IEICE TRANS. COMMUN., VOL.E103–B, NO.9 SEPTEMBER 2020

PAPER

# P-Cube: A New Two-Layer Topology for Data Center Networks Exploiting Dual-Port Servers

**Moeen AL-MAKHLAFI**[†a)], ***Nonmember***, **Huaxi GU**[†b)], ***Member***, **Xiaoshan YU**[†], ***and*** **Yunfeng LU**[†], ***Nonmembers***

**SUMMARY** Connecting a large number of servers with high bandwidth links is one of the most crucial and challenging tasks that the Data Center Network (DCN) must fulfill. DCN faces a lot of difficulties like the effective exploitation of DC components that, if highlighted, can aid in constructing high performance, scalable, reliable, and cost-effective DCN. In this paper, we investigate the server-centric structure. We observe that current DCs use servers that mostly come with dual ports. Effective exploitation of the ports of interest for building the topology structure can help in realizing the potentialities of reducing expensive topology. Our new network topology, named "Parallel Cubes" (PCube), is a duplicate defined structure that utilizes the ports in the servers and mini-switches to form a highly effective, scalable, and efficient network structure. P-Cube provides high performance in network latency and throughput and fault tolerance. Additionally, P-Cube is highly scalable to encompass hundreds of thousands of servers with a low stable diameter and high bisection width. We design a routing algorithm for P-Cube network that utilizes the P-Cube structure to strike a balance among the numerous links in the network. Finally, numerical results are provided to show that our proposed topology is a promising structure as it outperforms other topologies and it is superior to Fat-tree, BCube and DCell by approximately 24%, 16%, 8% respectively in terms of network throughput and latency. Moreover, P-Cube extremely outperforms Fat-tree, and BCube structures in terms of total cost, complexity of cabling and power consumption.

***key words:*** *data center networking, P-Cube topology, server-centric network, routing algorithm*

## 1. Introduction

Cloud computing has gained considerable scholarly interest in many fields. Data Center is the infrastructure of cloud computing and is the main concern for today's companies like Google, Facebook, Yahoo, Microsoft, Amazon, and other companies. Its operation is fundamental to providing many online applications such as web mail, search, gaming, and infrastructure services e.g., Dryad [2], MapReduce [3], GFS [1], BigTable [4], and HDFS [5]. In addition, those companies own data centers and host their data and their users' data. At the same time, as the need for data centers is increasing, companies face many problems in dealing with increasing needs such as material difficulties and financial burden. Interconnecting the physical DC components is one of the most crucial and challenging tasks for the Data Center networks (DCNs). Given a largely grown data center with corresponding increased servers, DCNs must be able to interconnect these large number of servers, which could be as large as millions of servers, by providing adequate bandwidth to guarantee the quality of cloud services, the demands for high reliability, flexibility and power density to ensure that different applications are running effectively. Thus, the DCNs infrastructure must be developed in order to be capable of providing the network with all the needed requirements, which includes a large number of computing and storage resources as well as architectural design to link nodes in between them appropriately.

A traditional DCN is constructed in three-layers. In general, it contains edge, aggregation and core layer switches in a down-top manner. The internet is connected to the data center by uplinks coming out of the switches in the core layer. Given the ever-increasing demands that the traditional DCNs cannot achieve (e.g., high growth in Internet applications and services, high reliability and high power density). Also, it has several disadvantages such as high cost, limited bandwidth, and inflexibility.

Recent proposed studies, such as [6], [7] and [8] attempted to build a high scalability network, however, these networks may still suffer from the following issues: the network size of Fat-tree [6] is restricted to the number of ports in the switch. Moreover, one of the main drawbacks is the fault intolerance at the root switch or the primary node at the core layer, which could cause the entire tree to collapse. DCell [7], typically demands more ports for each server. It has been proven that it provides a high performance in mega data centers "big band"capacity growth, but when an incremental approach is being followed, huge inefficiencies were presented. For example, $DCell_0$, $DCell_1$, $DCell_2$ and $DCell_3$ will have a four-port switch that is connected to 4 servers, 20 servers, 420 servers (i.e. 21 times), and 176,820 (i.e. 421 times), respectively. The exponential growth in the required number of servers affected the practicality of the approach, as being applied for different enterprise data centers. Unfortunately, BCube [8] shares the same problems besides its incapability in supporting partial networks due to its topology structure. Its topology structure affects its ability to getting expanded through incremental approaches or small footprint nodes.

Exploiting commodity servers coming with dual-port and commodity switches with multi-ports can result in getting potential advantages that are multifaceted. The advantages include high performance, scalable, reliable, and cost-effective data center networks (DCNs). The aforementioned advantages make investigating the technical cases to

Manuscript received October 21, 2019.
Manuscript revised January 15, 2020.
Manuscript publicized March 3, 2020.

[†]The authors are with State Key Laboratory of Integrated Service Networks, Xidian University, Xi'an China.
a) E-mail: moeen@stu.xidian.edu.cn
b) E-mail: hxgu@xidian.edu.cn

DOI: 10.1587/transcom.2019EBP3219

be crucial and vital to the research community.

In this paper, we propose P-Cube, a scalable network for containerized data centers, which exploits the dual-port servers and the low-cost commodity switches. P-Cube is constructed by many pods. Each pod contains two types of servers that differ in terms of network ports. To reduce the cost of construction, we employ half of the total servers as the dual-port servers (internal servers). The second half of the total servers are employed as multi-port servers (external servers) to provide a higher network capacity with high bisection bandwidth. Conversely, each pod contains two layers of the switches. On one hand, the first layer (internal switches) enables the connection between the types of servers. Specifically, there are $n$ internal switches, each of switch is endowed with $n$ ports for connecting $\frac{n}{2}$ internal and $\frac{n}{2}$ external servers within the same pod. On the other hand, the second layer (external switches) allows the connection among the external servers in different pods in order to improve the aggregate bottleneck throughput and provide high degrees of symmetry and regularity, which are paramount characteristics for the data center networks. In this network structure, we denote a level-k P-Cube as P-Cube$_k$ as it has $\frac{nk}{2}$ pods, where ($k > 0$), and $n$ is the number of an internal switch ports. The total number of servers is given by $N = \frac{n^3k}{2}$, and it increases doubly with P-Cube levels. For example, when $k = 8$ and $n = 48$, P-Cube can have as many as 442,368 servers. The diameter of P-Cube is stable and is equal to 7, in consequence, it can support the real-time requirements applications.

The rest of this paper is organized as follows. Section 2 presents the related works. Section 3 describes the construction and structure of P-Cube. Section 4 proposes our routing algorithm in P-Cube. Section 5 shows a discussion where we compare P-Cube and other topologies and evaluates P-Cube to verify our proposed methods. Finally, Sect. 6 concludes the paper.

## 2. Related Works

Recently many different DCN topologies have been developed in order to fit different criteria and purposes. These topologies can be classified roughly into two classes. The first class is switch-centric in which routing and networking are the main responsibilities of the switch e.g. Fat-tree [6], VL2 [10], FaceBook's, four-post [14], Google's Jupiter [15], Space Shuffle [16], and Diamond [11]. The second class is server-centric where routing schemes are realized more effectively by exploiting the amenability of servers in programming more than their counterparts (i.e. switches), such as ServerSwitch [9]. Note that ports on the server are used for networking and routing purposes. Typical topologies of server-centric include DCell [7], BCube [8], FiConn [12], BCN [17], DCube [18], Hyper-Flatnet [19], MDCube [20], BCCC [21], GBC3 [23], NovaCube [24], and DPillar [13].

