DCube: A Family of Network Structures for Containerized Data Centers Using Dual-Port Servers

Deke Guo*,[a], Chaoling Li[b], Jie Wu[c], Tao Chen[a], Xiaolei Zhou[a], Xueshan Luo[a]

[a]School of Information System and Management, National University of Defense Technology, Changsha Hunan 410073, China
[b]National Geographical Survey Center, Beijing 10083, China.
[c]Department of Computer and Information Sciences, Temple University, USA

Abstract

A fundamental goal of datacenter networking is to efficiently interconnect a large number of servers in a cost-effective way. Inspired by the commodity servers in today's data centers that come with dual-port, we consider how to design low-cost, robust, and symmetrical network structures for containerized data centers with dual-port servers and low-end switches. In this paper, we propose a family of such network structure called a DCube, including *H-DCube* and *M-DCube*. The DCube consists of one or multiple interconnected sub-networks, each of which is a compound graph made by interconnecting a certain number of basic building blocks by means of a hypercube-like graph. More precisely, the H-DCube and M-DCube utilize the hypercube and 1-möbius cube, respectively, while the M-DCube achieves a considerably higher aggregate bottleneck throughput compared to H-DCube. Mathematical analysis and simulation results show that the DCube exhibits a graceful performance degradation as the server or switch failure rate increases. Moreover, the DCube significantly reduces the required wires and switches compared to the BCube and fat-tree. In addition, the DCube achieves a higher speedup than the BCube does for the one-to-several traffic patterns. The proposed methodologies in this paper can apply to the compound graph of the basic building block and other hypercube-like graphs, such as Twisted cube, Flip MCube, and fastcube.

Key words: Data center networking, compound graph; Hypercube graph

1. Introduction

As one of the fundamental infrastructures for cloud computing, data centers have recently been studied extensively because of their support of many online applications and infrastructural services [1]. Inside a data center, a large number of servers are interconnected by network devices using a specific networking structure, which is becoming an important area of research.

A number of novel networking structures for large-scale data centers have been proposed recently [2]. These structures can be roughly divided into two categories. One is switch-centric, which organizes switches into structures other than tree and puts the interconnection intelligence on switches. Fat-Tree [3], VL2 [4], PortLand [5], Dragonfly [6], and PERCS [7] fall into this category. The other category is server-centric, which utilizes the rapid growth of the server hardware and multiple NIC ports to put the interconnection and routing intelligence on the servers also. DCell [8], FiConn [9], BCube [10], and BCN [11] fall into the second category. In this setting, the routing capability at each server can be implemented by software-based systems [12], FPGA-based systems [13], and ServerSwitch [14].

The containerized data center takes on a different method of building modern mega data centers [15, 16]. In a containerized data center, a few thousand servers, usually $1k\sim4k$, along with switches, are packed into a standard 20-feet or 40-feet shipping container. The container environment has several advantages: easy wiring, low cooling cost, high power density, etc. [17]. Containerized data centers can be interconnected by an inter-container networking structure, such as uFix [17] and MDCube [18], so as to scale a data center from thousands of servers to millions.

In this paper, we study a simple technical problem: can we build a low-cost, fault-tolerant, and symmetrical network structure for containerized data centers, using commodity servers each only with dual-port and low-end commodity switches? The potential benefits of solving such problem are multifaceted. Firstly, we do not use expensive, high-end switches which are widely used today. Thus, it costs less to build a network structure for containerized datacenters. Secondly, the wiring has a relatively low-cost and low-complexity since each server does not need to have any additional hardware installed except for two NIC ports. Lastly, it can offer an easy-to-build and easy-to-afford testbed at a university or institution [9]. Besides such benefits, most standard, off-the-shelf servers already have two high-speed ports, one primary port and one backup port. Hence, there is no need to physically upgrade the servers when using new servers or reusing servers in existing data centers.

In this paper, we propose a family of low-cost and robust network structures called DCube(n,k) for containerized data

*Corresponding author. Email: guodeke@gmail.com. Tel:+86-731-84576603.

centers with dual-port servers and low-end n-port switches. DCube(n,k) consists of one or $k>1$ interconnected sub-networks, each of which is a compound graph made by interconnecting a certain number of basic building blocks by means of a hypercube-like graph. In each subnetwork, the basic building block is just n/k servers connected to a switch. More precisely, we first design H-DCube which uses the hypercube graph, and we further enhance the aggregated bottleneck throughput significantly by proposing M-DCube which adopts the möbius cube. Note that the möbius cube has a diameter approximately half that of the hypercube and its expected distance is approximately two-thirds the hypercube's expected distance.

H-DCube and M-DCube offer high degrees of regularity and symmetry, which are desirable properties of data center networks. These benefits are obtained at the cost of only two links being associated with each server, regardless of the network size. In addition, DCube provides higher bandwidth for the one-to-one traffic and greatly improves the ability of fault tolerance. Mathematical analysis and simulation results show that the DCube has a higher aggregate bottleneck throughput than DCell as the server or switch failure rate increases. Moreover, the DCube significantly reduces the number of required wires and switches compared to BCube and fat-tree; hence, the construction cost, energy consumption, and cabling complexity are largely reduced. Additionally, the DCube achieves a higher speedup compared to BCube does for one-to-several traffic pattern by constructing more edge-disjoint complete graphs.

DCube, however, cannot achieve the same aggregate bottleneck throughput as BCube, which employs more ports for each server and switches for routing. In fact, the lower ABT (aggregate bottleneck throughput) of Dcube is the tradeoff of a significantly less number of links and switches. Such an issue can be addressed by other techniques at the application layer, such as the locality-aware task placement.

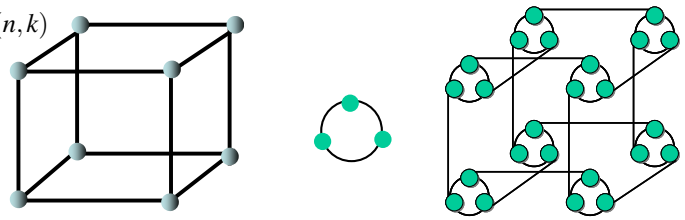
The rest of this paper is organized as follows. Section 2 describes the compound group and related work. Section 3 presents the structures and constructions of the H-DCube and M-DCube. Section 4 proposes dedicated routing schemes for the one-to-one and one-to-several traffic patterns. Section 5 evaluates the properties of the network structures proposed in this paper. Section 7 concludes this paper.

2. Preliminaries

2.1. Compound graph

A compound graph is suitable for constructing large interconnection networks due to its good regularity and expansibility, where many smaller networks at the lowest level are interconnected to constitute a larger network [19]. Consequently, lower level networks support local communication while higher level networks support remote communication.

Definition 1. *Given two regular graphs, G and G_1, a Level-1 regular compound graph $G(G_1)$ is obtained by replacing each node of G by a copy of G_1 and replacing each link of G by a link which connects two corresponding copies of G_1.*



(a) Regular graph G (b) Regular graph G1 (c) Compound graph G(G1)

Figure 1: An illustrative example of the compound graph, which interconnects eight rings by means of the three-dimensional hypercube.

A level-1 regular compound graph $G(G_1)$ employs G_1 as a unit cluster and connects many such clusters by means of a regular graph G. In the resultant graph, the topology of G is preserved, and only one link is inserted to connect two copies of G_1. An additional remote link is associated with each node in a cluster. A *constraint* must be satisfied for the two graphs to constitute a regular compound graph. The node degree of G must be equal to the number of nodes in G_1. Otherwise, an *irregular compound graph* is obtained. For ease of explanation, we show an example of the compound graph in Fig.1.

The basic idea of a compound graph can be extended to the context of a multi-level compound graph, recursively. For ease of explanation, we consider the case where the regular G is a complete graph. A level-2 compound graph $G^2(G_1)$ employs $G(G_1)$ as a unit cluster and connects many such clusters using a complete graph G. More generically, a level-i ($i>0$) graph $G^i(G_1)$ adopts a level-$(i-1)$ graph $G^{i-1}(G_1)$ as a unit cluster and connects many such clusters by a complete graph G.

As we will show in the next section, the topology of DCell is just a multi-level regular compound graph, while the topologies of FiConn and BCN are two different multi-level irregular compound graphs. The topologies of H-DCube and M-DCube proposed in this paper are two types of one-level regular compound graph. Note that the multi-level regular or irregular graph is a natural way to construct hierarchical network. On the other hand, BCube is an emulation of the generalized Hypercube and is an example of the product network [19]. DCell, FiConn, BCN, and BCube are defined by recursively utilizing the method of compound graph or product network.

2.2. Related work

Although several networking structures for large-scale data centers have been proposed recently, they are not very suitable for containerized data centers that are using only dual-port servers and low-end commodity switches. Firstly, the switch-centric network structures require expensive, high-end switches at the top levels in order to alleviate the bandwidth bottleneck to some extent by incurring an even higher cost. We will now discuss DCell, FiConn, and BCN, three enlightening server-centric structures for large-scale data centers.

The key insight behind DCell$_i$ is a level-i regular compound graph $G^i(DCell_0)$ constituted recursively for any $i\geq 1$. More precisely, any high-level DCell is constituted by connecting a given number of Dcells in the next level down via a complete graph. Thus, Dcells at the same level are fully connected with one another. $DCell_0$ is the basic building block in which n

servers are connected to a n port commodity switch. Although DCell has many desirable features for large-scale data centers, it faces obstacles in containerized data centers. Typically, DCell requires more ports and links per server, e.g., 4, for connecting about $4k$ servers via low-end commodity switches. If we want to interconnect near $4k$ dual-port servers, all low-cost switches would have to be replaced with high-end ones, each with a lot of ports which would incur an even higher cost. Additionally, the upcoming containerized data centers may hold more than $4k$ servers to accommodate the service expansion. This further limits the usage of DCell in this setting.

