routing, or using a fixed IP address to denote a VM is infeasible. Layer-2 routing is proposed to solve these problems.

C. SEATTLE

SEATTLE uses Ethernet addressing, and does not require a new protocol, i.e., Ethernet and IP are used. But it does require new switches because when an Ethernet frame arrives at a switch, it needs to perform a hashing function on the destination MAC address in the frame header to determine the identifier of the switch that stores the location of the destination, unless the location of the destination is cached. Either way, the frame needs to tunneled to the switch that stores the location or the switch to which the destination is connected. Further the switches need to run a link-state protocol. But host MAC addresses are not propagated via the link-state protocol, only switch-level topology information.

The solution is interesting in that it enjoys the advantages of flat addressing (e.g., plug-n-play operation with no address configuration required plus VM migration) without the problems caused by flooding, spanning tree algorithm, and broadcasts. The cost is that switches need to perform hashing functions and tunneling in case of cache misses, which means existing Ethernet switches cannot be used. Performance gains through a reduction of control overhead and state requirements are demonstrated through simulations when compared to Ethernet switching.

D. VL2

VL2 [4] inserts a layer-2.5 shim into the computers' typical network protocol stack to implement some features that are limited by network devices.

Addressing is IP based. The protocols are unchanged, i.e. Ethernet and IP are used. Switches and routers are used unchanged. The change required is the addition of a new layer between IP and MAC in end host's networking software stacks.

1) Addressing in VL2: Similar to PortLand, VL2 is designed to keep the flexibility of assigning IP addresses to services or VMs so that computers can be reassigned to services according to their requirements, without changing the actual network architecture. When a server needs to send a packet, the shim layer inserted into its protocol stack will inquire a directory system for the actual location of the destination, then rewrite the head of the packet and tunnel the packet to the shim layer of the destination.

To achieve the design goal above, VL2 makes use of two IP address families: the location-specific IP addresses (LAs) for network infrastructure, and the application-specific IP addresses (AAs) for applications. LAs are assigned to the switches and interfaces on computers, and are used to route packets. The AAs, however, are assigned to applications, and remain unchanged even if the physical location of the server changes or VMs migrate. Each AA is related to an LA. The VL2 directory system is implemented to store the mapping from AAs to LAs and answer the queries. The shim layer on each computer, or so-called VL2 agent, is responsible to replace AAs with LAs when sending packets, and LAs back to AAs when receiving packets.

2) VL2 Directory System: VL2 uses a centralized management system called the Directory System to offer necessary address mapping services required by the architecture. The system consists of two types of servers:

- A modest number (50 - 100 servers for 100K servers) of replicated *directory servers*, which cache address mappings and reply the queries;
- A small number (5 - 10 servers) of *replicated state machine (RSM)*, which offer a reliable storage of address mappings.

Each of the directory servers keeps a copy of the whole AA-LA mappings, and use its local information to reply to queries from servers. To keep consistency, a directory server periodically synchronizes its mappings with an RSM. When there is an update in the mapping, the update will be sent to a random directory server, which will forward the update to an RSM server. The RSM server will replicate the update to each of the RSMs, then replies to the directory server , which forwards the reply to the originating machine.

3) Packet Forwarding in VL2: Since the traffic between servers use AAs, while the underlying network only knows routes for LAs, the VL2 agent at each server has to trap the packets and encapsulate the packet with the LA address of the ToR switch of the destination. While the packet arrives at the LA, it will be decapsulated by the ToR switch and sent to the destination AA.

E. DCell

1) Addressing in DCell: Every server in a $DCell_k$ is identified with a $(k+1)$-tuple $[a_k, a_{k-1}, \cdots, a_1, a_0]$, where a_k indicates which $DCell_k$ the server belongs to, and a_0 is the number of the server in its $DCell_0$. With a tuple, a server can be identified by a unique ID $uid_k \in [0, t_k)$ (t_k is the number of servers in a $DCell_k$), by simply calculating $uid_k = a_0 + \sum\limits_{j=1}^{k} (a_j \times t_{j-1})$.

With its ID, a server in $DCell_k$ can be given a recursive defined expression as $[a_k, uid_{k-1}]$, where a_k is the number of the $DCell_{k-1}$ that the server belongs to, and uid_{k-1} is its ID in its $DCell_{k-1}$. In [5], DCell defines its own packet header in which 32-bit addressing is used.

2) DCellRouting: DCellRouting [5] is a simple routing algorithm for unicast, making use of the structure of DCell, and the addressing scheme discussed in IV-E1. However, it does not take potential failures into consideration, which will be addressed in an updated version of DCellRouting. DCellRouting uses a divide-and-conquer approach. Assume that the source and the destination are in the same $DCell_k$, but in different $DCell_{k-1}$s. The algorithm will first find the link (n_1, n_2) connecting the two $DCell_{k-1}$s, and thus the routing is divided into two sub-paths: from source to n_1, and from n_2 to destination. The procedure is repeated until finally all the sub-paths are direct links. It is proved in [5] that the maximum path length in DCellRouting is at most $2^{k+1} - 1$. Considering

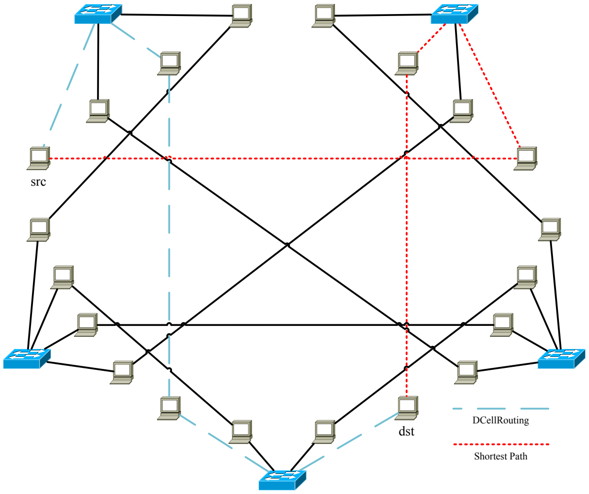

Fig. 9. DCellRouting and Shortest Path

that a small k is sufficient to support a large number of nodes, this algorithm can find the route quite fast.

Although the average length of paths found by DCellRouting is quite close to the shortest path [5], DCellRouting is not a shortest-path routing scheme. This can be shown in figure 9. The shortest path between the source and the destination is of length 4, while the route decided by DCellRouting is of length 5. This happens when the source and destination, although not in the same $DCell_{k-1}$, are both connected to another $DCell_{k-1}$. In such cases the shortest path between two nodes in different $DCell_{k-1}$s is not via the link connecting the two $DCell_{k-1}$s they belong to, but via the $DCell_{k-1}$ they are both connected to.