The previous efforts proposed numerous DCNs, however, most of those topologies are not suitable for the container data centers that contain low-cost commodity switches and dual-port servers. To satisfy the requirements of such container data centers, we therefore present FiConn [12], DCell [7] and BCube [8] for large-scale data centers. On the other hand, there is a need for ameliorating the bandwidth bottleneck, as such, we consider the high-end cost switches at the top levels, i.e., Fat-tree [6] and Leaf-Spine [22] in order to achieve the desired objective.

Leaf-spine is a DCN topology that is generally composed of two-layer switches, viz., leaf switches, which are connected to the servers and spine switches, which are connected to the leaf switches. Leaf switches mesh into the spine to form an access layer, by which the network connection points are delivered for the servers.

Fat-tree network is constructed by using three layers of switches, these are edge, aggregation, and core. Fat-tree consists of $k$ pods each containing two layers (first and second layers) of $\frac{k}{2}$ switches; the first layer of switches (edge) has $k$ ports, $\frac{k}{2}$ of the ports are connected to $\frac{k}{2}$ servers, while the remaining $\frac{k}{2}$ ports are connected to the second layer of switches (aggregation). At the third layer of switches (core), there are $(\frac{k}{2})^2$ $k$-port core switches, where each core switch uses a port to connect a pod at the second layer. Therefore, Fat-tree supports $\frac{k^3}{4}$ of servers. Given a typical 48 port-switches can support 27,648 servers.

FiConn is a recursive structure consisting of many low-level FiConn's levels, which are used to create a high-level FiConn. FiConn$_0$ uses commodity switches associated with n-ports to connect to n-dual-port servers. The servers in FiConn have two ports. The first port is an active port that is used to connect the commodity switch while the second port is a backup port that can be used to expand the network. The backup ports are reserved and divided into two halves in terms of functionality. The first half is used to construct level1 connections with the other FiConn$_0$'s, while the second half is used to construct the higher-level connections. Typically, in FiConn$_{k-1}$ with $v$ backup ports, the FiConn$_k$ is built of $(\frac{2}{v} + 1)$ FiConn$_{k-1}$'s, and the total number of the servers is equal to $N_k = N_{k-1}(\frac{N_{k-1}}{2^k} + 1), k > 0$.

DCell is a recursive structure that consists of many levels of low-level DCell's, which are being used to construct a high-level DCell. DCell$_0$ use commodity switches that are accompanied by $n$-ports to connect $n$ servers. With $N$ servers in DCell$_k$, $N + 1$ DCell$_k$s are used to construct DCell$_{k+1}$. The $N$ servers in a DCell$_k$ are connected to the other $N$ DCell$_k$s.

BCube is a recursively defined structure like Dcell structure. It is specially designed for modular DCs. BCube's basic block is BCube$_0$ and it is the same as DCell$_0$. BCube uses commodity switches that come with $n$-ports to connect $n$ servers. Constructing BCube$_k$ requires using $n$ BCube$_{k-1}$'s, and $n^k$ switches with $n$-ports, where $n \leq 8$.

Finally, we will mention some disparities between our proposed structure and the previous structures in several aspects as follows: P-Cube utilizes the servers for networking and routing rather than switches that are being utilized by Fat-tree. In addition, Fat-tree has three layers of switches,

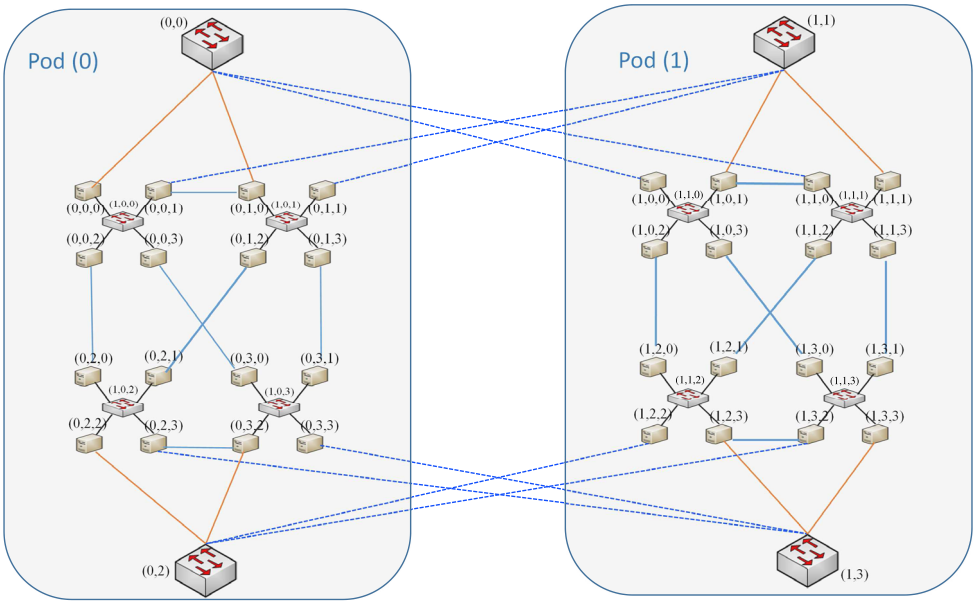



**Fig. 1** A P-Cube1 with n = 4. The P-Cube1 is composed of 2 pods, each pod is composed of 4 internal switches, each switch connected to 4 servers and 2 external switches.

but P-Cube has only two. Hence, Fat-tree uses the number of switches more than P-Cube. For $n$-port switches, let $N$ be the total number of the server being utilized. We can observe in Table 1 that Fat-tree utilizes $\frac{5N}{n}$ switches, while P-Cube utilizes $\frac{N}{n} + (nk), k > 0$ switches. Therefore, P-Cube reduces the cost of switches by approximately 70% compared with Fat-tree. Moreover, Fat-tree supports a number of servers that are restricted by the number of switch ports. While P-Cube extends to a huge number of servers and does not suffer from these restrictions.

Although FiConn is cheaper than the other structures, it has significantly low throughput compared to the other four structures. This is mainly because of the low bisection width of the incomplete structure. Typically, in FiConn and DCell, to connect around 4000 servers by using switches with 16-ports, we need at least a level-2 compound diagram. Hence, with an increased number of servers, we need to construct a higher level to make connections among the servers. Contrariwise, in P-Cube network whatever the increase in the number of servers, we just require a level-2 compound diagram to connect the servers. The traffic at levels-2 is more balanced than multi-level; as a result growing the output aggregate bottleneck throughput. More specifically, in FiConn the links in high-level that are used to interconnect the levels always load more flows than links in low-level that are used to connect with switches.

Although P-Cube has some similarities in design with BCube and DCell, there are some differences. First, the degree of nodes in $\text{BCube}_k$ and $\text{DCell}_k$ is $k+1$, while in P-Cube the degree of $\frac{N}{2}$ of nodes (i.e. internal servers) is two. As a result, exploiting existing dual-port on every server reduces the cost by eliminating the need to add Network Interface Cards (NICs) for $\frac{N}{2}$ servers. Second, P-Cube wiring costs less than DCell and BCube, because each $\frac{N}{2}$ nodes (internal server) uses only two ports. Third, routing in P-Cube outperforms DCell in its capability in making a balanced use of links at different levels.