FiConn and BCN are different from DCell: these build a large-scale data center consisting of a large number of dual-port servers. As mentioned in our previous work [20], the key insight behind DCell is the multi-level regular compound graph, while behind FiCoon and BCN is the multi-level irregular compound graph. The multi-level compound graph, however, incurs imbalanced traffic at different levels of links; hence, reducing the resulting aggregate bottleneck throughput. More specifically, those links that are interconnecting building clusters potentially carry higher traffic than links attached to switches, and high-level links always carry much more flows than low-level links in Dcell, FiCoon, and BCN.

Typically, DCell, FiConn, and BCN should have at least a level-2 compound regular or irregular graph for connecting about $4k$ servers using 16-port switches, and should require a higher level compound graph as the server size increases in future containers. On the contrary, the family of network structures proposed in this paper is always only a level-1 regular compound graph, regardless how many servers a container accommodates. The multi-level compound graph, however, incurs imbalanced traffic at different levels of links; hence, reducing the resulting aggregate bottleneck throughput. More specifically, those links that are interconnecting building clusters potentially carry higher traffic than links attached to switches, and high-level links always carry much more flows than low-level links in Dcell, FiCoon, and BCN. That is, DCube does not suffer from such disadvantages; hence, it exhibits better performance than these related proposals in terms of the link congestion and aggregate bottleneck throughput.

BCube is the first dedicated structure for containerized data centers using more than two ports, typically 4, hence the requirement of a large number of links and switches. DCube significantly reduces the number of required wires and switches compared to BCube; hence, the construction cost, energy-consumption, and cabling complexity are largely reduced. In addition, DCube achieves a higher speedup than BCube for one-to-one and one-to-several traffic patterns. A challenge that arises here is the fact that DCube cannot achieve the same aggregate bottleneck throughput (ABT) as BCube, which employs more ports for each server and switches for routing. In fact, the lower ABT of Dcube is the tradeoff of less number of links and switches. As shown in Section 6, this can be addressed by some techniques at the application layer, such as the locality-aware task placement.

Additionally, many interconnection networks have been proposed in parallel computing, such as Mesh, Torus, Hypercube, Fat Tree, Butterfly, de Bruijn digraph, and Kautz digraph. The Kautz digraph has the smallest diameter among all of existing non-trivial digraphs under the same configurations of network size and maximum node out-degree [21]. However, compared to the hierarchical network structures for data centers, such as DCube, the Kautz digraph cannot not achieve a relative small network diameter. Existing interconnection networks cannot be utilized to tackle the technical problem we proposed in this paper. For example, a 2-ary Kautz digraph requires 4 ports at each server since the in-degree and out-degree of a 2-ary kautz digraph is 2, while this paper considers the case that the commodity servers in today's data centers come with dual-port. Actually, literatures [8, 10] also report the similar observations about the introduction of existing interconnection networks in the field of data centers.

3. The DCube Structure

In this section, we begin with the construction of DCube, a family of server-centric structures for containerized data centers with dual-port servers. We then describe two representative designs of DCube, i.e., H-DCube and M-DCube, that emulate the hypercube and möbius cube, respectively.

3.1. DCube construction

DCube network is built with two kinds of devices: dual-port servers and n-port mini-switches. The basic building block, denoted by Cube, is simply n servers connecting to an n-port mini-switch. After arranging the n servers into k groups, the Cube is partitioned into k sub-blocks, denoted by Cube_0, Cube_1,...,Cube_{k-1}. Each sub-block is built with $m{=}n/k$ servers connecting to the n-port switch in the basic building block, as shown in Fig.2. A DCube network consists of k sub-networks, denoted by DCube_1, DCube_2,...,DCube_k, which share all of the mini-switches in the DCube network. Throughout this paper, we impose a limitation on the value of k such that $n \bmod k{=}0$.

For $0{\le}i{\le}k{-}1$, DCube_i is a compound graph of Cube_i and a hypercube-like graph. DCube_i is obtained by replacing each node of the hypercube-like graph with a copy of Cube_i and replacing each link of the hypercube-like graph with a link which connects two corresponding copies of Cube_i. In DCube_i, the topology properties of the hypercube-like graph are preserved at the cost of an additional link that is associated with each server in DCube_i. For each server in DCube_i, the first port is used to connect to the switch while the second port is used to interconnect with another server in a different copy of Cube_i. Although the construction of DCube requires that DCube_is adopt the homogeneous hypercube-like graph, the basic ideas also apply to the heterogeneous setting. That is, each DCube_i may use different hypercube-like graphs, such as the hypercube and its variants.

When constructing a DCube_i, a *constraint* that arises is that the node degree of the hypercube-like graph must be equal to the number of servers in Cube_i, so as to constitute a regular compound graph. Thus, this requires an m-dimensional *hypercube-like* graph, in which each node is assigned a unique address

$a_{m-1}\ldots a_1a_0$ from the vector space $Z_2{}^m$, where $m{=}n/k$. For $0{\le}i{\le}k{-}1$, we can infer that DCube$_i$ has $2^m\times m$ servers and 2^m switches; hence, DCube has $2^m\times m\times k{=}2^m\times n$ servers and 2^m switches. We can see that DCube can be uniquely defined by two parameters, n and k, and is characterized by DCube(n,k). For ease of presentation, we use the term DCube to represent DCube(n,k) throughout the rest of this paper.

We now present the construction of DCube(n,k) as follows. We number the k sub-networks from DCube$_0$ to DCube$_{k-1}$ and number all switches from 0 to $2^{n/k}{-}1$. Equivalently, we use an address $a_{m-1}\ldots a_1a_0$ from $Z_2{}^m$ to denote a switch. We can use a term u to number those servers that are connected to the same switch from 0 to $n{-}1$, and we can denote a server in DCube(n,k) using the form $\langle a_{m-1}\ldots a_1a_0,u\rangle$. The connection rule between servers using their second ports depends on the used m-dimensional hypercube-like graph. In this paper, we focus on the hypercube of diameter m and the 1-möbius cube of diameter $\lceil(m+1)/2\rceil$ [22]. The resulting structures are characterized by H-DCube and M-DCube, respectively. The basic ideas also apply to other hypercube-like graphs with a similar diameter as that of the 1-möbius cube, such as the 0-möbius cube, Twisted cube [23], Flip MCube [24], and Fastcube [25].

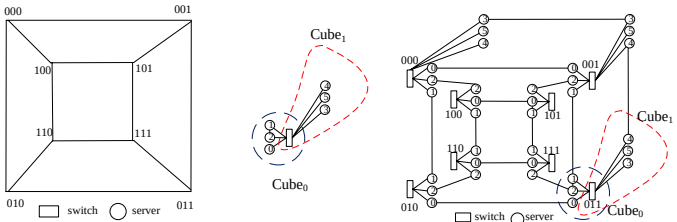
Before presenting the construction approach for the H-DCube and M-DCube, we first introduce notations and definitions used throughout this paper.

1. Let e_j denote the m-dimensional binary vector with only the j^{th} dimension equals to 1, where j is the index of e_j.
2. Let E_j denote the m-dimensional binary vector with 1 in dimensions x_j through x_0, where j is the index of e_j.
3. Given two m-dimensional binary vectors, $+$ denotes the modulo-2 addition for the corresponding elements.

3.2. H-DCube

In an m-dimensional hypercube, denoted by $H(m)$, two nodes, $x_{m-1}\ldots x_1x_0$ and $y_{m-1}\ldots y_1y_0$, are called the mutual j^{th} neighbors if their addresses differ by only the j^{th} vector component. That is $y_{m-1}\ldots y_1y_0{=}x_{m-1}\ldots x_1x_0{+}e_j$, where $0{\le}j{\le}m{-}1$. The node degree and network diameter of $H(m)$ are well known to be m. In an H-DCube(n,k), any server $\langle a_{m-1}\ldots a_j\ldots a_0,u\rangle$ is interconnected with another server $\langle a_{m-1}\ldots\overline{a_j}\ldots b_0,u\rangle$ using their second ports, where $j{=}u$ mod m. This simple connection rule guarantees the desired structure of an H-DCube network consisting of k sub-networks H-DCube$_i$ for $0{\le}i{\le}k{-}1$. We now discuss the correctness of such connection rule as in the following.

Only m servers and the unique switch in a basic building block falls into a sub-block Cube$_i$ if the sequence number u of those servers falls into the range of $[i{\times}m,(i{+}1){\times}m)$, where $0{\le}i{\le}k{-}1$. A sub-network, H-DCube$_i$, is a compound graph made by interconnecting a given number of copies of Cube$_i$ by means of $H(m)$, and is obtained by the following operations. Firstly, any node $\langle a_{m-1}\ldots a_j\ldots a_0\rangle$ and its j^{th} neighbor node $\langle a_{m-1}\ldots\overline{a_j}\ldots b_0\rangle$ in $H(m)$ are replaced by two copies of Cube$_i$. Secondly, the link from node $\langle a_{m-1}\ldots a_j\ldots a_0\rangle$ to its j^{th} neighbor node in $H(m)$ is replaced by a remote link between servers $\langle a_{m-1}\ldots a_j\ldots a_0,u\rangle$ and $\langle a_{m-1}\ldots\overline{a_j}\ldots b_0,u\rangle$,



(a) Regular graph G (b) Regular graph G1 (c) Compound graph G(G1)

Figure 2: DCube with $n=6$ and $k=2$, which employs a hypercube.

where $u=i\times m{+}j$. It is easy to see that only one link is connected to the second port of each server and that method is equivalent to the aforementioned connection rule. We can infer that all k sub-networks can be constructed in the same way, and they share all of the 2^m switches. Thus, the connection rule can guarantee the desired structure of an H-DCube network.