3) DCellBroadcast: DCellBroadcast is a robust broadcasting scheme for DCell. In DCellBroadcast, when a server initializes a broadcast, it will send the packet to all its $k+1$ neighbors. When receiving a broadcast packet, a server will check whether it has already received the same packet earlier; if not, it will broadcast the packet to all its neighbors. The steps for a broadcast packet to reach every server in the DCell is limited by the diameter (the longest of all the shortest paths between two nodes) of the DCell, $2^{k+1}-1$. This scheme generates duplicate broadcast packets, but is robust since it can send the packet to every server as long as the network is connected.

F. BCube

1) Addressing in BCube: As mentioned in III-B2, each $BCube_l$ contains n $BCube_{l-1}$s, and each $BCube_0$ contains n servers. In [6], a server in a BCube is identified with an address array $a_k a_{k-1} \cdots a_0$, where $a_i \in [0, n-1]$, $i \in [0, 1)$. An equivalent way of determining this address array is calculating the *BCube Address* $baddr = \sum_{i=0}^{k} a_i n^i$. A switch is similarly identified in BCube: $< l, s_{k-1} s_{k-2} \cdots s_0 >$, where $s_j \in [0, n-1]$, $j \in [0, k-1]$, $0 \le l \le k$ is the level of the switch.

2) BCubeRouting: Single-path Routing in BCube: The structure of BCube and the addressing scheme offers a simple way to implement single-path routing in BCube. In the addressing scheme, servers connected to the same switch differs in only one digit in their addresses, i.e., the Hamming distance between them is 1. Each packet includes a BCube header between the Ethernet header and IP header. When forwarding a packet from source to destination, the intermediate nodes will find out the different digits in their addresses, and change one digit in each step. This implies that the packet is transmitted from one server to another through the switch connecting both of them. These intermediate switches are uniquely determined as long as the order of changing the digits is determined. In this routing scheme, a packet needs to travel through at most $k+1$ switches and k servers to arrive at the destination.

G. Summary

Earlier in this section, several routing mechanisms designed for different architectures are discussed. Now the different schemes are compared, and from this comparison some basic guidelines for designing path selection algorithms for a data center network can be derived. Table VI shows the comparison of some properties of different architectures.

1) Addressing: An addressing scheme that leverages the regular architecture of a data center network can lead to effective routing. It has to be scalable because of the large number of servers in a data center. It does not necessarily have to follow the addressing schemes used in the Internet at large.

Most architectures discussed mainly use switches, eliminating routers. The "switch" mentioned here works on the second OSI layer and uses MAC addresses to identify NICs. Eliminating routers makes it possible to apply layer-2 addressing and switching in data centers, as illustrated by IV-B5 and IV-D. It should be noted that the addressing schemes discussed in sections **??**, IV-E and IV-F are not necessarily layer-3 addressing; they can be expressed in either MAC addresses or IP addresses, using routing protocols running at the corresponding layer. When routing with layer-3 addresses, the protocol stack does not need to be different when forwarding on Ethernet headers - IP headers are carried transparently. However, switching with layer-2 addresses eliminates the need for intermediate nodes to analyze every packet to get the layer-3 addresses and re-encapsulate within the layer-2 frame. In data centers, where high-speed, and scalable routing is required, it is meaningful to design a new protocol stack if it actually enhances the performance of the network.

2) Centralized and Distributed Routing: Except PortLand and VL2, all the other routing schemes discussed use a distributed approach. On the other hand, Portland and VL2 can be applied to any multi-rooted tree architecture [4], [26]. It is not restricted by the architecture itself to decide whether to use a centralized approach or a distributed approach when designing routing protocols in a data center. Each approach has its own advantages and disadvantages. Here some of the advantages of each approach are summarized.

Advantages of centralized routing include:

1) Optimized routing: The centralized system has the complete view of the entire data center, enabling it to make

TABLE VI
ROUTING SUMMARY

	Fat Tree [3]	DCell [5]	BCube [6]	Portland [26]	VL2 [4]
Address Level	IP	MAC/IP	MAC/IP	MAC	MAC
Address Resolution	Distributed	Distributed	Distributed	Centralized	Centralized
Route Decision	Intermediate	Source	Source	Intermediate	Intermediate
Compatibility	Specialized	Specialized	Specialized	Compatible	Compatible

optimal routing decisions thus ensuring better use of the hardware.

2) Greater ability to deal with faults: A centralized system can gather fault information from the entire data center, so that it is easier to judge the effect of a fault and mechanisms to handle it.
3) Easier configuration: Centralized routing protocol are deployed on one or several nodes inside a data center. Only these nodes need to be reconfigured when applying a different routing algorithm.

Advantages of distributed routing include:

1) Robustness: The routing function and information is distributed across all the servers in the data center. The routing protocol can operate as long as the data center remains connected.
2) More effective routing decision: Routing decisions are made according to local and neighbor information only, saving significant time and effort needed to gather information about the entire network.
3) Less overhead: Computers do not have to exchange routing information in a wide area so that bandwidth can be saved.

The two approaches are not mutually exclusive. For example, there is a distributed location discovery mechanism in the centralized PortLand [26] system. Most distributed routing protocols will gather information from servers inside a certain area to make a "centralized" routing decision in the area.

V. TRAFFIC ENGINEERING OF DATA CENTERS

The nature of data traffic is fundamental to the design of data centers. Data traffic includes traffic between data centers and the outside Internet, and traffic inside data centers. In this section we discuss traffic engineering of data centers.

When exploring the nature of data traffic, researchers usually use *flows* to describe data traffic. A *flow* is a sequence of packets that share common values in a subset of fields of their head, e.g., the 5-tuple $(src, src\_port, dst, dst\_port, protocol)$. The existence of a flow usually implies that these packets are sequentially and logically related. By allocating packets into flows, routing protocols can achieve load-balancing by distributing flows into different parallel paths between two nodes, while avoiding packet reordering problem by limiting a flow into only one path during a given time.

A. Traffic Characteristics inside Data Centers

According to the purpose of the data center, the method of collecting traffic information and the granularity of information, researchers have explored data center traffic patterns from different angles of view.