## 3. Architecture

In this section, we first introduce our P-Cube physical structure, and then we describe how we address the servers and switches in the Pod.

### 3.1 P-Cube Physical Structure

P-Cube is a server-centric topology that utilizes servers with multi NIC ports and mini-switches. We construct our P-Cube network by using many pods, each pod has two types of mini-switches. The first type is called “internal switch”, where every switch have $n$ ports to connect $n$ servers in the same pod. The other type is called “external switch”, where every switch has $\frac{n^2 \times k}{4}$ ports, and this switch is used to connect the peer servers in the different pods, in which a level-$k$ P-Cube is denoted as $\text{P-Cube}_k$ since it has $\frac{n \times k}{2}$ pods, where $k > 0$ and $n$ is the number of internal switch ports. The example shown in Fig. 1 illustrates how P-Cube of different Pods is being constructed. The Pod is the building block used in building larger $\text{P-Cube}_k$. Each pod has $n$ of internal switches, and 2 of external switches ($n = 4$ for $\text{P-Cube}_1$ as in Fig. 1). Typically, $n$ is an even number more than 4 such as 4, 8, 16..etc.

The procedure of constructing P-Cube is shown in Algorithm 1, where it is divided into two parts. Part I constructs $n$ nodes and $n$ internal switches and connecting all the nodes to their designated switch. In addition, an organized manner it constructs $\frac{n \times k}{2}$ number of pods to build $\text{P-Cube}_k$, and building connections for all nodes in the pod. Part II constructs

**Algorithm 1** The procedure of constructing P-Cube network and building the connections.

```
    /*k stands for the level of P-Cube, n is the number of internal switch ports */
    Part-I/* Build Pods in P-Cube_k, and build internal connections */
 1: Build Pods (k,n)
 2: for (int p = 0; p < (n/2) * k; p++)
 3:     for (int i = 0; i < n; i++)
 4:         for (int j = 0; j < n; j++)
 5:             connect node ( j ) to switch ( i ) ;
 6:             if(i ≠ j)
 7:                 connect node (p, i, j) and (p, j, i);
 8:             if (((i < n/2) and (j ≥ n/2)) or ((i ≥ n/2) and (j < n/2)))
 9:                 The node (p, i, j) is an internal server.
10:             else
11:                 The node (p, i, j) is an external server.
12:             End if;
13:         End for;
14:     End for;
15: End for;
16: return;
    Part-II/* Build the external switches and the external connections */
17: Build P-Cube (k,n)
18: for (int No_p = 0; No_p < (n/2) * k; No_p++)
19:     for (int p = 0; p < No_p; p++)
20:         for(int i = 0; i < n; i++;)
21:             for(int j = 0; j < n; j++)
22:                 if(i < n/2)
23:                     int T = No_p mod (n/2)
24:                     if(j = T)
25:                       connect node(p, i, j) to switch (No_p, T);
26:                 else
27:                     int B = No_p mod (n/2) + (n/2)
28:                     if(j = B)
29:                       connect node(p, i, j) to switch (No_p, B);
30:                 End if;
31:             End for;
32:         End for;
33:     End for;
34: End for;
35: return;
```

$n \times k$ number of external switches and building all external connections among the external nodes to those switches in several pods.

There are two types of servers in terms of network ports. The first type is called internal servers, where there are $N/2$ servers with dual-port. The first port is connected to an internal switch and the second port is connected to another internal server in the same pod. The second type is called external servers, where there are $N/2$ servers with multi-ports depending on $k$, where one of the ports is connected to an internal switch in the same pod, another port is connected to another external server if and only if the server_id and switch_id are not equal (see Algorithm 1, Line 6), and the rest of the ports are connected with an external switch in several pods. The parameter $N$ denotes the total number of servers in P-Cube, which is equal to $\frac{k \times n^3}{2}$. Consequently, the total number of internal servers and the total number of external servers will be equal to $N/2$ for each of them.

### 3.2 Addressing

The internal switches are addressed by $(k, p, i)$, where $k$ denotes the level of the topology, $p$ denotes the pod number (in $[0, \frac{n \times k}{2}]$), and $i$ denotes the switch number (in $[0, n]$). Every pod has two external switches. The first one is at the top and the second one is at the bottom, in which they are addressed by $(p, T)$, $(p, B)$ respectively, where $T = p \ mod \ \frac{n}{2}$, and $B = T + \frac{n}{2}$.

The servers in P-Cube are addressed by $(p, i, j)$, where $j$ denotes the position of that server in that switch (in $[0, n]$ starting from left to right, top to down). Precisely, the internal servers are addressed by $(p, i<\frac{n}{2}, j \geq \frac{n}{2})$ *and* $(p, i \geq \frac{n}{2}, j < \frac{n}{2})$, while the external servers are addressed by $(p, i<\frac{n}{2}, j < \frac{n}{2})$ *and* $(p, i \geq \frac{n}{2}, j \geq \frac{n}{2})$ as shown in Algorithm 1 (Line 8-11).

Now we will focus on how we build the connections. In Algorithm 1 and Fig. 1 an example to illustrate the P-Cube interconnection rules, when $n$=4, P-Cube$_1$ has $\frac{n \times k}{2}$=2 pods. Every pod has 2 external switches, and 4 internal switches, where every switch is connected to 4 servers. Every pod has 16 servers, 8 of those servers are internal servers, which are connected to each other according to their own address, *connect node* $(p, i, j)$ *to* $(p, j, i)$, where $(i \neq j)$ (Line 6, 7). By referring to Fig. 1, the internal servers in $pod_0$ will be $(0, 0, 2)$, $(0, 0, 3)$, $(0, 1, 2)$, $(0, 1, 3)$, $(0, 2, 0)$, $(0, 2, 1)$, $(0, 3, 0)$, and $(0, 3, 1)$ (Line 8, 9). The remaining 8 servers are external servers(Line 10, 11), that are connected to external switches in P-Cube according to their own address and the external switch address, *connect node* $(p, i, j)$ *to switch* $(p, T)$ *or switch* $(p, B)$ (Line 25, 29), $if \ and \ only \ if \ j = T$ *or* $j = B$ (Line 24, 28), respectively, where $T = P \ mod \ \frac{n}{2}$, and $B = T + \frac{n}{2}$. By referring to Fig. 1, the external servers in $pod_0$ will be $(0, 0, 0)$, $(0, 0, 1)$, $(0, 1, 0)$, $(0, 1, 1)$, $(0, 2, 2)$, $(0, 2, 3)$, $(0, 3, 2)$, and $(0, 3, 3)$.

## 4. Routing in P-Cube

In this section, we propose our P-Cube Fault-tolerant Routing scheme (PFR) an optimal, routing solution that effectively exploits P-Cube structure and can effectively deal with various failures. We start to present a routing protocol without failures.

### 4.1 Routing without Failure (P-CubeRouting)

To improve the performance of P-Cube, we designed an efficient shortest-path routing algorithm, which can exploits the duplicate structure of P-Cube to forward the packet and balances the numerous links in the network. The designed routing algorithm, named "P-CubeRouting", is shown in Algorithm 2. P-CubeRouting comprises two types of traffic routing in order to route and transmit packets between any pairs of servers. The first type is intra-pod traffic, where the source and destination servers are in the same pod. The second type is inter-pod traffic, where the source and destination servers

are in different pods. Consequently, Algorithm 2 consists of three parts, which are marked as an internal switch, an external switch, and the *P-CubeRouting*. In *P-CubeRouting*, we assume that ($p_s$,$i_s$,$j_s$) and ($p_d$,$i_d$,$j_d$) are the source and destination addresses of a packet. (Line 18) in Algorithm 2 shows how the function *P-CubeRouting*() route the packets from any source ($p_s$,$i_s$,$j_s$) to any destination ($p_d$,$i_d$,$j_d$).