Fig.2 plots an H-DCube with $n{=}6$ and $k{=}2$, which consists of 8 basic building blocks, each with 6 servers and one switch. Note that the entire structure of each of the three basic building blocks 000, 001, and 011 are plotted, while the structures of the other basic building blocks are only partially plotted. All devices form two sub-networks, H-DCube$_0$ and H-DCube$_1$, each is a compound graph of $Cube_i$ and a 3-dimensional hypercube. The servers, whose sequence numbers are less than 3, belong to H-DCube$_0$, while others belong to H-DCube$_1$. Fig.2 shows entire and partial structures of H-DCube$_0$ and H-DCube$_1$, respectively. Clearly, H-DCube$_0$ and H-DCube$_1$ share all of the switches.

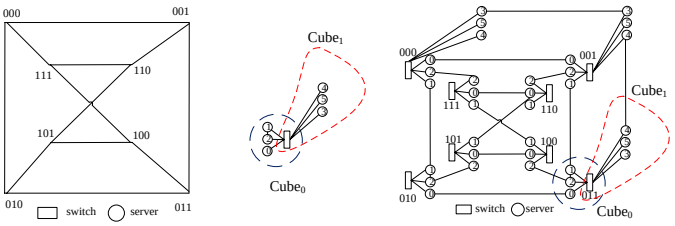
3.3. M-DCube

An m-dimensional möbius cube is such an undirected graph: its node set is the same as that of an m-dimensional hypercube; any node $X{=}x_{m-1}\ldots x_1x_0$ connects to m other nodes Y_j $(0{\le}j{\le}m{-}1)$, where Y_j satisfies one of the following equations:

$$Y_i = \begin{cases} x_{m-1}\ldots x_{j+1}\overline{x_j}x_{j-1}\ldots x_0, & \text{if } x_{j+1}=0 \\ x_{m-1}\ldots x_{j+1}\overline{x_jx_{j-1}\ldots x_0}, & \text{if } x_{j+1}=1 \end{cases} \quad (1)$$

According to the above definition, a node X connects to its j^{th} neighbor $Y_j{=}X+e_j$ that differs in bit x_j if $x_{j+1}{=}0$ and to $Y_j{=}X+E_j$ if $x_{j+1}{=}1$. The connection between X and Y_{m-1} has x_m as undefined. Here, x_m is either equal to 1 or 0, resulting in slightly different network topologies. This paper assumes that $x_m{=}1$: the resulting network is called the 1-möbius cube [22]. The node degree and network diameter of the m-dimensional 1-möbius cube are m and $\lceil(m+1)/2\rceil$, respectively.

In an M-DCube(n,k), all $2^m\times n$ servers and 2^m switches are first grouped into 2^m basic building blocks, each of which consists of n servers connecting to one switch using their first ports. For any server $\langle a_{m-1}\ldots a_{j+1}a_ja_{j-1}\ldots a_0,u\rangle$, we connect it to a server $\langle a_{m-1}\ldots a_{j+1}\overline{a_j}a_{j-1}\ldots a_0,u\rangle$ if $a_{j+1}{=}0$ or $\langle a_{m-1}\ldots a_{j+1}\overline{a_ja_{j-1}\ldots a_0},u\rangle$ if $a_{j+1}{=}1$ via their second ports, where $j{=}u$ mod m. This connection rule guarantees the desired structure of an M-DCube network consisting of k sub-networks M-DCube$_i$ for $0{\le}i{\le}k{-}1$. We omit the discussion about the correctness of this connection rule since the proof is very similar to that discussed in Section 3.2. Fig.3 shows an M-DCube



(a) Regular graph G (b) Regular graph G1 (c) Compound graph G(G1)

Figure 3: DCube with $n = 6$ and $k = 2$, which employs a 1-möbius cube.

network with n=6 and k=2, which consists of 8 basic building blocks, each with 6 servers and one switch. The servers, whose sequence numbers are less than 3, belong to M-DCube$_0$ and others are associated with M-DCube$_1$.

In summary, we can support 2048 servers in DCube(8,1) using 8-port switches and 4096 servers in DCube(16,2) using 16-port switches for both D-DCube and M-DCube. Another possible way of using 16-port switches is to construct DCube(16,1) with 1048576 servers, which is too large for containerized data centers. Moreover, the network diameter and expected routing path length are also relatively higher than that of DCube(16,2). Inspired by such fact, we prefer to construct DCube(n,k) as k>1 interconnected sub-networks when the number of ports on each switch exceeds an upper bound, for example 8.

4. Routing for one-to-one and one-to several traffic patterns

One-to-one traffic is the basic traffic pattern and good one-to-one support also results in good several-to-one and all-to-one support. In this section, we start with the single-path routing scheme for one-to-one traffic pattern, which only needs local decisions to identify a path or the next hop for any pair of servers in DCube. We then study the parallel multi-paths for one-to-one traffic pattern. Finally, we analyze the one-to-several traffic support properties of DCube.

4.1. Single-path routing in DCube

For two servers, A and B, we use $h(A,B)$ to denote the hamming distance between the two switches that are connecting the two servers, respectively, which is the number of different digits in their address arrays. It is clear that the maximum hamming distance between two switches in a DCube(n,k) is m=n/k. In this paper, two servers are neighbors if they connect to the same switch or if they directly connect to each other. The distance between two neighboring servers is one. Additionally, two switches are neighbors if there exists at least one pair of directly connected servers, each belonging to one of the two switches. Actually, the construction rules of H-DCube and M-DCube ensure that two neighboring switches have k pairs of such connecting servers, and each belongs to one sub-network. For example, two switches, 000 and 001, are neighbors since two servers, $\langle 000,0\rangle$ and $\langle 000,3\rangle$, directly connect to two servers, $\langle 001,0\rangle$ and $\langle 001,3\rangle$, respectively, as shown in Fig.2.

Based on such facts, we design two routing algorithms, H-DCubeRouting and M-DCubeRouting, as shown in Algorithms 1 and 3 respectively, to find a single path for any server pair.

Algorithm 1 H-DCubeRouting(A,B)

Require: A=$\langle a_{m-1}\dots a_0,u_a\rangle$ and B=$\langle b_{m-1}\dots b_0,u_b\rangle$
1: $path(A,B)=\{A,\}$;
2: *symbols* is a permutation of **Expansion-hypercube**(A,B);
3: $Pswitch=\langle a_{m-1}\dots a_0\rangle$ and $Cswitch=\langle a_{m-1}\dots a_0\rangle$;
4: **while** *symbols* not empty **do**
5: Let e_i denote the leftmost term in *symbols*;
6: $Cswitch=Cswitch+e_i$ and $u=\lfloor u_a/m\rfloor\times m+i$;
7: append $\langle Pswitch,u\rangle$ and $\langle Cswitch,u\rangle$ to $path(A,B)$;
8: remove e_i from *symbols* and $Pswitch=Cswitch$;
9: append server B to $path(A,B)$;
10: return $path(A,B)$;

Expansion-hypercube(A,B)
1: $terms=\{\}$;
2: **for** i=$m-1$ to 0 **do**
3: **if** $A[i]\neq B[i]$ **then**
4: $\{A[i]=a_i;\ B[i]=b_i.\}$
5: append e_i to *terms*;
6: return *terms*;

4.1.1. Single-path routing in H-DCube

In H-DCubeRouting, we assume that A=$\langle a_{m-1}\dots a_0,u_a\rangle$ and B=$\langle b_{m-1}\dots b_0,u_b\rangle$ are the source and destination servers, respectively. We first find a sequence of switches by correcting one digit of the previous switch, so as to produce a switch path from the source switch $a_{m-1}\dots a_0$ to the destination switch $b_{m-1}\dots b_0$. To make two adjacent switches in the switch path be neighbors, we have to choose one from the k pairs of connecting servers, each is connected to one of the adjacent switches.

A natural way of selecting the pair of connected servers belonging to the same sub-network, H-DCube$_i$ (i=$\lfloor u_a/m\rfloor$), is shown in Algorithm 1. Generally, another pair of connecting servers is also desirable if the two servers belong to the same sub-network, H-DCube$_i$ (i=$\lfloor u_b/m\rfloor$), as the destination server B. These efforts ensure that all intermediate servers in a routing path belong to the same sub-network, so as to ensure the load balance of each server under a uniform traffic model. The switches in the resulting path of Algorithm 1 can be uniquely determined by the identifiers of servers and hence are omitted from the path.

From H-DCubeRouting, we obtain the following theorem.

Theorem 1. *The diameter of an H-DCube(n,k) is* $2\times m+1$*, where* m=n/k.

PROOF. In an H-DCube(n,k), the shortest path between any two servers traverses, at most, m+1 switches, including the source switch, the destination switch, and other $m-1$ intermediate switches.

For any intermediate switch, there exists a one-hop packet transmission from the server receiving a packet to another server, which will forward the packet to its neighboring server in the next switch along with the switch path. For the source switch, there also exists a one-hop packet transmission if the source server cannot directly forward a packet to a server in the next switch. For the destination switch, a one-hop packet transmission is also necessary if the server receiving a packet is not the destination server. In addition, the total length of these m inter-

Algorithm 2 Expansion-mobius(A)

Require: A=$a_{m-1}\cdots a_0$ is a m-dimensional vector over $\{0,1\}$; $A[i]$=a_i

```
1: symbols={} and index=m−1;
2: while index < 0 do
3:    if index == 0 then
4:       if A[index] == 1 then
5:          append E_0 to symbols;
6:          index = index − 1;
7:    else
8:       if A[index]==0 then
9:          index==index−1;
10:      else
11:         if A[index]A[index − 1] == 10 then
12:            append e_index to symbols;
13:         if A[index]A[index − 1] == 11 then
14:            append E_index to symbols;
15:            A = a_{m−1} ··· ā_{index−2} ··· a_0;
16:         index = index − 2;
17: return symbols;
```

switch sub-paths between any adjacent switches in the shortest path is m. Thus, Theorem is 1 proven.