1) Traffic in Query Processing Data Center: Query processing is important for applications that need to analyze massive data sets, e.g., web search. Kandula et al. [40] investigated network related events from 1,500 servers which represent a logical distributed query processing cluster in a data center for over 2 months. The data center, which uses a tree-based topology, consists of a set of commodity servers supporting map reduce jobs and a distributed replicated block store layer for persistent storage. The jobs are written in a high-level SQL like language. According to [40], the main characteristics of the traffic can be summarized as:

1) *Traffic Patterns*. Two patterns, the so-called work-seeds-bandwidth pattern and scatter-gather pattern comprise the most of the traffic in the data center. Work-seeds-bandwidth pattern shows that a large chunk of data traffic is among servers within a rack. In fact, using more detailed data, the researchers showed that the probability of exchanging no data is 89% for server pairs inside the same rack, and 99.5% for server pairs from different racks. And scatter-gather pattern shows the traffic resulted in by the map-reduce applications which require servers to push/pull data to/from many servers across the cluster. A server either talks to almost every server or to less than 25% servers within the same rack. Further, a server either does not talk to any server outside its rack, or talks to about 1-10% of servers outside its rack.
2) *Congestion*. Congestion is very often in the data center. 86% of links experience congestion lasting for more than 10 seconds, and 15% of links experience congestion lasting for more than 100 seconds. Besides, most congestions are short-lived. More than 90% congestions last for less than 10 seconds, while the longest lasted 382 seconds. There is high correlation between congestion and read failures. In map-reduce scenario, the reduce phase are responsible for a fair amount of the network traffic. The extract phase, which parses data blocks also contributed a fair amount of the flows on high utilization links.
3) *Flow Characteristics*. 80% of flows last for less than 10 seconds, and about less than 0.1% last for longer than 200 seconds. More than half of the total bytes are in flows less than 25 seconds.

These characteristics decide how to design network protocols for data centers. For example, the fact that most data exchanging happens inside racks requires more bandwidth between servers in the same rack. Congestions are responsible for a large chunk of performance reduction, and the phases that cause most congestions should be carefully designed. Furthermore, since most data are transferred in short flows, it is not sufficient to schedule long flows only.

2) SNMP Data from Multiple Data Centers: Benson et al. [41] analyzed data center traffic characteristics with the SNMP data collected from 19 commercial data centers, which support a wide range of applications including search, video streaming, instant messaging, map-reduce and web applications. All of these data centers use tree-based topologies, with components distributed in different levels. The researchers offered macroscopic view and microscopic view for the data traffic.

- *Macroscopic View*. Data shows that utilization is significantly higher in core layer than in aggregation and edge layers, and edge links have higher utilization than aggregate links because of their low capacities. Although having highest utilization, core layer suffers least loss rates (expressed by the percentage of data discarded). Aggregation level has the highest loss rate. Another important observation is that most packets (about 50%) can be put into two groups according to their length: around 40B and around 1500B. While 1500B is the length of MTU, 40B is the typical length of controlling messages.
- *Microscopic View*. The researchers use another data set which is comprised of packet traces collected from five switches of one of the data centers to get a more fine-grained view of data center traffic. The data shows that packet arrivals follow an ON/OFF pattern, which means that most packets arrive in groups and there are obvious intervals between these groups. It is shown in [41] that lognormal curve produces the best fit to both the distributions of interval times (the OFF periods) and the ON periods.

3) Other Traffic Patterns: Besides the two papers above, there are analysis for certain traffic patterns in other literatures. Alizadeh et al. [42] show that

VI. Data Center Transport Protocols

Although many researchers assume that data center networks use the standard TCP, there are significant proposals in designing special network protocols suitable for data center networks. Each of them emphasizes special features of data center networks and optimizes some performance metrics.

A. Data Center TCP (DCTCP)

Alizadeh et al. proposed DCTCP, a TCP-like protocol designed for data centers in [42]. The goal of DCTCP is to achieve high burst tolerance, low latency and high throughput with commodity shallow-buffered switches. DCTCP achieves these goals by reacting to congestion in proportion to the extent of congestion. It uses a marking scheme at switches that sets the Congestion Experienced (CE) codepoint of packets as soon as the buffer occupancy exceeds a fixed small threshold. The source node reacts by reducing the window by a factor which depends on the fraction of marked packets: the larger the fraction, the bigger the decrease factor. The DCTCP algorithm consists of three main components:

1) *Simple Marking at the Switch*: There is only one parameter - the marking threshold K. A packet is marked with the CE codepoint if the queue occupancy is larger than K on its arrival.
2) *ECN(Explicit Congestion Notification)-Echo at the Receiver*: Unlike standard TCP, the way information in the CE codepoint of a received packet in DCTCP is conveyed back to the sender. DCTCP achieves this setting the received code point in the ACK packet header. However, to reduce the load on the data center, Delayed ACKs are used. One cumulative ACK for every m consecutively received packets.
3) *Controller at the Sender*: The sender maintains an estimate of the fraction of packets that are marked, which is updated once for every window of data. After estimating this fraction the sender decides whether the network is experiencing a congestion event, and changes the window size accordingly. This is unlike in standard TCP where congestion events detected with triple-duplicate ACKs lead to a halving of the congestion window.

DCTCP maintains most of the other TCP features such as Slow Start, additive increase in congestion avoidance and recovery from packet loss.

B. Safe and Effective Fine-grained TCP Retransmissions for Data Centers

VII. Performance Enhancement

We discussed some basic routing schemes for various topologies in section IV. These technologies establish and maintain connections in data center networks. However, as described in III-D3, most data center networks offer hardware redundancy. Hardware redundancy is not only useful in dealing with potential failures, but can also be exploited to enhance network performance when there are no failures. In this section, we discuss several approaches proposed by research to exploit the hardware redundancy to enhance network performance under non-faulty conditions; using hardware redundancy to deal with faults will be discussed in section VIII.

A. Centralized Flow Scheduling in Fat Tree

Al-Fares et al. propose a centralized flow scheduler for fat tree in [3]. Large flows that last relatively long time are the main concern. It is assumed in this scheme that at most one large flow originates from one host at a time. Only two kinds of components, the edge switches and the central flow scheduler take part in the management of flows. An edge switch assigns a new flow to its least loaded port by local judgment. However, it will keep tracking the growth of these flows. Once a flow grows above a threshold, the edge switch will send a notification to the central scheduler with the information of large flows. The central scheduler is responsible for reassigning paths to large flows. It will keep track of all the large flows in the network. When it gets notification of a new inter-pod large flow, it scans the core switches to find one which is on a path without reserved links. On finding such a path, the central scheduler will mark all its links as reserved, and inform the relevant switches on the path. For intra-pod large flows, the central scheduler only needs to carry out a scan on the aggregation switches in that pod and find one path without reserved link. When a flow is not updated for a given period of time, the central scheduler will decide that the flow is no longer active, and clears the reservations. During the execution of the procedure scanning and path assigning mentioned above, the originating edge switch of the large flow will continue to transmit it rather than stop and wait.