For the intra-pod traffic (Line 20), the source server will check the destination server address of the sending packet. If the source_server_id equals to the destination_switch_id (Line 21), then it will check whether the source_switch_id equals to the destination_server_id (Line 22), i.e. directly connected to the source server through NIC. If those two conditions are met, the packet will be sent to the destination server (Line 23). However, if the second condition is not met (Line 25), then the source server is directly connected to an intermediate server via NIC. Consequently, the packet will be sent to the intermediate server (Line 26). The intermediate server is also connected to the internal switch ($k$, $p_d$, $i_d$). Then, the intermediate server will forward the packet to the internal switch (Line 27), which then forwards the packet to the destination server(Line 29). For example, in Fig. 1, the shortest path between the two nodes [0,1,2] and [0,2,0], is ([0,1,2], [0,2,1], [0,2,0]).

Otherwise (Line 32), an intermediate link ($m1$, $m2$) will be established between the source and destination servers. The source server is connected to an internal switch ($k$, $p_s$, $i_s$). It will, therefore, forward the packet to the internal switch (Line 33). The internal switch will then forward the packet to an intermediate server $m1$ (Line 1-10), which will consequently send it to an intermediate server $m2$. Similarly, $m2$ is connected to an internal switch ($k$, $p_d$, $i_d$), whose function is to forward the packet to the destination server. For example, in Fig. 1, the shortest path between the two nodes [0,0,1] and [0,3,1] is ([0,0,1], [0,0,3], [0,3,0], [0,3,1]).

For the inter-pod traffic (Line 37), *P-CubeRouting* contains two situations according to the type of the source server. The first situation is when the source server ($p_s$, $i_s$, $j_s$) is an external server (Line 38), i.e. source server is directly connected to the external switch through NIC. Hence, it will forward the packet to an external switch (Line 39) whose function is forwarding the packet to a server in the same pod as the destination's (Line 11-17). Once the packet arrives at the same pod as its destination, we utilize the intra-pod traffic to forward the packet to the destination server as illustrated earlier (Line 41). As shown in the following example in Fig. 1 the shortest path between the two nodes [0,1,0] and [1,1,3] is ([0,1,0], [0,0], [1,1,0], [1,1,3]).

The second situation is when the source server ($p_s$, $i_s$, $j_s$) is an internal server (Line 42,45), i.e. the source server is connected to the internal switch. The internal switch will forward the packet to an external server (Line 1-10). Once the packet arrives at an external server, *P-CubeRouting* will treat it as in the first situation of inter-pod traffic until it reaches the destination (Line 44,48). As shown in the following example in Fig. 1 the shortest path between the two nodes [0,3,1] and [1,0,2] is ([0,3,1], [0,3,2], [0,2], [1,2,2], [1,2,0], [1,0,2]).

**Algorithm 2** P-CubeRouting.

```
    /*(p_s, i_s, j_s) source server
      (p_d, i_d, j_d) destination server
      where (p, i, j) denotes to (pod_id, switch_id, server_id)
       respectively, and k denotes the level of the topology */
    Part-I/* Routing the packets in the internal switch */
 1: Internal switch (p_s, i_s, j_s, p_d, i_d, j_d)
 2:     if (p_s==p_d)
 3:         if (i_s==i_d)
 4:             output port number = j_d ;
 5:         else
 6:             output port number = i_d ;
 7:     else
 8:         output port number = i_d ;
 9: send packet to the output port
10: end;
    Part-II/* Routing the packets in the external switch */
11: External switch (p_d, i_d)
12:     if (i_d < n/2)
13:         output port number = p_d×n/2+i_d ;
14:      else
15:         output port number =(p_d×n/2)+(i_d-n/2) ;
16: send packet to the output port
17: end;
    Part-III/* Routing the packets in a servers */
18: P-CubeRouting (p_s, i_s, j_s, p_d, i_d, j_d)
19: while packet not deliver to (p_d, i_d, j_d) do
20:     if (p_s==p_d)
21:         if (j_s==i_d)
22:             if (j_d==i_s)
23:                 send packet to node(p_d, i_d, j_d) ;
24:                 Break;
25:             else
26:                 Send packet to intermediate node (p_s,j_s,i_s);
27:                 send packet to internal switch (k, p_d, i_d);
28:                 Internal switch (p_s, j_s, i_s, p_d, i_d, j_d);
29:                 send packet to node (p_d, i_d, j_d) ;
30:                 Break;
31:             end if
32:         else
33:             send packet to internal switch (k, p_s, i_s);
34:             Internal switch (p_s, i_s, j_s, p_d, i_d, j_d);
35:             Go to line (18)
36:         end if
37:     else
38:         if (((i_s < n/2)and(j_s < n/2)) or ((i_s ≥ n/2)and(j_s ≥ n/2)))
39:             send packet to external switch (p, j);
40:             External switch (p_d, i_d);
41:             Go to line (18)
42:         else if(((i_s < n/2)and(i_d ≥ n/2))or((i_s ≥ n/2)and(i_d < n/2)))
43:             Send packet to the corresponding node;
44:             Go to line (18)
45:         else
46:             send packet to internal switch (k, p_s, i_s)
47:             Internal switch (p_s, i_s, j_s, p_d, i_d, j_d)
48:             Go to line (18)
49:         end if;
50:     end if;
51: end while;
52: return;
```

## 4.2 P-Cube Fault-Tolerant Routing (DFR)

Network failures affect network performance dramatically.

**Algorithm 3** PFR: P-Cube Fault-tolerant Routing.

/* $k$, $n$, int.sw, ext.sw and R denotes to the level of the network, the number of internal switch ports, internal switch, external switch and the chosen Route respectively, where $k \geq 1$*/