4.1.2. Single-path routing in M-DCube

We first discuss the expansion techniques of a vector, which are fundamental to our detailed discussion on the routing of M-DCube. The set R=$\{e_j, E_j | 0 \le j \le m-1\}$ forms a redundant basis for Z_2^m. Any vector X in Z_2^m can be expanded by R in the form:

$$X = \sum_{j=1}^{m-1} (\alpha_j e_j + \beta_j E_j), \quad (2)$$

with each $\alpha_j \in \{0,1\}$ and $\beta_j \in \{0,1\}$.

Definition 2. *For a vector X, the set of e_j and E_j with non-zero coefficients in Equation 2, denoted as $E(X)$, is called an expansion of the vector X. Any $t \in E(X)$ is a term of this expansion of X. The weight of an expansion $E(X)$ is called $W(X)$ and is equal to the cardinality of $E(X)$.*

There can be more than one expansion of a vector due to the use of a redundant basis. Thus, an expansion with minimal weight is referred to as the minimal expansion of X. Algorithm 2 shows a simple procedure for finding the minimal expansion for any vector. In each round, the algorithm first generates a sub-vector starting from the bit position *index* to the rightmost bit position of the vector X. If the sub-vector is 1, a term E_0 is added into the *symbols* set. If the sub-vector is 0, the algorithm is terminated. If the leftmost bit of the sub-vector is 0, the algorithm decreases the *index* by one and executes the next round. If the leftmost two bits of the sub-vector are 10, a term e_{index} is appended to the *symbols* set. Otherwise, a term E_{index} is added into the *symbols* set, and the vector X is updated by $X + E_{index}$ since the term E_{index} complements all bits from the position *index* to the rightmost position of X. The algorithm then carries out the next round after decreasing the *index* by two.

For a source server A=$\langle a_{m-1}a_{m-2}\cdots a_0, u_a\rangle$ and a destination server B=$\langle b_{m-1}b_{m-2}\cdots b_0, u_b\rangle$ in M-DCube(n,k), we define A+B as the vector obtained by the mod 2 sum of the switch addresses $a_{m-1}a_{m-2}\cdots a_0$ and $b_{m-1}b_{m-2}\cdots b_0$. To generate the shortest path between A and B, we first derive a switch path from the source switch $a_{m-1}a_{m-2}\cdots a_0$ to the destination switch $b_{m-1}b_{m-2}\cdots b_0$. We then find a pair of servers to connect two adjacent switches indirectly. Actually, the switch path between any pair of switches in M-DCube(n,k) is equivalent to the path between two corresponding nodes in the m-dimensional mö bius cube.

For any switch, e_i or E_i denotes its immediate neighbor along dimension i. For this reason, we refer to e_i or E_i as a routing symbol. To form a switch path, a sequence of routing symbols should be applied to the source switch. The minimal expansion $E(A+B)$, achieved by Algorithm 2, cannot be directly used to produce the switch path due to the following challenging issue. According to the definition of a 1-mö bius cube, given any node, only one of e_i and E_i can be the routing symbol along the i^{th} dimension, where $0 \le i \le m-1$. Consequently, a routing symbol in the minimal expansion does not always correspond to an edge in the 1-mö bius cube, and hence may be inapplicable to the current node. A natural way to deal with this issue is to replace any inapplicable routing symbol with an equivalent routing sequence obtained from Theorem 2.

Theorem 2. *Given a node $A = a_{m-1}a_{m-2}\cdots a_0$:*

1. *if e_i is inapplicable to the node A, it can be replaced by an equivalent routing sequence, E_iE_{i-1} or $E_{i-1}E_i$, which is applicable to A.*
2. *if E_i is inapplicable to the node A, it can be replaced by an equivalent routing sequence, e_iE_{i-1} or $E_{i-1}e_i$, which is applicable to A.*

PROOF. It is clear that e_i=$E_i + E_{i-1}$ and that E_i=$e_i + E_{i-1}$. Assume that e_i is inapplicable to node A. This implies that a_{i+1}=1; hence, E_i is applicable to node A. If a_i=1, then E_{i-1} is applicable to node A; thus, $E_{i-1}E_i$ is applicable to node A. Here, E_iE_{i-1} is inapplicable to node A since traversal along edge E_i from node A makes a_i become 0. If a_i=0, E_{i-1} is inapplicable to node A, but the application of E_i complements the bit a_i. Now E_{i-1} is applicable to node A+E_i, making E_iE_{i-1} be applicable to node A.

Assume that E_i is inapplicable to node A. This implies that a_{i+1}=0; thus, e_i is applicable to node A. If a_i=1, then E_{i-1} is applicable to node A; thus, $E_{i-1}e_i$ is applicable to node A since traversal along edge E_{i-1} from node A does not complement the bit a_{i+1}. If a_i=0, the application of e_i complement bit a_i. Now E_{i-1} is applicable to node A+e_i, making e_iE_{i-1} be applicable to A. Thus, Theorem 2 is proven.

According to the aforementioned strategies, we design M-DCubeRouting, as shown in Algorithm 3, to find a path from a source server A to a destination server B. The algorithm begins with achieving the minimal expansion of A+B by invoking Algorithm 2. It then calls the exact-routing algorithm to derive

Algorithm 3 M-DCubeRouting(A,B)

Require: A=$\langle a_{m-1}\dots a_0,u_a\rangle$ and B=$\langle b_{m-1}\dots b_0,u_b\rangle$;
1: *symbols*=**Expansion-mobius**(A+B);
2: $path(A,B)=\{A,\}$;
3: **Exactrouting**($a_{m-1}\dots a_0$,*symbols*);
4: append server B to $path(A,B)$;

Exactrouting(S,*symbols*)

Require: S denotes a current switch in the shortest path;
1: **while** *symbols* is not empty **do**
2: Let t denote the leftmost term in *symbols*;
3: **if** t is applicable to S. **then**
4: Let t' denote the rightmost applicable term to S in *symbols*; {The rightmost applicable term can be the leftmost applicable term.}
5: u=$\lfloor u_a/m\rfloor\times m+i$, where i is the index of t'=e_i or t'=E_i;
6: append $\langle S,u\rangle$ and $\langle S+t',u\rangle$ to $path(A,B)$;
7: remove t' from *symbols* and **Exact-routing**(S+t',*symbols*);
8: **else**
9: **if** The term t is in the form e_i **then**
10: replace e_i with E_iE_{i-1} if a_i=0 or $E_{i-1}E_i$ otherwise;
11: **else**
12: replace E_i with e_iE_{i-1} if a_i=0 or $E_{i-1}e_i$ otherwise;
13: **Exactrouting**(S,*symbols*);

a sequence of routing symbols, which can be successfully applied to the source switch so as to establish a switch path to the destination switch. In each round, the exact-routing algorithm ranks all terms in the *symbols* set in descending order according to the index of each term and then examines the leftmost term. If the leftmost term t is inapplicable to the current switch S, it is replaced by the equivalent routing sequence defined in Theorem 2. If the leftmost term is applicable to the current switch S, the rightmost applicable term t' will be applied first and then it updates the current switch S and *symbols*. This strategy can avoid the appearance of the worst result as in the following. If the leftmost applicable term is E_j, the application of it will make makes all next applicable terms become inapplicable.

After deriving the shortest switch path between the source and destination servers, we need to choose one from k pairs of the connecting servers between any adjacent switch S and $S+t'$ so as to make them be neighbors in M-DCube(n,k). As shown in Fig.3, two switches, 000 and 111, are neighbors since servers, $\langle 000,2\rangle$ and $\langle 000,5\rangle$, are directly connected to servers, $\langle 111,2\rangle$ and $\langle 111,5\rangle$, respectively. It is natural to choose the pair of servers which belong to the same sub-network, M-DCube$_i$ (i=$\lfloor u_a/m\rfloor$), as the source server A. Servers $\langle S,u\rangle$ and $\langle S$+$t',u\rangle$ append to the routing path, where u=$\lfloor u_a/m\rfloor\times m+i$ and i denotes the index of t'. Actually, another pair of connecting servers is also desirable if they belong to the same sub-network, M-DCube$_i$ (i=$\lfloor u_b/m\rfloor$), as the destination server B.

From M-DCubeRouting, we obtain the following theorem.

Theorem 3. *The diameter of an M-DCube(n,k) is* $2\times\lceil(m+1)/2\rceil+1$*, where* m=n/k.

PROOF. Given any two servers, A and B, in an m-dimensional 1-möbius cube, the weight of minimal expansion $E(A$+$B)$ is at most $\lceil m/2\rceil$ since no two terms have adjacent indices, according to Algorithm 2. The leftmost inapplicable term t_i in the minimal expansion is then replaced by a routing sequence with a length of 2. This strategy ensures that no other inapplicable terms exist in the routing path after replacing t_i since E_{I-1} complements the bit a_{j+1} for any inapplicable term t_j, where j<i. Thus, Algorithm 3 ensures that the diameter of an m-dimensional 1-möbius cube is $\lceil(m+1)/2\rceil$, and hence the shortest path between any two servers in an M-DCube(n,k) traverses, at most, $\lceil(m+1)/2\rceil+1$ switches. As mentioned in the proof of Theorem 1, there is a one-hop transmission within each switch and between two adjacent switches in the routing path. Thus, Theorem 3 is proven.

4.2. Multi-paths for one-to-one traffic

Traditionally, two parallel paths between a source server and a destination server exist if the intermediate servers and switches on one path do not appear on the other. It is clear that there exists, at most, two parallel paths for any pair of servers under this strict definition due to the dual-port on each server. In this paper, *two paths are called parallel* if the intermediate switches on one path are not involved in the other path, except for the beginning and ending switches. In addition, two neighboring switches possess k pairs of directly connected servers. To maximize the utility of such an advantage, a switch path can be utilized as k *weak parallel paths*, which share the same set of switches but have different intermediate servers. Such parallel and weak parallel paths between any pair of servers can be further utilized to improve the transmission rate or to enhance the transmission reliability for one-to-one traffic with Multipath TCP [26, 27]. In addition, Multipath TCP can explore such multiple paths to tackle traffic congestion, leading to higher network utilization.