1) Hedera's Flow Scheduling for Fat Tree: Hedera, another centralized flow scheduling system for fat tree, is proposed in [27]. Hedera detects large flows at edge switches, and selects a path according to the result of the estimated demand of large flows.

Network flow can be limited by the bandwidth between the source and destination, or the capacity of the ports on them. Hedera considers the bandwidth limitation only. The target of demand estimation is to fairly assign all available bandwidth to existing flows. Hedera uses a $N \times N$ matrix to estimate the demand of all flows, where N is the total number of computers. An element $[i, j]$ contains 3 values: the number of flows from computer i to computer j, the estimated demand of each of the flows, and a flag marking whether the demand of a flow has converged (i.e., fully utilizing available bandwidth without limiting other flows). It is assumed that multiple flows from the same source to the same destination has the same demand, and all the demands are expressed in percentages, assuming that the bandwidth between any two computers are the same. The original values consider the capacity of sources only. The idea of the algorithm is to repeat iterations of increasing the flow capacities from the sources and decreasing exceeded capacities at the receivers until all demands converge.

A simulated annealing algorithm is used to design a scheduler in [27]. The definitions of the elements of the algorithm is as follows:

- State s: a set of mappings from destination computers to core switches. Each computer is assigned to a core switch through which the incoming flows of the computer are transmitted. By limiting each computer to a core switch, the cost of searching paths can be significantly reduced, although the result may not be globally optimal.
- Energy function E: the total exceeded capacity of the current state.
- Temperature T: number of remaining iterations.
- Neighbor generator function: swaps the assigned core switches of a pair of computers in the current state s to get a neighbor state s_n.
- Acceptance probability P: the possibility of the transition from current state s to a neighbor state s_n. P is a function of T, E, E_n, where

$$P(E_n, E, T) = \begin{cases} 1, E_n < E \\ e^{c(E-E_n)/T}, E_n \geq E \end{cases}$$

c is a constant. It is suggested in [27] to use $c = 0.5 \times T_0$ for 16-host cluster and $c = 1000 \times T_0$ for larger data centers.

In each iteration of the simulated annealing algorithm, it will jump to a neighbor state s_n with possibility P, and decrease the temperature T. When T drops to 0, the algorithm is finished. Allowing jumping to a state with higher energy can avoid local minima.

2) Random Traffic Spreading of VL2: VL2 [4] makes use of two techniques to offer hot-spot-free performance: VLB (Valiant Load Balancing) and ECMP (Equal Cost MultiPath).

VL2 implements VLB by sending flows through a randomly-chosen intermediate switch, and ECMP by distributing flows across paths with equal cost. Path selection of VLB and ECMP is random, which means no centralized coordination and traffic engineering is needed. However, if there are large coexisting flows, random distribution of flows may lead to persistent congestion on certain links while leaving other links idle. According to the results shown in [4], this does not become a real problem for the tree-based architecture that the researchers used.

B. BCube Source Routing

BCube Source Routing (BSR) [6] is a source routing protocol designed for BCube architecture. In source routing the entire route is decided by the source itself, and it will insert the routing information in the header of every packet. By doing this, the source can control the path without the coordination of intermediate nodes. Besides, the intermediate nodes just need to forward the packets according to the information in the header, which can simplify their functionalities. Before introducing BSR, we start from addressing and simple routing in BCube.

It should be noted that BCubeRouting only uses single paths to transfer the packets between two computers, while according to the order of changing the digits in the addresses, there can be $k + 1$ parallel paths between two computers. Thus BCubeRouting wastes most of the network capacity of BCube. To fully utilize the high capacity of BCube and enable automatic load-balancing, BSR is proposed in [6]. In BSR, the network traffic is divided into flows.

BSR uses a reactive mechanism. On receiving a new flow, the source node sends probe packets along multiple parallel paths towards the destination. Intermediate nodes reply to the probe packets with necessary information about link state. With the information returned from intermediate nodes, the source can make routing decisions. When the source receives the responses, it will calculate a metric from the information, and select the best path. When this path selection procedure is performed, the source node will not hold the flow packets into a buffer, instead, it will initially choose a path from the parallel paths set to start transmitting. After the path selection is completed and a new path is selected, the source node will switch the flow to the better new path. This may cause temporal packet reordering. Considering that path selection can be done in a short time, the switching will not pose performance problem.

The links used by a flow may break during transmission. To address this problem, the source node sends probes periodically to detect network failures. When an intermediate node finds the next hop of a packet is failed, it will send a path failure message to the source node. The source will immediately switch the flow to the next best path in the parallel paths set. Then, after the periodical probing timer expires, the source will start a new round of path selection. Doing this can avoid packet buffering when failure occurs.

When there are multiple concurrent flows between two nodes, they may be distributed to the same path. Besides, they will perform periodical probing at the same time. To avoid this *path oscillation*, a random value can be injected into the timer of periodical path selection.

C. Traffic De-multiplexing in c-Through

Hybrid networks such as c-Through[7] face a special problem: how to assign traffic into two different types of network, electrical and optical. In c-Through, the problem is solved by using VLAN-based routing. The ToR switches are assigned into two different VLANs which are logically separated. This distinguishes the optical network from the electrical network. Since the optical VLAN may be reconfigured frequently according to the changing traffic, spanning tree protocol is disabled in optical VLAN due to its long convergence time. At server side, a management daemon is run on each server to de-multiplex traffic to the optical and electrical network. The assigned path of each packet is decided according to its destination. In practice, to make more efficient use of the optical network, the optically-connected destinations will have higher transmission priority compared to electrically-connected destinations.

D. Summary of Performance Enhancement Technologies

The technologies we discussed above uses the abundant hardware redundancy available in the data center networks to get better performance. In this section we summarize these technologies to find some guidelines to design routing techniques with better performance. Table VII shows the comparison of different performance enhancement technologies.

1) Multi-path Routing: The hardware redundancy of the data center networks enables multi-path routing, which can make better use of the idle components, and avoid overloaded components. Some conditions must be met when multi-path routing is used:

1) There must be hardware redundancy. Topologies such as basic tree have no hardware redundancy. There is only one deterministic path between two computers, so no multi-path routing can be applied.
2) Multi-path routing should be able to achieve fairness when assigning data transmission to a path, i.e., data traffic should be dispersed, and every path is not necessarily the “best” path. (The meaning of “best” is decided according to the metric used to evaluate a path; for example, a best path can be a path with least latency, or largest bandwidth.)
3) Paths may overlap at some links/nodes.