1: **PFT** $(p_s, i_s, j_s, p_d, i_d, j_d)$
2: **if** $(src.fail()||dis.fail())$ **then**
3: return null;
4: **else**
5: **if** $(p_s == p_d)$ **then**
6: **if** $(i_s == i_d)$ **then**
7: **while** Int.sw.fail() **do**
8: **if** $(((i_{s\&d} < \frac{n}{2})$ and $(j_{s\&d} < \frac{n}{2}))$ or $((i_{s\&d} \geq \frac{n}{2})$ and $(j_{s\&d} \geq \frac{n}{2})))$**then**
9: **if** (src==$uni$) **then**
10: return R$_0$= src→ext.sw$(p_x, j_s)$→$(p_d, j_d, i_d)$→ dis$(p_d, i_d, j_d)$;
11: **else if** (dis==$uni$) **then**
12: return R$_1$= src→$(p_s, j_s, i_s)$→ext.sw$(p_x, j_d)$→ dis$(p_d, i_d, j_d)$;
13: **else**
14: return R$_2$= src→ext.sw$(p_x, j_s)$→$(p_s, j_d, j_s)$→ int.sw$(k, p_s, j_d)$→$(p_d, j_d, i_d)$→dis;
15: **end if**;
16: **else**
17: **if** (src==$uni$) **then**
18: return R$_3$= src→ext.sw$(p_x, j_s)$→cluster $(v)$→ cluster $(j_d)$→dis$(p_d, i_d, j_d)$;
19: **else if** (dis==$uni$) **then**
20: return R$_4$= src→cluster $(j_s)$→cluster $(v)$ → ext.sw$(p_x, j_d)$→dis$(p_d, i_d, j_d)$;
21: **else**
22: return R$_5$= src→ cluster $(j_s)$→cluster $(j_d)$ → dis$(p_d, i_d, j_d)$;
23: **end if**;
24: **end if**;
25: **end while**;
26: **else**
27: **while** Link $(src, dis)$.fail() **do**
28: **if** $(((i_{s\&d} < \frac{n}{2})$ and $(j_{s\&d} < \frac{n}{2}))$ or $((i_{s\&d} \geq \frac{n}{2})$ and $(j_{s\&d} \geq \frac{n}{2})))$ **then**
29: return R$_6$= src→ext.sw$(p_x, j_s)$→$(p_d, i_d, j_s)$→ Int.sw$(k, p_d, i_d)$→dis$(p_d, i_d, j_d)$;
30: **else**
31: return R$_7$= src→int.sw$(k, p_s, i_s)$→$(p_s, i_s, j_w)$→ cluster $(w)$ →cluster $(id)$→dis;
32: **end if**;
33: **end while**;
34: **end if**;
35: **else**
36: **while** ext.sw $(p, j_m)$.fail() **do**
37: **if** $(j_s == j_d == j_m)$ **then**
38: **if** $(k > 1)$ **then**
39: return R$_8$= src→ext.sw$(p_x, j_s)$→ dis;
40: **else**
41: return R$_9$= src→int.sw$(k,p_s,i_s)$→$(p_s, i_s, j_v)$→ ext.sw$(p_x, j_v)$→$(p_d, i_d, j_v)$ →dis;
42: **end if**;
43: **else if** $(j_m == j_s)$ **then**
44: return R$_{10}$= src→int.sw$(k,p_s,i_s)$→$(p_s, i_s, j_d)$→ ext.sw$(p_x, j_d)$→dis$(p_d, i_d, j_d)$;
45: **else if** $(j_m == j_d)$ **then**
46: return R$_{11}$= src→ext.sw$(p_x, j_s)$→ $(p_d, i_d, j_s)$→ int.sw$(k,p_d,i_d)$ →dis$(p_d, i_d, j_d)$;
47: **end if**;
48: **end while**;
49: **end if**;
50: **end if**;

In order to deal with these failures in the real-world, we need to design a fault-tolerant routing scheme for P-Cube network (PFR). PFR takes the network status into consideration when making decisions of the routing to avoid network overcrowding and to achieve a perfect load balance as shown in Algorithm 3. PFR addresses three types of failures: internal switch failure, link failure, and external switch failure. In the PFR scheme, we call each internal switch $(k, p, i)$ connected with n servers as a cluster $i$. In Algorithm 3, we denote the source and destination addresses as $src$ $(p_s, i_s, j_s)$, and $dis$ $(p_d, i_d, j_d)$ respectively. When we have $(i = j)$, we call the server a unique server, $uni$ $(p, i, i)$.

Internal switch failure happens when the source and the destination servers are in the same pod and in the same switch (Line 5, 6). We assume that the internal switch $i$ or port has failed. Generally, the routing procedure can be divided into two cases as shown in Algorithm 3. The first case is when the $src$ and $dis$ servers are external servers (Line 8). We have three possible situations. The first situation is if $src$ = $uni$ (Line 9), then the traffic will cross the path Route0. Second, if $dis$ = $uni$ (Line 11), as a result the traffic will take the path Route1. Third, if $src \neq uni$ and $dis \neq uni$ (Line 13), the traffic will traverse the path Route2.

$$x \in \begin{cases} c & k = 1 \\ c + \frac{(k-1)n}{2}, c + \frac{(k-2)n}{2} + c, ..., c & k > 1, \end{cases} \quad (1)$$

where $c = j_z \ mod \ \frac{n}{2}$, $z = s, d$

Consequently, the $src$ and $dis$ can directly reach each other through $k$ external switches. In addition, the external switch $(p_x, j_z)$ is preferentially connected with the $src$ or $dis$ server based on the most available bandwidth, where the value $x$ can be acquired using (1). From another perspective, when the external switch $(p_x, j_z)$ or port fails, its connection bandwidth can be considered zero, thus the traffic can be routed via other external switches, where there are $k$ external switches that can use it to handle the faults in the network. The second case is when $src$ or $dis$ or both are internal servers (Line 16). Like the previous case, we have three possibilities of failure situation. The first possibility is if $src$ = $uni$ (Line 17) then the traffic will go through the path Route3. Second, if $dis$ = $uni$ (Line 19), the traffic will take the path Route4. Consider the case in which the external switch $(p_x, j_z)$ or the cluster$(v)$ connection fails, the traffic can be routed over other external switches or other clusters, where the value $x$ can be obtained using (1), since there are $k$ external switches and $(\frac{n}{2} - 1)$ clusters that can be used to deal with the faults in the network. Third, if $src \neq uni$ and $dis \neq uni$ (Line 21), the traffic will pass through the path Route5.

As for the link failure, it occurs when the source and the destination servers are in the same pod but are not in the same switch (Line 26). In this case, we assume the link $(src, dis)$ fails (Line 27). We will check if $src$ and $dis$ servers are external servers (Line 28), then the traffic will travel over the path Route6, where we can get $x$ from (1), since there are

$k$ external switches. Otherwise, the traffic will traverse the path Route7. Now we look at the case in which the cluster $(w)$ connection fails. The traffic can still get to its destination by passing it over other clusters, as there are $(n-2)$ clusters that we can leverage in order to take care of the faults in the network.

External switch failure arises when the source and the destination servers are in different pods (Line 35). Our presumption is that the external switch $(p_x, j_m)$ or port has failed (Line 36). The routing scheme can be considered in two cases as shown in Algorithm 3. The first case is when the $src$ and $dis$ servers connected by the external servers $(p_x, j_m)$ (Line 37). There are two possible situations here: The first situation is if *the level of the network is greater than* 1 (Line 38). That means there are $k-1$ of external switches $(p_x, j_m)$, that connects the $src$ and $dis$ with each other, where the value $x$ can be found using (1). then the traffic will cross Route8 by one of such external switches (Line 39). Second, if *the level of the network is equal to* 1 (Line 40), as a result, the traffic will take the Route9 (Line 41). From another perspective, when the external switch $(p_x, j_v)$ or port fails, the traffic still able to reach its destination by travel it over other external switches, where there are $(\frac{n}{2}-1)$ external switches that can use it to handle the faults in the network. The second case is when only the $src$ is connected to the external switch $(p_x, j_m)$ (Line 43), the traffic will traverse the Route10 (Line 44). Lastly, when only the $dis$ is connected to the external switch $(p_x, j_m)$, in this case, the traffic will go through Route11.

## 5. Analysis and Evaluation

### 5.1 Comparisons

In this subsection, we compare our proposed P-Cube with Fat-tree, DCell, Leaf-Spine and Bcube. The analysis of the basic network parameters of these structures is summarized in Table 1, where we choose 5 critical metrics to evaluate the performance of each architecture and the meanings of the 3 metrics are listed as follows:

Network Diameter: The diameter determines the maximum shortest distance (denoted by the number of links) between all the pair of nodes. In Table 1, we observe that P-Cube, Fat-tree and Leaf-Spine are outstanding, due to their stable and relatively small network diameter. In contrast, DCell, FiConn, and BCube have a big network diameter. Besides, their network diameter grows according to the growing of layers, as shown in Fig. 2.