The following theorem specifies the exact number of parallel paths and weak parallel paths between any two servers in a DCube(n,k).

4.2.1. H-DCube

Theorem 4. *There are m parallel and n weak parallel paths between any two servers in an H-DCube(n,k), where m=n/k.*

The m parallel paths between any two servers in an H-DCube(n,k) can be simplified to m parallel switch paths since all inter-switch sub-paths of two adjacent switches in the m paths are disjoint. Thus, we can show the correctness of Theorem 4 by constructing such m parallel switch paths. Algorithm 1 produces a shortest switch path from A to B using any permutation of the minimal expansion $E(A$+$B)$, which contains e_j for some $0\le j\le m-1$ but not E_j for any $0\le j\le m-1$. In the minimal expansion of A+B, $W(A$+$B)$ distinct terms form an initial routing sequence, resulting in $W(A$+$B)!$ minimal routing sequences. Theorem 5 indicates that only $W(A$+$B)$ parallel switch paths from A to B can be generated.

Theorem 5. *Let the minimal expansion $E(A$+$B)$ generated by the* Expansion-hypercube *be $t_1,t_2,\dots,t_{W(A+B)}$. Algorithm 1 generates $W(A$+$B)$ parallel switch paths from A to B using permutations as follows. The i^{th} permutation for $0\le i<W(A$+$B)$ is denoted as $p_1,p_2,\dots,p_{W(A+B)}$, where p_j=$t_{(j+i) \bmod W(A+B)}$ for $1\le j\le W(A$+$B)$.*

PROOF. Actually, these permutations are obtained by moving each term of the initial routing sequence to the mod left by i, for $0 \leq i < W(A+B)$, under the following two constraints. Firstly, any pair of such permutations differ in the addition of leftmost j terms for $1 \leq j \leq W(A+B)$. Secondly, the addition of any leftmost j terms is different from that of any left j' terms where $j \neq j'$ for each of these permutations. Thus, this pattern ensures that the resulting $W(A+B)$ paths are disjoint except for the source and destination switches; thus, the $W(A+B)$ parallel switch paths are produced. For example, e_1e_0 and e_0e_1 are two minimal routing sequences, resulting in two parallel switch paths between servers $\langle 000,0\rangle$ and $\langle 011,0\rangle$, as shown in Fig.2. The resulting two parallel paths are $\{\langle 000,0\rangle, \langle 000,1\rangle, \langle 010,1\rangle, \langle 010,0\rangle, \langle 011,0\rangle\}$ and $\{\langle 000,0\rangle, \langle 001,0\rangle, \langle 001,1\rangle, \langle 011,1\rangle, \langle 011,0\rangle\}$, respectively.

Assume that t'_h belongs to $\{e_{m-1}, \cdots, e_1, e_0\}$ but does not appear in the minimal expansion $E(A+B)$. We achieve a new routing sequence by appending t'_h to the leftmost and rightmost terms of one existing routing sequence. This further results in a switch path, which is parallel with the $W(A+B)$ switch paths generated in Theorem 5. For example, $e_2e_1e_0e_2$ or $e_2e_0e_1e_2$ produces another path, which is parallel with the two paths generated by e_1e_0 and e_0e_1, for two servers, $\langle 000,0\rangle$ and $\langle 011,0\rangle$, as shown in Fig.2. The path generated by $e_2e_1e_0e_2$ is $\{\langle 000,0\rangle, \langle 000,2\rangle, \langle 100,2\rangle, \langle 100,1\rangle, \langle 110,1\rangle, \langle 110,0\rangle, \langle 111,0\rangle, \langle 111,2\rangle, \langle 011,2\rangle, \langle 011,0\rangle\}$. The path resulting from $e_2e_0e_1e_2$ is $\{\langle 000,0\rangle, \langle 000,2\rangle, \langle 100,2\rangle, \langle 100,0\rangle, \langle 101,0\rangle, \langle 110,1\rangle, \langle 111,1\rangle, \langle 111,2\rangle, \langle 011,2\rangle, \langle 011,0\rangle\}$.

Consider that $m-W(A+B)$ terms in $\{e_{m-1}, \cdots, e_1, e_0\}$ do not appear in the minimal expansion $E(A+B)$. Thus, we can derive $m-W(A+B)$ parallel switch paths with lengths of $W(A+B)+2$ using the same approach as mentioned above. Thus, we can construct m parallel switch paths between two servers,, A and B in an H-DCube(n,k). If we produce another switch path between A and B using a new routing sequence, at least one switch in the new path has to have appeared on existing switch paths. The root cause for this is that the leftmost and rightmost terms of the new routing sequence must have to have appeared at the beginning and/or end of the m previous routing sequences. Thus, the largest number of parallel switch paths between any pair of servers in an H-DCube(n,k) must be m.

After discussing the parallel switch paths between any pair of servers in an H-DCube(n,k), we further consider the subpath between any adjacent switches in these paths. Algorithm 1 selects one pair of connecting servers for each pair of neighboring switches in any switch path so as to realize a path including servers and switches. However, k weak parallel paths can be produced based on a given switch path between two servers after updating line 6 with $u=j\times m+i$ for $0 \leq j \leq k-1$. That is, each one of the m parallel paths between two servers can be realized as k weak parallel paths. For this reason, we can induce that there are $m \times k = n$ weak parallel paths between any two servers. Thus, Theorem 4 is proven.

4.2.2. M-DCube

Theorem 6. *There are m parallel and n weak parallel paths between any two servers in an M-DCube(n,k), where $m=n/k$.*

We use a similar approach to show the correctness of Theorem 6 by constructing such parallel paths and weak parallel paths. Given two servers, A and B, in an M-DCube(n,k), *Expansion-mobius* generates a minimal expansion of $A+B$ in the scenario of an m-dimensional möbius cube. It is worth noticing that some terms in the minimal expansion may be inapplicable to the current switch and should thus be replaced by an equivalent routing sequence as defined in Theorem 2. To address this issue, *M-DCubeRouting* generates an initial routing sequence by invoking *exact-routing* with the minimal expansion $E(A+B)$ as input. Assume that the initial routing sequence is denoted as $t_1, t_2, ..., t_l$, where $l \geq W(A+B)$.

M-DCubeRouting can further generate some parallel switch paths between servers A and B by using permutations of the initial routing sequence. Any permutation on the initial routing sequence forms a new routing sequence. For this reason, one can conclude that there exists $l!$ routing sequences, but only the following ones can produce l parallel switch paths. Assume that the i^{th} permutation for $0 \leq i < l$ is denoted as $p_1, p_2, ..., p_l$, where $p_j = t_{(j+i) \bmod l}$ for $1 \leq j \leq l$.

The first challenging issue we face is the fact that terms in each permutation of the initial routing sequence may be inapplicable to the current switch and should be revised according to Theorem 2 so as to generate an applicable routing sequence. The resulting applicable routing sequence can generate a new switch path, which will be parallel with existing switch paths. For example, the initial routing sequence for a shortest path from server $A=\langle a_2a_1a_0=000,0\rangle$ to server $B=\langle 100,0\rangle$ in Fig.3 is E_2E1, which is applicable. The first permutation of E_2E_1 is it. The second permutation of E_2E1 is E_1E_2, in which E_1 is inapplicable to $A=000$ since $a_2 = 0$. As a result, E_1E_2 should be replaced by $e_1E_0E_2$.

Besides the l parallel switch paths, we will show how generate other $m-l$ parallel switch paths between any servers, A and B, in an M-DCube(n,k). Let t'_m denote any term, which belongs to $\{e_{m-1}, \cdots, e_1, e_0\}$, but is not the leftmost term in the routing sequences defined by the above permutation operation. We achieve a new routing sequence by appending t'_m to the leftmost and rightmost terms of one existing routing sequence, which further results in a new switch path. This switch path is parallel to the l switch paths generated by the aforementioned l permutations of the initial routing sequence. For example, the routing sequence, $e_0E_2E_1e_0$, is achieved by appending $t'_m = e_0$ to the beginning and end of E_2E_1. It then generates a new path from $A=\langle 000,0\rangle$ to $B=\langle 100,0\rangle$ in Fig.3. That is, $\{\langle 000,0\rangle, \langle 001,0\rangle, \langle 001,2\rangle, \langle 110,2\rangle, \langle 110,1\rangle, \langle 101,1\rangle, \langle 101,0\rangle, \langle 100,0\rangle\}$.

The second challenging issue we face is the fact that appending a t'_m term to the leftmost and rightmost terms of an existing routing sequence, for example the initial routing sequence $t_1, t_2, ..., t_l$, does not necessarily result in a parallel path in a general scenario. Actually, if t'_m appears at the end of an existing routing sequence, then the last two switches, including the destination switch, in the new switch path must have

occurred in the related path.

To produce a parallel path based on $t'_m t_1, t_2, ..., t_l t'_m$, we need to find a t''_m from $t_1, t_2, ..., t_l$, which does not appear at the end of those routing sequences defined by the above permutation operation. If there exists such a t''_m, we move it to the end of $t'_m t_1, t_2, ..., t_l t'_m$ and optimize it so as to generate an applicable and parallel path. Otherwise, we need to replace a given term in $t_1, t_2, ..., t_l$ with an equivalent routing sequence, as defined in Theorem 2, which contains such a t''_m. We then move the t''_m to the end of the resulting routing sequence and optimize it so as to generate an applicable and parallel path. Based on those techniques, we can derive other $m-l$ switch paths which are parallel with the l switch paths generated by the above permutation operation.