It should be noted that in data centers with hardware redundancy, even if source routing is not used, there can be other ways to make use of the idle redundant links: nodes can detour packets according to local information about the network conditions. This is also referred to as detour routing. It offers a more flexible way to make use of idle links. The disadvantage of detour routing is that there must be mechanisms to avoid possible rings.

A node need not to be able to send data packages through multiple links concurrently to the same destination. The multiple links can be stored at the source node, and take effect only when the current path being used fails. However, the ability of using multiple links concurrently is helpful to disperse data traffic more evenly, which requires more careful design to avoid reordering of sequential packets. Flow scheduling is designed to address this problem.

2) Flow Scheduling: Grouping data into flows helps researchers to get a better view of data transmissions in data centers so that the routing protocols can achieve fairness and efficiency in large scale. Typically the data to identify a flow is intercepted by the routing protocol and judged by the protocol itself. No special field inside a data packet such as $flow\_id$ is defined to identify a flow, because that would need redesign of the whole network layer.

Flow scheduling is meaningful only when there are multiple optional paths, so that different flows can be placed on different paths according to their capability. There are two approaches to direct a flow: decide the whole path of the flow at the source, or make routing decisions at intermediate nodes. The latter approach needs every intermediate node to identify a flow again and again, while in the former approach intermediate nodes only need to extract routing information from a packet and send it to the next hop on the path.

Flow scheduling solves the reordering problem introduced by multi-path routing. The data packets in the same flow are transferred sequentially without being reordered.

Flow scheduling needs to gather information from the first packets of a flow, while at the same time these packets may have been sent out through different paths. According to current research [40] of flows in data centers, more than 80% of flows last less than 10 seconds, while only less than 0.1% of flows last more than 2000 seconds. On the other hand, 50% of data are in flows lasting less than 25 seconds. The main challenges in scheduling flows include:

1) It is impossible to predict whether a flow is a large flow; if two large flows are routed to the same path, the traffic will be unbalanced.
2) The first packets of a flow may be reordered. Given that most flows last a short time, and most data are in these short flows, the reordering problem still cannot be solved well.

To solve the first problem, changing the path of a flow during its transmission should be allowed to keep network traffic balanced, although it may lead to packet reordering. For the second problem, maintaining a buffer at the source may help, but it may be expensive, and increase latency.

VIII. Fault Tolerant Routing

Considering the large number of computers and switches used in a data center, it is unlikely that the system can keep running without failures throughout its lifetime. A data center network must be designed to be able to recover automatically

TABLE VII
PERFORMANCE ENHANCEMENT TECHNIQUES

	Fat Tree [3]	BCube [6]	Hedera [27]	VL2 [4]
Flow Scheduling Algorithm	Reassign Paths for Large Flows	Metric-based Routing	Simulated Annealing	Random Path Selection
Decision Maker of Multi-path Selection	Central Scheduler	Source Node	Edge Switches	Source Node

from common failures, and maintain most of its performance in the meantime. On hardware level, most data center topologies employ redundant links to ensure connectivity in case of hardware failures. Moreover, the routing mechanisms running above make use of the redundancy offered by the hardware to recover from communication failures. Since we have discussed hardware redundancy in III-D3, we will turn to the software mechanisms, i.e., the fault tolerant routing protocols used by different data center networks.

A. Fault Models

Failures in data centers may be caused by different components. Classifying the failures into *fault models* helps to figure out the characteristics of different failures and methods to detect and deal with certain types of failures. Failures can be classified based on the following aspects.

1) Failure Type: This attribute identifies the level at which components are diagnosed as having failed [35]. Either the whole component fails, or one of the links connected to it fails. The former is referred to as *node failure*, and the latter as *link failure*.

For components such as the computers in a fat trees, that have only one link, node failure and link failure have the same effect. On the other hand, a node failure of a node with multiple links can be considered as link failures for all its links. A mechanism that can deal with link failures can also deal with node failures. However, this does not preclude the need for distinguishing between node failures and link failures. For example, marking a node as failed means that the routing protocol can choose to detour around that node without checking its links one by one, thus saving significant time.

It should be noted that link failures and node failures are not necessarily hardware failures. The real meaning of a link failure or node failure is that a link or all the links connected to a component are no longer feasible to transmit data from the application's point of view.

2) Fault Region: A *fault region* is a useful concept when multiple related components are faulty. For example, in a data center constructed of racks, power failure of a rack will disconnect all the components in the rack. If a routing protocol can figure out rack failures, it can detour the whole rack. Since the positioning of the components may vary, it depends on the physical architecture and logical architecture of a data center to define its fault regions.

3) Fault Neighborhood: This attribute reflects the extent of dissemination of fault information. According to the distance from the faulty node to the selected node, there can be 3 different modes: *global*, in which every node in the network has the information of all faulty nodes; *local*, in which only the adjacent nodes of a faulty node are aware of its information; *k-neighborhood*, only the information of faulty nodes within distance (usually using hops to express the distance) k is tracked. The global mode can help to make better route choices based on global information, while needs more storage and periodic global synchronization. On the other hand, routing based on local information is simpler and faster, while may lead to longer paths. The k-neighborhood mode can make a balance between the two extreme conditions.

4) Failure Mode: This attribute shows the duration of failures. A *static* failure is present since the system starts up, while a *dynamic* failure occurs during the operation of the system. Both kinds of failures are considered to be *permanent*. In contrast, a fault can be *transient*. Dynamic failures and transient failures are the main challenge when designing a routing protocol.

5) Failure Time: This attribute shows the frequency of component failures. Two values, the *mean time between failures (MTBF)* and *mean time to repair (MTTR)* are used to evaluate the possibility of a failure and the time needed to recover from it. For example, if MTBF and MTTR are close for one kind of component, it is likely that there will be multiple concurrent faulty components in the data center. According to the research of MTTR in [4], 95% of failures are resolved in 10 minutes, 98% in less than an hour, 99.6% in less than a day, but 0.09% failures last for more than 10 days. These percentiles may vary in data centers using different hardware, but can serve as a reference for the distribution of MTTR.

6) Taxonomy of Faults: Figure 10 shows a taxonomy of fault models we have mentioned. One fault can be classified into multiple models. Although one routing protocol may not make use of all of the models above, distinguishing different faults with these models will help a routing protocol to deal with faults according to their characteristics.

B. Link Failure Response in Fat Tree

Al-Fares et al. [3] proposed a failure broadcast protocol to detect link failures in fat tree architecture. The architecture only discusses simplest link failures, without mentioning other fault models in VIII-A.