Bisection Bandwidth: The network is partitioned into two equally-sized parts of link capacities, which by adding both of them, results in obtaining the bisection bandwidth. It can be used to measure the worst-case network capacity. P-Cube, Fat-tree and Leaf-Spine compared with other topologies (i.e. DCell, FiConn, and BCube) achieves the best bisection bandwidth and their bandwidth per server is stable.

Physical Cost and cabling complexity: Here, we compare the cost, the power consumption, and cabling complexity of P-Cube with Leaf-Spine, Fat-tree, BCube, DCell. As the total cost and power consumption of the servers are the same, we only estimate the cost and power consumption of both the switches and NICs. Based on the formulas shown in Table 1, we obtain the number of switches and links in the structures. Note that in Leaf-Spine we have chosen a non-oversubscription as per an actual industrially recommended configuration [27], because it uses leaf and spine switches with the same line speed, making comparisons more straightforward. Firstly, we consider five structures of Data Center using different ports switches to connect 2,048 servers. These structures are configured as follows: P-Cube structure is P-Cube (16, 1). Fat-tree structure is constructed of three switches layers, where each of the layers 1 and 2 contains 256 16- port switches and layer 3 contains 128 16-ports switches. The Leaf-Spine structure is composed of two switches layers, where the first layer contains 64 64-port switches and the second layer contains 16 128-ports switches. BCube structure is a part of $BCube_2$ constructed of 8 full $BCube_1$s, where $(n=16)$. DCell structure is a part of $DCell_2$ constructed of 8 $DCell_1$s. In this setup, P-Cube employs 128 16-ports internal switches and 16 64-ports external switches; it also uses 3 NICs ports for each of the half of the servers (external servers) and 2 NICs ports for each of the second half of the servers (internal servers). Leaf-Spine uses 64 64-port switches in Leaf layer and 16 128-ports switches in Spine layer; it also uses 1 NIC ports for each server. The remaining three structures, i.e., Fat-tree, Bcube and DCell utilize 640, 512 and 136 16-ports switches as well as 1, 3 and 3 NICs ports for each server, respectively.

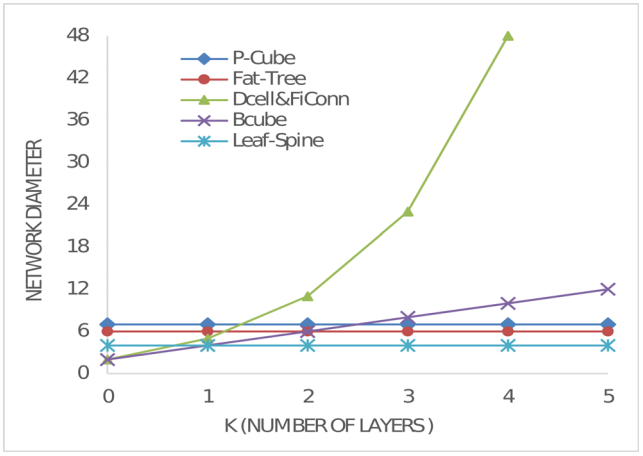



**Fig. 2** A comparison of network diameters between P-Cube topology and other topologies.

Then, we construct P-Cube, Fat-tree, Leaf-Spine, BCube, and DCell to connect the 4,096, 8,192 and 16,384 servers, respectively. In a word, the whole configuration of these structures and network components (i.e., the amount and ports of the switches) is portrayed in Table 3.

Table 2 shows the price and power consumption of the network components (i.e., switches and NICs) used in the data centers [8], [25], [26]. We obtain the costs and power consumption (shown in the table) from the existing commer-

**Table 1** A comparison between P-Cube architecture and other network architectures.

| | Diameter | No. of switch | No. of wires | No. of servers | Bisection bandwidth |
|---|---|---|---|---|---|
| P-Cube | 7 | $\frac{N}{n} + (nk), k > 0$ | $\{\sum_{n=1}^{n}(n-1)\} \times \frac{nk}{2} + (\frac{k}{2}+1) \times \frac{n^3k}{2}, k > 0$ | $(\frac{n^3}{2} \times k), k > 0$ | $\frac{N}{2}$ |
| FatTree[6] | 6 | $\frac{5N}{n}$ | $N \log \frac{n}{2} \frac{N}{2}$ | $\frac{n^3}{4}$ | $\frac{N}{2}$ |
| DCell[7] | $< 2^{k+1} - 1$ | $\frac{N}{n}$ | $(\frac{k}{2}+1) \times N$ | $> (n+\frac{1}{2})^{2^k} - \frac{1}{2}, < (n+1)^{2^k} - 1$ | $\frac{N}{4N \log_n N}$ |
| BCube[8] | $k+1$ | $(k+1)n^k$ | $N \log_n N$ | $n^{k+1}$ | $\frac{N-1}{k+1}$ |
| Leaf-Spine | 4 | $\frac{5N}{2n_1}$ | $2 \times$ N | $\frac{n_1 \times N_e}{2}$ | $\frac{N}{2}$ |
| FiConn[12] | $< 2^{k+1} - 1$ | $\frac{N}{n}$ | $(2 - \frac{1}{2^k}) \times N - \frac{N}{n}$ | $\geq ((\frac{n}{4})^{2^k} \times 2^{k+2})$ | $\frac{N}{\log_N}$ |

$n_1$ is the number of ports on a leaf switch, $N_e$ is the total number of leaf switches.

**Table 2** Cost and power consumption of switch and NICs [8], [25], [26].

| | Ports | Cost($) | Power(W) |
|---|---|---|---|
| **Switch** [25], [26] | P ≤ 128 | 20 per port | 2 per port |
| | P = 256 | 10922 | 622 |
| | P = 512 | 65532 | 2490 |
| **NIC** [8] | 1 | 5 | 5 |
| | 2 | 10 | 7.5 |
| | 4 | 20 | 10 |

cial components and also the laboratory pro-type, which can be changed with the adoption of novel technology. They would be understood as approximations to real values.

Table 3 summarizes the construction costs and power consumptions of both the switches and the NICs, in addition to the number of the wires of the four networking structures for a data center with 2048, 4096, 8192, and 16384 servers. Note that in the aforementioned settings, DCell and BCube structures are incomplete, consequently, there is need for constructing partial structures.

Moreover, Fig. 3(a) shows that compared to Fat-tree and BCube, P-Cube, Leaf-Spine and DCell use less number of links, regardless of the data center size. In addition, the number of links used in P-Cube is less than that of Leaf-Spine and DCell, this is because half of the servers in P-Cube structure (internal servers) are endowed with only dual ports irrespective of the network size, as explained in Sect. 3.1. Hence, P-Cube dramatically reduces the complexity of cabling, particularly for large container data centers. Furthermore, as illustrated in Table3 and Fig. 3(b), PCube and DCell outperform Fat-tree, Leaf-Spine, and BCube in terms of the construction costs of both the switches and the NICs, regardless of the size of the data center. However, PCube, Leaf-Spine, and DCell outperform Fat-tree and BCube in terms of the power consumption of both the switches and the NICs as shown in Fig. 3(c).