As discussed in the proof of Theorem 4, there are k pairs of connected servers between any two neighboring switches. For this reason, M-DCubeRouting produces k weak parallel paths, based on one switch path between two servers, by updating line 5 with $u=j\times m+i$ for $0\leq j\leq k-1$. Thus, one can induce that the m parallel paths between two servers can be realized as $m\times k=n$ weak parallel paths; hence, Theorem 6 is proven.

4.3. Speedup for one-to-several traffic

A complete graph consisting of a set of servers can speed up data replications in distributed file systems. We show that edge-disjoint complete graphs with $m+1$ servers can be efficiently constructed in a DCube(n,k).

Theorem 7. *In a DCube(n,k), a server $\langle src, u_s\rangle$ and a set of m servers can form an edge-disjoint complete graph, where each of the m servers connects to a different neighboring switch of the switch src.*

In the case of H-DCube(n,k), the i^{th} neighbor of switch *src* is defined as $src+e_i$ for $0\leq i<m$. Assume that $src+e_i$ and $src+e_j$ are two neighboring switches of the switch *src*, where $i\neq j$. A switch path with a length of two from $src+e_i$ to $src+e_j$ can be generated by a routing sequence, $e_j e_i$, and is denoted as $\{src+e_i,\ src+e_i+e_j,\ src+e_i+e_j+e_i=src+e_j\}$. It is easy to see that two different pairs of e_i and e_j cannot produce the same result of e_i+e_j, where $i\neq j$. Consequently, this pattern ensures that the switch paths among a switch *src* and its m neighbors are edge-disjoint.

In the case of M-DCube(n,k), the i^{th} neighbor of switch *src* is defined as $src+t_i$ for $0\leq i<m$, where t_i is e_i or E_i according to the definition of an m-dimensional möbius cube. Assume that $src+t_i$ and $src+t_j$ are two neighbors of the switch *src*, where $i\neq j$. A switch path from $src+t_i$ to $src+t_j$ can be generated by an initial routing sequence, $t_j t_i$. In special cases, the resulting switch path is applicable and denoted as $\{src+t_i,\ src+t_i+t_j,\ src+t_i+t_j+t_i=src+t_j\}$. In general cases, each term in the initial routing sequence might be inapplicable. To generate an applicable switch path, each inapplicable term should be replaced with an equivalent routing sequence consisting of two terms, as defined in Theorem 2. For this reason, we can induce that the applicable and shortest switch path from $src+t_i$ to $src+t_j$ is, at most, four hops. In addition, we can see that two different pairs of t_i and t_j cannot produce the same result of t_i+t_j, where $i\neq j$. That is, the switch paths among a switch *src* and its m neighbors are edge-disjoint.

From the above construction approaches, we can see that the resulting complete graph is only two switch hops and is, at most, four switch-hops in H-DCube(n,k) and M-DCube(n,k), respectively.

Given an edge-disjoint complete graph formed by the source switch *src* and its m neighboring switches, each edge in the complete graph should be replaced by a pair of connected servers since two adjacent switches are not connected directly. It is worth noticing that there are k pairs of connecting servers for each edge in the complete graph. We only choose the pair of servers, which are located in the same sub-network DCube$_i$ as the source server, $\langle src, u_s\rangle$. The motivation is to separate the traffic in k complete graphs for any server $\langle src, u_s\rangle$ into the corresponding sub-networks. This operation ensures that the whole paths among the source server and m selected servers, including switches and servers, are still edge-disjoint.

We further show how to choose the m servers, denoted as d_j for $0\leq j\leq m-1$, for the source server $\langle src, u_s\rangle$. For the j^{th} neighboring switch of the switch *src*, we choose d_j from n servers connecting to that switch, such that d_j locates in the same sub-network DCube$_i$ as the source server, where $i=\lfloor u_s/m\rfloor$. In this way, each d_j has m choices since a switch allocates m of n servers to each sub-network DCube$_i$. In this paper, we just randomly select d_j from m choices and, we will study other selection methods of d_j in our future work.

So far, we have demonstrated that a complete graph can be formed by the source server and a set of m selected servers in the sub-network DCube$_i$. Actually, we can generate k such complete graphs for any server $\langle src, u_s\rangle$ by the following approach since a DCube(n,k) consists of k sub-networks DCube$_i$s. Assume that the selected server for d_j in the sub-network DCube$_i$, is denoted as $\langle src+t_j, u_j\rangle$ for $0\leq j\leq m-1$. The corresponding server $\langle src+t_j, u_j+k\times m\rangle$ and the source server generate a new complete graph in the k^{th} sub-network DCube$_k$, where $k\neq i$.

A file on distributed file systems can be divided into chunks, and each chunk is typically replicated to three chunk servers. The source and the chunk servers establish a pipeline to reduce the replication time, as discussed in literature [10]. The edge-disjoint complete graph that is built into DCube works well for chunk replication speedup. When one writes a chunk to r ($r\leq m+1$) chunk servers, it sends $1/r$ of the chunk to each chunk server. Meanwhile, every chunk server distributes its copy to the other $r-1$ servers by using the edge-disjoint edges. Consequently, this will be r times faster than the pipeline model.

5. Analysis and Evaluation

In this section, we conduct simulations to evaluate several basic properties of DCube.They include the speedup for one-to-one and one-to-several traffic patterns, aggregate bottleneck throughput based on measurements of real-world data center traffic from [28], the cost, the power consumption, and the cabling complexity. We also compare the performance of DCube

with not only Fat-tree but also DCell, HCN, Fat-tree and BCube, which are three particularly enlightening server-centric data-center structures. In the evaluation setting, the number of servers in DCube ranges from 2048 to 12288, and the capacity of each link is 1Gb/s. The setting matches the scale and configurations of a typical containerized data center.

To ensure a fair comparison, such network structures interconnect the same number of servers, denoted as N, with switches each of n ports. They, however, differ in the number of server ports, the number of switches, the number of cables, and the interconnection rules. DCell, HCN, and BCube are recursively defined structures, whose levels are denoted by k_1, k_2 and k_3, respectively, where $k_1 \leq k_3$.

5.1. Speedup for one-to-one and one-to-several traffic

For the one-to-one and one-to-several traffic patterns, we show the speedup as compared with other networking structure. We first summarize the throughput of such two traffic patterns under different networking structures in Table 1.

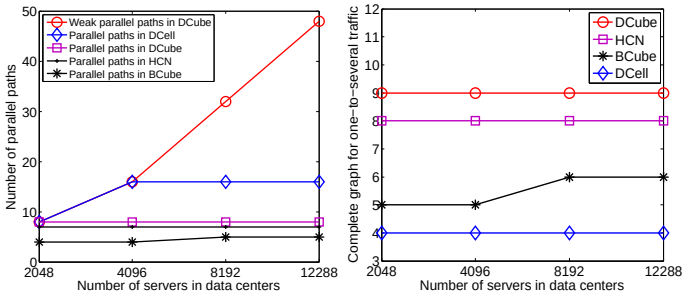
For any server pair, A and B, DCube(n,k) provides $\lceil n/k \rceil$ parallel and n weak parallel paths for them. These properties not only speedup the one-to-one traffic, but also offer graceful degradation of performance. We can see from Fig.4(a) that DCube offers more parallel paths for any pair of servers than HCN and BCube, as the network size increases from 2048 to 4096, 8192, and 12288. Although DCell possesses more parallel paths for any server pair than DCube, DCube delivers large number of weak parallel paths and hence achieves better speedup performance for one-to-one traffic.

For any source server, we show that the complete graph can significantly speedup the one-to-several traffic. Assume that server A=$\langle 000,2\rangle$ in Fig.2 replicates 20G data to two servers, B=$\langle 010,2\rangle$ and C=$\langle 001,2\rangle$. With the complete graph approach, the data is split into two parts and sent to both B and C, respectively. B and C then exchange their data with each other. On the contrary, with the pipeline approach, A sends the data to B, and B sends the data to C. The complete graph can achieve about 2 times the speedup compared to the pipeline approach. In general, when a source deliver a chunk to r servers in the same complete graph, it sends $1/r$ of the chunk to each of the server. Meanwhile, every chunk server distributes its copy to the other $r-1$ servers using the disjoint edges in the complete graph. This will be r times faster than the pipeline model. This implies that DCube executes speedup well when it comes to the one-to-several traffic pattern.

Recall that DCube can offer the largest complete graph of size $n/k+1$. The largest cardinality of a complete graph in DCell and BCube is k_1+2 and k_3+2, as proved in [29]. Fig.4(b) plots the largest cardinality, $r+1$, of a complete graph for one-to-several traffic in DCube, DCell, HCN and BCube. We can see that DCube always outperforms others when the data center

Table 1: Comparison of M-DCube, DCell, HCN, Fat-tree, and BCube

Throughput	M-DCube	DCell	HCN	Fat Tree	BCube
One to one	2	k_1+1	2	1	k_3+1
One to several	$\frac{n}{k}+1$	k_1+2	n	1	k_3+2
All-to-all	$\frac{N}{1/3\times n/k}$	$\frac{N}{2^{k_1}}$	$\frac{N}{2/3\times 2^{k_2}}$	N	N



(a) (b)

Figure 4: (a) Number of parallel paths and weak parallel paths. (b) Order of the complete graph for one-to-several traffic.

size increases from 2048 to 4096, 8192, and 12288. This means that DCube results in a higher speedup than DCell, HCN and BCube for one-to-several traffic pattern.

5.2. Aggregate bottleneck throughput

Aggregate bottleneck throughput (ABT) is defined as the number of flows times the throughput of the bottleneck flow under the all-to-all traffic pattern [10]. ABT of Fat-Tree and BCube are N since they achieve the nonblock communication between any pair of servers. ABT of H-DCube and M-DCube are proved in Theorem 8 and Lemma 1. ABT of Dcell is N/k_1, as proved in [29], while that of HCN is $\frac{N}{2/3\times 2^{k_2}}$.