As described in III-D3, there is no redundancy link between a computer and an edge switch, and the only way to deal with such a failed link is to add extra links between computers and edge switches. On the other hand, link failures between switches can be classified as:

- between edge switches (lower in a pod) and aggregation switches (upper in a pod);
- between aggregation switches and core switches.

A *Bidirectional Forwarding Detection* session (BFD [43]) is used to detect link failures. The protocols exploits different

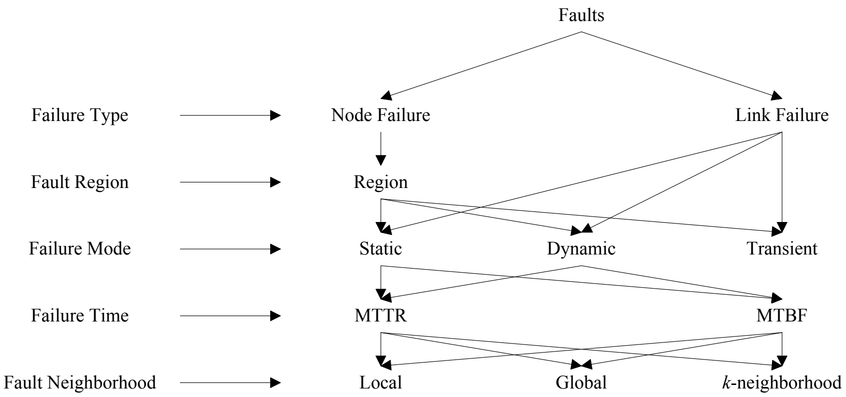

Fig. 10. A Taxonomy for Faults

mechanisms to deal with the two kinds of link failures. For a link failure between an edge switch and an aggregation switch, three kinds of traffic may be affected:

1) Outgoing traffic originated by a computer connected to the edge switch. In this case, the edge switch will detect the link failure, the cost of the link will be set to infinity and another available link will be chosen.
2) Intra-pod traffic that uses the aggregation switch as an intermediary. In this case this aggregation switch detects the link failure, and will broadcast this link failure to every other edge switch in the pod, and they will try to detour this link.
3) Inter-pod traffic coming to the aggregation switch from a core switch. In this case, the aggregation switch detects the link failure, and will broadcast this failure to all the core switches. The core switches then inform all aggregation switches, so that they will detour the affected core switch when sending traffic to the subnet connected to the edge switch.

For a link failure between an aggregation switch and a core switch, two kinds of traffic may be affected:

1) Outgoing inter-pod traffic. In this case the failure is detected by the aggregation switch. It will set this link as unavailable, and simply choose the link to another core switch.
2) Incoming inter-pod traffic. In this case the failure is detected by the core switch. In this case it will broadcast this link failure to all other aggregation switches connected to it, announcing that it cannot send traffic to that pod. The aggregation switches will detour this core switch when trying to send traffic to that pod.

When the failed link is recovered, it will be detected by the BFD session, and the measures above will be reversed. We can see that in this fault tolerant scheme the fault information is sent to nodes at most two hops away, so it uses 2-neighborhood mode.

1) Fault Tolerant Routing in PortLand and Hedera: Niranjan Mysore et al. [26] proposed a fault tolerant routing scheme for PortLand, which also uses link failure to represent all possible failures. Since PortLand exploits a centralized fabric manager to maintain the entire network, it is more effective in gathering and spreading fault messages. The periodic LDM mentioned in IV-B4 can also serve as probe of link failures. If a node does not receive LDM for a certain period of time, it will assume that a link failure has occurred and send the information to the fabric manager. The fabric manager uses a *fault matrix* to store the status of each link in the whole network. On receiving a fault message, the fabric manager updates the fault matrix, and informs all the affected nodes of this link failure. These nodes will reroute with the new information. On recovery of a link failure, the procedure is similar except that the information is a recovery message instead of a fault message. With the fabric manager, the extent of the fault information can be easily controlled. Besides, by using multicast instead of broadcast, the cost of spreading the fault message can be reduced to minimum.

Hedera [27], whose implementation augments PortLand routing and fault tolerant protocols, has the same mechanisms of dealing with failures.

C. DCell Fault-tolerant Routing

DCell Fault-tolerant Routing (DFR) uses DCellRouting and DCellBroadcast to enable fault-tolerant routing in DCell. It defines three types of failures: server failure, rack failure and link failure, and it uses three techniques, local reroute, local link-state and jump-up to deal with these failures respectively.

- Local Reroute: bypass a link failure according to local decisions only. When a link failure occurs, local reroute will be done at one of the two ends of the failed link. If the failed link (n_1, n_2) is a level-l link (i.e., the two ends of the link is in the same $DCell_l$ but in different $DCell_{k-1}$s), n_1 will choose a node in another $DCell_{k-1}$ that does not contain n_2 as its *proxy*. As long as not all links from n_1's $DCell_{k-1}$ to other $DCell_{k-1}$s are failed, such a proxy can always be found. After the proxy is selected, DCellRouting will be executed on the proxy to find a path to the destination. Local reroute may transmit the packet to the other end of a failed link, and if it is a failed node, local reroute cannot bypass it.
- Local Link-state: use link-state information gathered with DCellBroadcast to bypass node failures. By means of DCellBroadcast, a node in a $DCell_b$ (the range of a

broadcast packet is delivered, where number b is defined according to hardware capability, generally contains a rack of switches and the computers connected to them) can know the status of all the links connecting its own $DCell_b$ and other $DCell_b$s. In this scheme, a node will calculate the route inside its $DCell_b$ local with link-state information and the Dijkstra algorithm. When a link failure happens, the node will invoke local-reroute to find a proxy. Since the decision is made upon link-state of the whole $DCell_b$, no matter whether the link failure implies a potential node failure, the failed node can be bypassed.
- Jump-up: a scheme to address rack failure, in which the whole $DCell_b$ fails. In local link-state, when a whole $DCell_b$ in a path fails, the packet will be re-routed endlessly round the $DCell_b$. To address such kind of failure, the jump-up scheme is introduced. When a link failure is detected, and the first attempt to re-route to another path in the $DCell_b$ is unsuccessful, it will be assumed that the whole $DCell_b$, i.e., the whole rack is down. Then the receiver of the re-routed packet will "jump-up" to a higher level DCell to bypass the whole failed $DCell_b$. Besides, since it is impossible to reach a node inside a failed $DCell_b$, a time-to-live (TTL) field is added into every packet to avoid endless re-routing.