## 5.2 Evaluation

In this subsection, we conduct simulations utilizing OPNET simulator to evaluate the performance of P-Cube network using the proposed P-CubeRouting algorithm. However, there are several measurements to evaluate the DCN topology, here we will focus on two important measurements: network throughput and latency (ETE delay). The definition of ETE delay is the time spent to send packets from the source node to the destination node, which includes a processing delay, a transmission delay, a propagation delay, and the queuing delay. The definition of offered load is the number of packets per second injected into the network. The definition of throughput is the number of the packets arrived at the destinations in a duration of time, where with increasing the offer lode, the latency will decrease until load reaches the end of limitation of that network could afford, which is called its saturation point.

In the simulations, the network of P-Cube is set to 432 servers, with n=6 and k=4. DCell network is set to 420 servers, with n=20 and k=1. The BCube network is set to 400 servers. Fat-tree network is set to 432 servers, with 12 pods. For the packet size is set to 4096 bits, and we set the link bandwidth to 10 Gbps.

We illustrate the packet delay. As shown in Fig. 4(a), in around 1.8 $\mu$s, P-Cube ensures very low delay until the offered load of 0.81. While reaching this load, the network becomes saturated and the packet delay increases noticeably from 1.8 microseconds to around 0.08 $ms$. Comparing with DCell, BCube and Fat-tree networks illustrates ETE delay of DCell, BCube, and Fat-tree saturates at 0.73, 0.65, and 0.57 of the offered load respectively. We observe P-Cube saturates of offered load higher 8%, 16%, and 24% than structures DCell, BCube and Fat-tree respectively.

We then evaluate the throughput. As shown in Fig. 4(b), P-Cube achieved a high throughput of about 4.3 $Tbps$, with the offered load of 3 $\mu$s. After this load, the throughput increases slightly. Comparing with DCell, BCube, and Fat-tree networks in the same offered load of 3 $\mu$s, the achieved throughput is about 2.53 $Tbps$, 2.51 $Tbps$, and 2.07 $Tbps$ respectively.

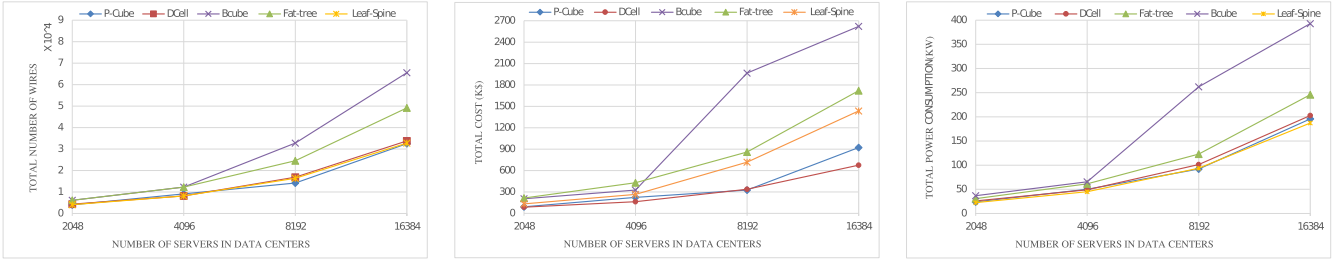
## 5.3 Summary

P-Cube significantly minimizes the required numbers of switches and wires compared to Fat-tree, and BCube. In consequence, P-Cube structures largely reduce the complexity of cabling compared to other structures, especially for large container data centers. Moreover, P-Cube extremely outperforms Fat-tree, and BCube structures in terms of the total cost and the power consumption. In addition to these benefits, P-Cube considerably outperforms Fat-tree, BCube,

**Table 3** Cost, power consumption, and wiring comparison of different networking structures with different data centers sizes.

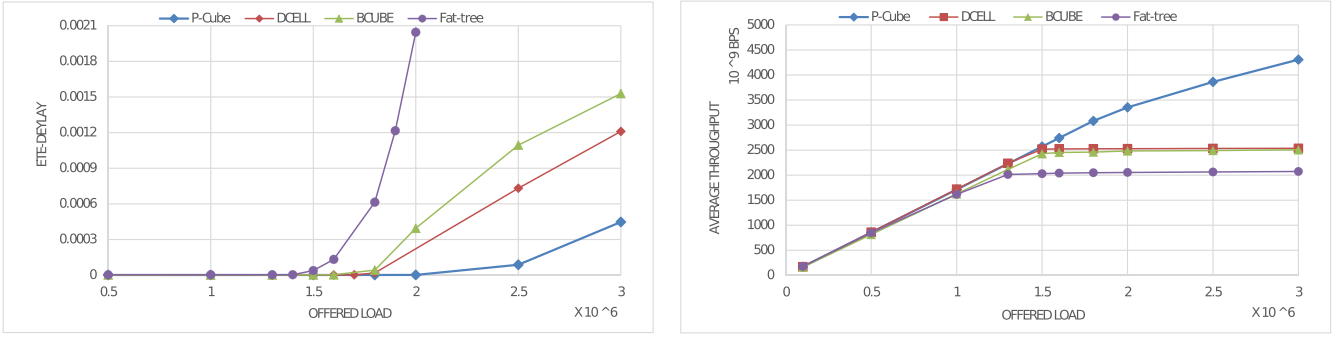
| Servers No. | Structures | Cost($k$\$) | | Power($kw$) | | No. of links | Switches No : amount (ports) |
|---|---|---|---|---|---|---|---|
| | | Switch | NIC | Switch | NIC | | |
| **2048** | Fat-tree | 204.8 | 10.2 | 20.5 | 10.2 | 6144 | L1, L2=256(16), L3=128(16) |
| | Leaf-Spine | 122.8 | 10.2 | 12.3 | 10.2 | 4096 | L1 = 64(64), L2 = 16(128) |
| | BCube | 163.8 | 40.9 | 16.4 | 20.5 | 6144 | 512(16) |
| | DCell(2,176) | 43.5 | 43.5 | 4.3 | 21.8 | 4352 | 136(16) |
| | P-Cube(16,1) | 61.4 | 30.7 | 6.1 | 17.9 | 4032 | 128(16)[1], 16(64)[2] |
| **4096** | Fat-tree | 409.6 | 20.5 | 40.9 | 20.5 | 12288 | L1, L2=512(16), L3 = 256(16) |
| | Leaf-Spine | 245.7 | 20.5 | 24.6 | 20.5 | 8192 | L1 = 128(64), L2 = 32(128) |
| | BCube | 245.7 | 81.9 | 24.6 | 40.9 | 12288 | 768(16) |
| | DCell(4,080) | 81.6 | 81.6 | 8.2 | 40.8 | 8160 | 255(16) |
| | P-Cube(16,2) | 163.8 | 61.4 | 16.3 | 35.8 | 10112 | 256(16)[1], 32(128)[2] |
| **8192** | Fat-tree | 819.2 | 40.9 | 81.9 | 40.9 | 24576 | L1, L2=512(32), L3 = 256(32) |
| | Leaf-Spine | 677.1 | 40.9 | 52.6 | 40.9 | 16384 | L1 = 128(128), L2 = 32(256) |
| | BCube | 1802 | 163.8 | 180.2 | 81.9 | 32768 | 5632(16) |
| | DCell(8,448) | 168.9 | 168.9 | 16.9 | 84.5 | 16896 | 264(32) |
| | P-Cube (32,1) 8 pods | 204.8 | 122.8 | 20.5 | 71.7 | 14208 | 256(32)[1], 16(128)[2] |
| **16384** | Fat-tree | 1638 | 81.9 | 163.8 | 81.9 | 49152 | L1, L2=1024(32), L3 = 512(32) |
| | Leaf-Spine | 1354 | 81.9 | 105.3 | 81.9 | 32,768 | L1 = 256(128), L2 = 64(256) |
| | BCube | 2293 | 327.7 | 229.4 | 163.8 | 65536 | 7168(16) |
| | DCell(16,896) | 337.9 | 337.9 | 33.8 | 168.9 | 33792 | 528(32) |
| | P-Cube (32,1) | 677 | 245.7 | 52.6 | 143.3 | 32512 | 512(32)[1], 32(256)[2] |

[1] Internal switches, [2] External switches.