Fat-Tree and BCube outperform M-DCube in terms of the ABT under all-to-all traffic pattern. Such a result is not surprising since M-DCube utilizes much less switches, links, and ports than the other two structures. We argue that the benefits of H-DCube and M-DCube outweigh such a downside since it is unlikely that all servers frequently participate in the all-to-all communication. Moreover, M-DCube achieves higher ABT than DCell and HCN since $n/(3k)$ is typically less than $2^{k_2+1}/3$ and 2^{k_1}. Besides the above theoretical analysis, we also conduct simulations, based on real-world data center traffic from [28], to evaluate the ABT of three networking structures. We can see from Fig.5 that DCube achieves higher ABT than DCell and HCN, irrespective the data center size.

Theorem 8. *For a H-DCube(n,k) network, its ABT under the all-to-all traffic pattern is $\frac{N}{2/3\times n/k}$, where n is the number of ports per switch and N is the number of servers.*

PROOF. The diameter of H-DCube(n,k) is $\frac{2\times n}{k}+1$, as we have proved in Theorem 1. Accordingly, we can derive that the expected distance, i.e., the average path length, approximates to $\frac{n}{k}$. The links in H-DCube(n,k) consist of two parts. Firstly, each of N server connects to a switch using its first port and thus generates one link. The number of such links is N. Secondly, each of N server connects with another server using its second port and thus generates one link. The number of such links is $N/2$. The total number of links in H-DCube(n,k) is $3N/2$.

The number of flows carried in one link is $f_{num}=\frac{N(N-1)n/k}{3N/2}$, where $N(N-1)$ is the total number of flows. The throughput one flow receives is thus $\frac{1}{f_{num}}$, assuming that the bandwidth of a link is one. The aggregate bottleneck throughput is therefore $N(N-1)\frac{1}{f_{num}}=\frac{N}{2/3\times n/k}$. Thus, Theorem 8 is approved.

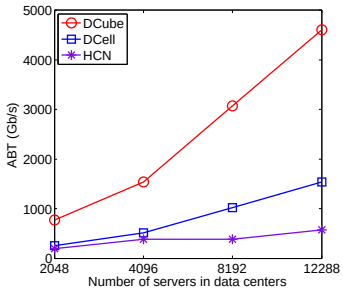


Figure 5: ABT of different structures under number of servers in data centers.

Lemma 1. *For a M-DCube(n,k) network, its ABT under the all-to-all traffic pattern is $\frac{N}{1/3\times n/k}$, where n is the number of ports per switch and N is the number of servers.*

PROOF. The diameter of M-DCube(n,k) is $2\times\lceil(n/k+1)/2\rceil+1$, as we have proved in Theorem 3. Accordingly, we can derive that the expected distance approximates to $\frac{n}{2k}$. The proving process of Lemma 1 is similar to that of Theorem 8.

5.3. Qualification of cost and cabling complexity

We first consider five networking structures for a container with 2048 servers and many 8-port switches. Such structures are constructed as follows. DCube structure is a DCube$(8,1)$. DCell is a partial DCell$(8,2)$ with 28 DCell$(8,1)$s. HCN structure is a partial HCN$(8,3)$ with 4 full HCN$(8,2)$s. BCube structure is a partial BCube$_3$ with 4 full BCube$_2$s, where n=8. Fat-tree structure has five layers of switches, with layers 0 to 3 having 512 switches per-layer and layer-4 having 256 switches [10]. In this setting, DCube, DCell, HCN, BCube and Fat-tree employ 256, 252, 256, 1280 and 2304 8-port switches, while the number of NIC ports on each server are 2, 3, 4, 4, and 1, respectively. Note that a 8-port switch costs about $40 and consumes near 4.5W of power. For one-port, two-port, and 4-port NICs, their costs are about $5, 1$0, and $20, while the power consumptions are about 5W, 7.5W, and 10W, respectively.

We then consider DCube, DCell, HCN and BCube for a container with 4096, 8192, and 12288 servers, respectively. In this setting, DCube structures are DCube$(16,2)$ using 16-port switches, DCube$(32,4)$ using 32-port switches, and DCube$(48,6)$ using 48-port switches, respectively. DCell structures are partial DCell(16,2)s with 15, 30 and 45 DCell(16,1)s using 16-port switches, respectively. HCN structures using 8 ports switches are a HCN(8,3), a partial HCN(8,4) with 2 HCN(8,3)s, and a partial HCN(8,4) with 3 HCN(8,3)s, respectively. BCube structures with 8-port switches are a full BCube$_3$, a partial BCube$_4$ with 2 full BCube$_3$s, and a partial BCube$_4$ with 3 full BCube$_3$s, respectively. Note that BCubes with 4096, 8192, and 12288 servers may have different structures. For example, BCube structures with 16-port switches are a full BCube$_2$, a partial BCube$_3$ with 2 full BCube$_2$s, and a partial BCube$_3$ with 3 full BCube$_2$s, respectively. Note that a 16-port switch costs about $150 and consumes 21W of power, a 32-port switch costs about $400 and consumes 75W of power, while a 48-port switch costs about $600 and consumes 103W of power.

Fig.6 summarizes the number of wires and switches, the cost of switches and NICs, and the power consumption of switches and NICs in such five networking structures with 2048, 4096, 8192, and 12288 servers, respectively. When DCell, HCN and BCube structures are incomplete under some aforementioned settings, we need to build partial structures. For a partial BCube$_i$, Guo et. al suggest that we build the needed BCube$_{i-1}$s and then connect the BCube$_{i-1}$s using full layer$_i$ switches [10].

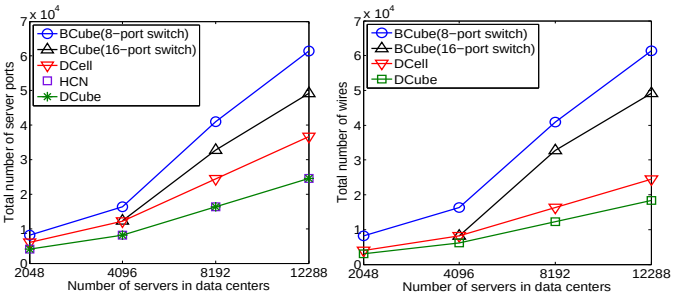
		Cost(k$)		Power(kw)		Wires	Switchs
		Switch	NIC	Switch	NIC	No.	No.
2048	Fat-tree	92	10	10	10	10240	2304
	BCube	51	41	5.8	20	8192	1280 (8-port)
	HCN	10	20	1.2	15	3068	256 (8-port)
	DCell(2016)	10	40	1.1	18	4032	252 (8-port)
	DCube(8,1)	10	20	1.2	15	3072	256 (8-port)
4096	BCube	81	82	9.3	40	16384	2048 (8-port)
	HCN	20.5	82	2.3	41	6140	512 (8-port)
	BCube	115	61	16	37	12288	768 (16-port)
	DCell(4080)	38	81.6	5.4	41	8160	255 (16-port)
	DCube(16,2)	38	41	5.4	31	6144	256 (16-port)
8192	BCube	324	205	37.1	100	40960	8192 (8-port)
	HCN	41	164	4.6	82	12290	1024 (8-port)
	BCube	845	164	118	82	32768	5632 (16-port)
	DCell(8160)	76.5	163	10.7	82	16320	510 (16-port)
	DCube(32,4)	102	82	19.2	61	12288	256 (32-port)
12288	BCube	571	307.5	65	120	61440	14336 (8-port)
	HCN	61	246	7	123	18428	1536 (8-port)
	BCube	960	246	134	123	49152	6400 (16-port)
	DCell(12240)	114.7	245	16	122	24480	765 (16-port)
	DCube(48,6)	153.6	123	26	92	18432	256 (48-port)

Figure 6: Measuring networking structures under different sizes of data centers.

More precisely, Fig.7(a) and Fig.7(b) demonstrate that both the number of NIC ports on servers and that of links of DCube are always considerably less than that of BCube and DCell, irrespective the data center size. Thus, DCube largely reduces the cabling complexity, especially for large containerized data centers. Note that DCube and HCN achieve the similar performance in terms of the two metrics since they interconnect dual-port servers. Moreover, DCube has other advantages due to the less number of wires and switches. As shown in Fig.6 and Fig.8, DCube outperforms BCube, DCell and HCN in terms of the entire cost and power consumption of switches and NICs, irrespective the data center size. Additionally, we find that the BCube structure with 16-port switches results in more cost and power consumption compared to the BCube structure with 8-port switches under the same number of servers.

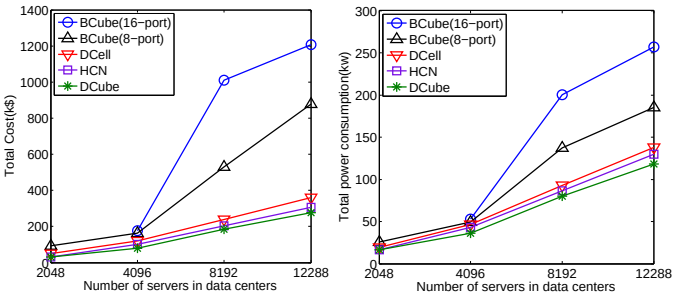
5.4. Summary

DCube significantly reduces the required wires and switches compared to Fat-tree and BCube, because of this, it largely reduces the cabling complexity compared to other structures, especially for large containerized data centers. On the other hand, DCube considerably outperforms BCube in terms of the entire cost and power consumption, irrespective the data center size. Besides these benefits, the maximum throughput of DCube is twice that of Fat-tree, but less than that of DCell and BCube whose number of levels is typically larger than 2. For the one-to-several traffic pattern, DCube achieves a higher speedup than Fat-tree, DCell and BCube. Additionally, DCube achieve the higher ABT than DCell and offers graceful degradation.



(a) Total number of NIC ports on servers. (b) Total number of links.

Figure 7: The sum of NIC ports and links vary as the increasing servers in data centers.