By using all the techniques above, DFR can handle link failure, node failure and rack failure.

D. Fault Tolerant Routing in BCube

As discussed in VII-B, BCube can detect link failures by sending periodic probes, and turn to backup paths when failure occurs.

E. Summary of Fault Tolerant Routing Protocols

We discussed some fault tolerant routing techniques in this section. From a comparison between them, we can find some guidelines of designing fault tolerant routing protocols. Table VIII gives some comparisons between the fault tolerant techniques we discussed. Some of them are not mentioned in this section but in previous sections, because performance enhancement techniques sometimes can also serve as fault tolerant techniques.

1) All these protocols have their own mechanisms to detect link failures. Since node failures can be viewed as a set of link failures, being able to deal with link failures also means being able to deal with node failures or failures affecting even more components, e.g, rack failures.
2) The MTTR and MTBF of different components should be considered when designing fault tolerant mechanisms, e.g., when deciding the frequency of running failure detecting processes.
3) Spreading failure information can reduce the time wasted to wait for a timeout. However, this may involve non-related nodes, and updating such information may occupy a lot of bandwidth.
4) Protocols should be designed to discover components recovered from failure.

Some fault tolerant techniques such as keeping back-up routes in routing tables can serve to enhance performance when there are no faults. It is meaningful to consider both aspects together when designing the routing protocol.

IX. Conclusions

In this survey, we discussed some representative topologies of data center networks. We discussed the properties of these topologies, the routing protocols designed for them, the techniques for enhancing their performance, and fault tolerant techniques. Researchers are still proposing potential new topologies for future data center networks. Any topology has to strike a balance between performance, cost and sustainability.

Acknowledgment

The work described in this paper has been supported by HK RGC under RGC-Competitive Earmarked Research Grant HKUST 617907.

References

[1] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "Above the clouds: A berkeley view of cloud computing," EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2009-28, Feb 2009. [Online]. Available: http://www.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-28.html

[2] A. Greenberg, J. Hamilton, D. A. Maltz, and P. Patel, "The cost of a cloud: research problems in data center networks," *SIGCOMM Comput. Commun. Rev.*, vol. 39, no. 1, pp. 68–73, 2009.

[3] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," in *Proceedings of the ACM SIGCOMM 2008 conference on Data communication*. ACM, 2008, pp. 63–74.

[4] 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," *SIGCOMM Comput. Commun. Rev.*, vol. 39, no. 4, pp. 51–62, 2009.

[5] C. Guo, H. Wu, K. Tan, L. Shi, Y. Zhang, and S. Lu, "DCell: A scalable and fault-tolerant network structure for data centers," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 4, pp. 75–86, 2008.

[6] 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," *ACM SIGCOMM Computer Communication Review*, vol. 39, no. 4, pp. 63–74, 2009.

[7] G. Wang, D. Andersen, M. Kaminsky, K. Papagiannaki, T. Ng, M. Kozuch, and M. Ryan, "c-through: Part-time optics in data centers," in *ACM SIGCOMM Computer Communication Review*, vol. 40, no. 4. ACM, 2010, pp. 327–338.

[8] N. Farrington, G. Porter, S. Radhakrishnan, H. Bazzaz, V. Subramanya, Y. Fainman, G. Papen, and A. Vahdat, "Helios: a hybrid electrical/optical switch architecture for modular data centers," in *ACM SIGCOMM Computer Communication Review*, vol. 40, no. 4. ACM, 2010, pp. 339–350.

[9] 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 *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*. USENIX Association, 2012, pp. 18–18.

[10] K. Chen, C. Hu, X. Zhang, K. Zheng, Y. Chen, and A. Vasilakos, "Survey on routing in data centers: insights and future directions," *Network, IEEE*, vol. 25, no. 4, pp. 6–10, 2011.

[11] M. Bari, R. Boutaba, R. Esteves, L. Granville, M. Podlesny, M. Rabbani, Q. Zhang, and M. Zhani, "Data center network virtualization: A survey," *Communications Surveys Tutorials, IEEE*, vol. PP, no. 99, pp. 1 –20, 2012.

[12] C. Kachris and I. Tomkos, "A survey on optical interconnects for data centers," *Communications Surveys Tutorials, IEEE*, vol. 14, no. 4, pp. 1021 –1036, quarter 2012.

TABLE VIII
FAULT TOLERANT TECHNIQUES

	Fat Tree [3]	DCell [5]	BCube [6]	PortLand [26] & Hedera [27]
Failure Types	Link Failure	Link Failure/ Server Failure/ Rack Failure	Link Failure	Link Failure
Failure Detecting Mechanisms	BFD (Periodic Probes)	DCellBroadcast (Reactive Probes)	Periodic Probes	LDM (Periodic Probes)
Area of Spreading Failure Information	Edge/ Aggregation/ Core	Local/ Rack/ Inter-rack	Source	All Nodes on the Path
Decision Maker	Intermediate	Intermediate	Source	Fabric Manager

[13] K. Wu, J. Xiao, and L. Ni, "Rethinking the architecture design of data center networks," *Frontiers of Computer Science*, vol. 6, pp. 596–603, 2012. [Online]. Available: http://dx.doi.org/10.1007/s11704-012-1155-6

[14] H. Zimmermann, "OSI reference model–The ISO model of architecture for open systems interconnection," *Communications, IEEE Transactions on*, vol. 28, no. 4, pp. 425–432, 1980.

[15] K. Kant, "Data center evolution: A tutorial on state of the art, issues, and challenges," *Computer Networks*, vol. 53, no. 17, pp. 2939–2965, 2009.

[16] N. Boden, D. Cohen, R. Felderman, A. Kulawik, C. Seitz, J. Seizovic, and W. Su, "Myrinet: A Gigabit-per-Second Local Area Network," *IEEE Micro*, vol. 15, no. 1, pp. 29–36, 1995.

[17] L. Ong and J. Yoakum, "An Introduction to the Stream Control Transmission Protocol (SCTP)," RFC 3286 (Informational), Internet Engineering Task Force, May 2002. [Online]. Available: http://www.ietf.org/rfc/rfc3286.txt

[18] C. Perkins, "IP Mobility Support," RFC 2002 (Proposed Standard), Internet Engineering Task Force, Oct. 1996, obsoleted by RFC 3220, updated by RFC 2290. [Online]. Available: http://www.ietf.org/rfc/rfc2002.txt

[19] Hyper-v quick migration basp team connectivity loss - ibm bladecenter and system x. [Online]. Available: http://www-947.ibm.com/support/entry/portal/docdisplay?lndocid=migr-5087137

[20] D. Thaler and C. Hopps, "Multipath Issues in Unicast and Multicast Next-Hop Selection," RFC 2991 (Informational), Internet Engineering Task Force, Nov. 2000. [Online]. Available: http://www.ietf.org/rfc/rfc2991.txt

[21] IEEE Standards Association, "Media access control (mac) bridges and virtual bridge local area networks," 2011.