(a) Total cost of switches and NICs. (b) Power consumed by switches and NICs.

Figure 8: The cost and power consumption of switches and NICs under different settings.

6. Discussions

6.1. Locality-aware task placement

Although the proposed network structures have many advantages, such as easy wiring and low cost, they may not be able to achieve the same ABT as BCube under the all-to-all traffic pattern. Recall that BCube offers many NIC ports for each server and utilizes large numbers of switches so as to achieve a higher ABT; hence, significantly bringing about more cost and power consumption. In fact, the relatively lower ABT of DCube compared to BCube results from the lower number of links and switches, resulting in a longer average routing path; this is the tradeoff of other measurements. Fortunately, this issue can be addressed by some techniques at the application layer since a server is likely to communicate with a small subset of other servers for typical applications in common data centers.

Therefore, a locality-aware approach can be used for placing those tasks onto servers in DCube. That is, those tasks with intensive data exchanges can be first placed onto servers that connect to the same switch. If those tasks need some more servers, they may reserve the several nearest basic building blocks. There are only a few switch-hops, maybe even one, between those building blocks. It is easy to see that DCube is usually sufficient enough to contain hundreds of servers where the number of switch hops is, at most, two. Therefore, the locality-aware mechanism can largely save network bandwidth by avoiding unnecessary remote data communications.

6.2. Extension to more server interfaces

The basic idea of this paper is to design a family of network structures for containerized data centers using constant number of embedded NIC interfaces. Although we assume that all servers are equipped with two built-in NIC interfaces, the design methodologies of DCube can be easily extended to involve any constant number, denoted as q, of server interfaces. In fact, servers with four embedded NIC interfaces have been made available due to the rapid innovation on server hardware.

Given any server with q interfaces, it can contribute $q-1$ interfaces for interconnecting other basic building blocks after reserving one port for connecting to switch. Intuitively, a server with q ports can be treated as a set of $q-1$ dual-port servers. In this way, we can extend DCube to embrace any constant number of server ports. Additionally, each server can contribute its $q-1$ interfaces as a virtual interface by the port trunking [18], e.g., three 1Gbps interfaces can be bundled into a virtual interface at 3Gbps. In this way, the link capacity between two servers can be significantly improved. There may exist many specific ways for interconnecting servers with a constant node degree of more than 2. We leave this research issue to be the focus of our future work.

6.3. Impact of server routing

Server routing is a challenging issue faced by server-centric networking structures for data centers. In a DCube structure, servers connecting to other basic building blocks have the responsibility of forwarding packets. Although DCube can use software-based or FPGA-based forwarding schemes, just as initial server-centric structures do, it incurs additional forwarding delay.

To address the latency due to server routing using software-based or FPGA-based forwarding schemes, Guo et. al proposed ServerSwitch [14]. It integrates the programmable commodity switching chip into a built-in NIC for packet forwarding and leverage the CPU and RAM of server for in-network packet processing and storage. Since ServerSwitch is easily configured and it can forward packets at line-rate, we can easily re-configure ServerSwitch to forward self-defined packets for DCube without any hardware re-designing. Recently, more hardware-based commodity devices have been implemented to support server routing by reducing the forwarding latency.

7. Conclusion

We present DCube, a family of low-cost and robust network structures for containerized data centers with dual-port servers and commodity switches. It offers high degrees of regularity and symmetry, which very well conform to containerized data centers. These benefits are obtained at the cost of only associating with each server only two links, regardless of the network size. This largely reduces the cabling complexity of the containerized data center. In addition, DCube achieves a higher speedup than BCube for one-to-one and one-to-several traffic patterns. Moreover, DCube exhibits a graceful performance degradation as the server and switch failure rate increases. Although this paper first considers that all servers are equipped with two built-in NIC ports, the design methodologies of DCube can be easily extended to any number of NIC ports after minimal modifications.

References

[1] S. Ghemawat, H. Gobioff, and S.-T. Leung, "The google file system," in *Proc. SOSP*, Bolton Landing, NY, USA, 2003, pp. 29–43.

[2] Y. Zhang and N. Ansari, "On architecture design, congestion notification, tcp incast and power consumption in data centers," *IEEE Communications Surveys and Tutorials*, vol. 15, no. 1, pp. 39–64, 2013.

[3] M. A. Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," in *Proc. SIGCOMM*, Seattle, Washington, USA, 2008.

[4] A. Greenberg, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. A. Maltz, and P. P. and, "Vl2: A scalable and flexible data center network," in *Proc. SIGCOMM*, Barcelona, Spain, 2009.

[5] R. Mysore, A. Pamboris, and N. Farrington, "Portland: A scalable fault-tolerant layer 2 data center network fabric," in *Proc. SIGCOMM*, Barcelona, Spain, 2009.

[6] J. Kim, W. J. Dally, S. Scott, and D. Abts, "Technology-driven, highly-scalable dragonfly topology," in *Proc. 35th ISCA*, Beijing, China, 2008, pp. 77–88.

[7] B. Arimilli, R. Arimilli, V. Chung, S. Clark, W. Denzel, B. Drerup, T. Hoefler, J. Joyner, J. Lewis, J. Li, N. Ni, and R. Rajamony, "The PERCS High-Performance Interconnect," in *Proc. 18th Symposium on High-Performance Interconnects*, Mountain View, CA, USA, Aug. 2010.

[8] C. Guo, H. Wu, K. Tan, L. Shi, Y. Zhang, and S. Lu, "Dcell: A scalable and fault-tolerant network structure for data centers," in *Proc. SIGCOMM*, Seattle, Washington, USA, 2008.

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

[10] C. Guo, G. Lu, D. Li, H. Wu, X. Zhang, Y. Shi, C. Tian, Y. Zhang, and S. Lu, "Bcube: A high performance, server-centric network architecture for modular data centers," in *Proc. SIGCOMM*, Barcelona, Spain, 2009.

[11] D. Guo, T. Chen, D. Li, Y. Liu, X. Liu, and G. Chen, "Bcn: Expansible network structures for data centers using hierarchical compound graphs," in *Proc. 30th IEEE International Conference on Computer Communications (INFOCOM)*, Shanghai, China, 2011, pp. 61–65.

[12] S. Han, K. Jang, K. Park, and S. Moon, "Packetshader: a gpu-accelerated software router," in *Proc. ACM SIGCOMM*, New Delhi, India, 2010.

[13] G. Lu, Y. Shi, C. Guo, and Y. Zhang, "Cafe: A configurable packet forwarding engine for data," in *Proc. PRESTO*, Barcelona, Spain, 2009.

[14] G. Lu, C. Guo, Y. Li, and Z. Zhou, "Serverswitch: A programmable and high performance platform for data center networks," in *Proc. 8th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, Boston, MA, USA, 2011, pp. 15–28.

[15] J. Hamilton, "An architecture for modular data centers," in *Proc. 3rd CIDR*, Asilomar, CA, USA, 2007, pp. 306–313.

[16] M. Waldrop, "Data center in a box," *Scientific American*, 2007.

[17] D. Li, M. Xu, H. Zhao, and X. Fu, "Building mega data center from heterogeneous containers," in *Proc. IEEE ICNP*, Vancouver, BC Canada, 2011.

[18] H. Wu, G. Lu, D. Li, C. Guo, and Y. Zhang, "Mdcube: A high performance network structure for modular data center interconnection," in *Proc. CoNEXT*, Rome, Italy, 2009.

[19] D. P. Agrawal, C. Chen, and J. R. Burke, "Hybrid graph-based networks for multiprocessing," *Telecommunication system*, vol. 10, pp. 107–134, 1998.

[20] D. Guo, T. Chen, D. Li, M. Li, Y. Liu, and G. Chen, "Expansible and cost-effective network structures for data centers using dual-port servers," *IEEE Transactions on Computers*, vol. 62, no. 7, pp. 1303–1317, 2013.

[21] D. Guo, J. Wu, Y. Liu, H. Jin, H. Chen, and T. Chen, "Quasi-kautz digraphs for peer-to-peer networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 22, no. 6, pp. 1042–1055, 2011.

[22] P. Cull and S. M. Larson, "The mobius cubes," *IEEE Transactions on Computers*, vol. 44, no. 5, pp. 647–659, 1995.

[23] J.-H. Park, H.-C. Kim, and H.-S. Lim, "Many-to-many disjoint path covers in hypercube-like interconnection networks with faulty elements," *IEEE Transactions on Parallel Distributed Systems*, vol. 17, no. 3, pp. 227–240, 2006.

[24] M. Singhvi and K. Ghose, "The mcube: A symmetrical cube based network with twisted links," in *Proc. 9th International Parallel Processing Symposium*, Washington, DC, USA, 1995, pp. 11–16.

[25] S. Ghozati and T. Smires, "The fastcube: a variation on hypercube topology with lower diameter," *Computers and Electrical Engineering*, vol. 29, no. 1, pp. 151–171, 2003.

[26] D. Wischik, C. Raiciu, A. Greenhalgh, and M. Handley, "Implementation and evaluation of congestion control for multipath tcp," in *Proc. 7th USENIX NSDI*, Boston, MA, USA, 2011.

[27] C. Raiciu, S. Barre, C. Pluntke, A. Greenhalgh, D. Wischik, and M. Handley, "Improving datacenter performance and robustness with multipath tcp," in *Proc. ACM SIGCOMM*, Toronto, Ontario, Canada, 2011.

[28] T. Benson, A. Akella, and D. A. Maltz, "Network traffic characteristics of data centers in the wild," in *ACM SIGCOMM Conference on Internet Measurement Conference*, Melbourne, Australia, 2010.

[29] D. Li, C. Guo, H. Wu, K. Tan, Y. Zhang, S. Lu, and J. Wu, "Scalable and cost-effective interconnection of data-center servers using dual server ports," *IEEE/ACM Transactions on Networking*, vol. 19, no. 1, pp. 102–114, 2011.