[22] C. Kim, M. Caesar, and J. Rexford, "Floodless in seattle: a scalable ethernet architecture for large enterprises," in *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 4. ACM, 2008, pp. 3–14.

[23] D. Farinacci, V. Fuller, D. Meyer, and D. Lewis, "Locator/ID separation protocol (LISP)," Internet Draft (draft-ietf-lisp-23), May 2012, (Work in progress).

[24] R. Perlman, D. Eastlake 3rd, D. Dutt, S. Gai, and A. Ghanwani, "Routing Bridges (RBridges): Base Protocol Specification," RFC 6325 (Proposed Standard), Internet Engineering Task Force, Jul. 2011, updated by RFCs 6327, 6439. [Online]. Available: http://www.ietf.org/rfc/rfc6325.txt

[25] E. Nordmark and M. Bagnulo, "Shim6: Level 3 Multihoming Shim Protocol for IPv6," RFC 5533 (Proposed Standard), Internet Engineering Task Force, Jun. 2009. [Online]. Available: http://www.ietf.org/rfc/rfc5533.txt

[26] 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," *ACM SIGCOMM Computer Communication Review*, vol. 39, no. 4, pp. 39–50, 2009.

[27] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat, "Hedera: Dynamic flow scheduling for data center networks," in *Proceedings of the 7th USENIX conference on Networked systems design and implementation*. USENIX Association, 2010, p. 19.

[28] D. Li, C. Guo, H. Wu, K. Tan, Y. Zhang, and S. Lu, "Ficonn: Using backup port for server interconnection in data centers," in *IEEE INFOCOM*, 2009.

[29] H. Wu, G. Lu, D. Li, C. Guo, and Y. Zhang, "MDCube: a high performance network structure for modular data center interconnection," in *Proceedings of the 5th international conference on Emerging networking experiments and technologies*. ACM, 2009, pp. 25–36.

[30] H. Abu-Libdeh, P. Costa, A. Rowstron, G. O'Shea, and A. Donnelly, "Symbiotic routing in future data centers," *ACM SIGCOMM Computer Communication Review*, vol. 40, no. 4, pp. 51–62, 2010.

[31] C. E. Leiserson, "Fat-trees: universal networks for hardware-efficient supercomputing," *IEEE Trans. Comput.*, vol. 34, no. 10, pp. 892–901, 1985.

[32] W. Dally and B. Towles, *Principles and practices of interconnection networks*. Morgan Kaufmann, 2004.

[33] J. Edmonds, "Paths, trees, and flowers," *Canadian Journal of mathematics*, vol. 17, no. 3, pp. 449–467, 1965.

[34] M. Müller-Hannemann and A. Schwartz, "Implementing weighted b-matching algorithms: insights from a computational study," *Journal of Experimental Algorithmics (JEA)*, vol. 5, p. 8, 2000.

[35] J. Duato, S. Yalamanchili, and L. Ni, *Interconnection networks: An engineering approach*. Morgan Kaufmann, 2003.

[36] L. Bhuyan and D. Agrawal, "Generalized hypercube and hyperbus structures for a computer network," *IEEE Transactions on Computers*, pp. 323–333, 1984.

[37] M. Hamdi and R. Hall, "Rcc-full: An effective network for parallel computations," *Journal of Parallel and Distributed Computing*, vol. 41, no. 2, pp. 139–155, 1997.

[38] Y. Zhang, A. Sub, and G. Jiang, "Understanding data center network architectures in virtualized environments: A view from multi-tier applications," *Computer Networks*, vol. 55, pp. 2196–2208, 2011.

[39] L. Popa, S. Ratnasamy, G. Iannaccone, A. Krishnamurthy, and I. Stoica, "A cost comparison of datacenter network architectures," in *Proceedings of the 6th International COnference*. ACM, 2010, p. 16.

[40] S. Kandula, S. Sengupta, A. Greenberg, P. Patel, and R. Chaiken, "The nature of data center traffic: measurements & analysis," in *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement conference*. ACM, 2009, pp. 202–208.

[41] T. Benson, A. Anand, A. Akella, and M. Zhang, "Understanding data center traffic characteristics," in *WREN '09: Proceedings of the 1st ACM workshop on Research on enterprise networking*. New York, NY, USA: ACM, 2009, pp. 65–72.

[42] M. Alizadeh, A. Greenberg, D. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan, "Data center TCP (DCTCP)," *ACM SIGCOMM Computer Communication Review*, vol. 40, no. 4, pp. 63–74, 2010.

[43] D. Katz and D. Ward, "Bfd for ipv4 and ipv6 (single hop)," *draft-ietf-bfd-v4v6-1hop-09 (work in progress)*, 2009.

Yang Liu is a Ph.D. student in The Hong Kong University of Science and Technology, Clear Water Bay, Kowloon, Hong Kong.

Jogesh K. Muppala (M'85-SM'02) received the Ph. D. degree in Electrical Engineering from Duke University, Durham, NC in 1991. He is currently an associate professor in the Department of Computer Science and Engineering, The Hong Kong University of Science and Technology, Clear Water Bay, Kowloon, Hong Kong. He was previously a Member of the Technical Staff at Software Productivity Consortium (Herndon, Virginia, USA) from 1991 to 1992, where he was involved in the development of modeling techniques for systems and software. While at Duke University, he participated in the development of two modeling tools, the Stochastic Petri Net Package (SPNP) and the symbolic Hierarchical Automated Reliability and Performance Evaluator (SHARPE), both of which are being used in several universities and industry in the USA. He was the program co-chair for the 1999 Pacific Rim International Symposium on Dependable Computing held in Hong Kong in December 1999. He also co-founded and organized The Asia-Pacific Workshop on Embedded System Education and Research. He has also served on program committees of many international conferences. He received the Excellence in Teaching Innovation Award 2007. He was also awarded the Teaching Excellence Appreciation Award by the Dean of Engineering, HKUST. Dr. Muppala is a Senior Member of IEEE, IEEE Computer Society and IEEE Communications Society, and a participating representative from HKUST with EDUCAUSE.

Malathi Veeraraghavan is with the Charles L. Brown Department of Electrical and Computer Engineering, University of Virginia, Charlottesville, VA 22904.