(a) The total number of links. (b) Cost construction comparison. (c) Power consumption comparison.

**Fig. 3** Cost and power consumption comparison of different data center networks with several servers.




(a) ETE Delay. (b) Throughput.

**Fig. 4** A comparison of ETE delay and throughput between P-Cube topology and other topologies.

and DCell structures in terms of network throughput and ETE Delay, where the maximum throughput of P-Cube is about twice of Fat-tree. Based on the above analysis, we can observe that the P-Cube structure is more suitable for constructing large-scale data center networks with low cost and power consumption.

## 6. Conclusion

In this paper, we proposed a novel server-centric structure for data centers named P-Cube that provides high performance, scalability, fault tolerance, reliability, and low-cost networks. P-Cube eliminates the need to use high-end switches by exploiting the built-in dual-port configuration found in most data center servers. Further, the utilization of commodity switches and servers make P-Cube a low-cost structure. P-Cube also provides a higher network capacity with a high bisection width and a low stable diameter. It can accommodate hundreds of thousands of nodes, which proves that P-Cube can provide high scalability. The specially designed routing scheme (P-CubeRouting and DFR) aid P-Cube to realize its maximum theoretical performance and balance the numerous links in the network. At last, we provided numerical results that show our topology is better than other topologies in the literature, such as Fat-tree, BCube and DCell, on the basis of network throughput and latency. Furthermore, the total cost, complexity of cabling and power consumption of Fat-tree and BCube structures are significantly higher than our structure.

## Acknowledgments

This work was supported in part by the National Key RD Program of China under Grant 2018YFE0202800, and National Natural Science Foundation of China under Grant 61634004 and 61934002, and the Natural Science Foundation of Shaanxi Province for Distinguished Young Scholars under Grant no 2020JC-26, and the Fundamental Research Funds for the Central Universities under Grant No. JB190105 and XJS200119, and the Open Project Program of the State Key Laboratory of Mathematical Engineering and Advanced Computing under Grant No. 2019A01.

### References

[1] S. Ghemawat, H. Gobioff, and S-T. Leung, "The Google file system," ACM SIGOPS Oper. Syst. Rev., vol.37, no.5, 2003.
[2] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly, "Dryad: Distributed data-parallel programs from sequential building blocks," ACM SIGOPS Oper. Syst. Rev., vol.41, no.3, pp.59–72, March 2007.
[3] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," Commun. ACM, vol.51, no.1, pp.107–113, Jan. 2008.
[4] F. Chang, J. Dean, S. Ghemawat, W.C. Hsieh, D.A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R.E. Gruber, "Bigtable: A distributed storage system for structured data," ACM Trans. Comput. Syst. (TOCS), vol.26, no.2, p.4, June 2008.
[5] D. Borthakur, "The hadoop distributed file system: Architecture and design," Hadoop Project Website, vol.11, no.2007, p.21, Aug. 2007.
[6] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," ACM SIGCOMM Comput. Commun. Rev., vol.38, no.4, pp.63–74, Aug. 2008.
[7] 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 Comput. Commun. Rev., vol.38, no.4, pp.75–86, Aug. 2008.
[8] 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 Comput. Commun. Rev., vol.39, no.4, pp.63–74, Aug. 2009.
[9] G. Lu, C. Guo, Y. Li, Z. Zhou, T. Yuan, H. Wu, Y. Xiong, R. Gao, and Y. Zhang, "Serverswitch: A programmable and high performance platform for data center networks," Nsdi, vol.11, pp.2–2, March 2011.
[10] 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," ACM SIGCOMM Comput. Commun. Rev., vol.39, no.4, pp.51–62, 2009.
[11] Y. Sun, J. Chen, Q. Lu, and W. Fang, "Diamond: An improved fat-tree architecture for large-scale data centers," J. Commun., vol.9, no.1, pp.91–98, Jan. 2014.
[12] D. Li, C. Guo, H. Wu, K. Tan, and S. Lu, "FiConn: Using backup port for server interconnection in data centers," INFOCOM, vol.9, pp.2276–2285, April 2009.
[13] Y. Liao, D. Yin, and L. Gao, "DPillar: Scalable dual-port server interconnection for data center networks," 2010 Proc. 19th Int. Conf. on Computer Communications & Networks, pp.1–6, Aug. 2010.
[14] N. Farrington and A. Andreyev, "Facebook's data center network architecture," Proc. 2013 IEEE Optical Int. Conference, pp.49–50, May 2013.
[15] A. Singh, J. Ong, A. Agarwal, G. Anderson, A. Armistead, R. Bannon, S. Boving, G. Desai, B. Felderman, P. Germano, A. Kanagala, J. Provost, J. Simmons, E. Tanda, J. Wanderer, U. Hölzle, S. Stuart, and A. Vahdat, "Jupiter rising: A decade of clos topologies and centralized control in Google's datacenter network," ACM SIGCOMM Comput. Commun. Rev., vol.45, no.1, pp.183–197, Sept. 2015.
[16] Y. Yu and C. Qian, "Space shuffle: A scalable, flexible, and high-performance data center network," IEEE Trans. Parallel Distrib. Syst., vol.27, no.11, pp.3351–3365, Feb. 2015.
[17] D. Guo, T. Chen, D. Li, Y. Liu, X. Liu, and G. Chen, "BCN: Expansible network structures for data centers using hierarchical compound graphs," Proc. IEEE INFOCOM, pp.61–65, April 2011.
[18] D. Guo, C. Li, J. Wu, and X. Zhou, "DCube: A family of high performance modular data centers using dual-Port servers," Elsevier J. Comput. Commun., Vol.53, pp.13–25, Feb. 2013.
[19] Z. Chkirbene, S. Foufou, M. Hamdi, and R. Hamila, "Hyper-flatnet: A novel network architecture for data centers," Proc. 2015 IEEE Int. Conf. on Communication Workshop (ICCW), pp.1877–1882, June 2015.
[20] H. Wu, G. Lu, D. Li, C. Guo, and Y. Zhang, "MDCube: A high performance network structure for modular data center interconnection," Proc. 5th Int. Conf. on Emerging networking experiments & technologies, pp.25–36, Dec. 2009.
[21] Z. Li, Z. Guo, and Y. Yang, "BCCC: An expandable network for data centers," IEEE/ACM Trans. Netw., vol.24, no.6, pp.3740–3755, Dec. 2016.
[22] M. Alizadeh and T. Edsall, "On the data path performance of leaf-spine data center fabrics," IEEE Annual Symposium on High-Performance Interconnects (HOTI), pp.71–74, Aug. 2013.
[23] Z. Li and Y. Yang, "GBC3: A versatile cube-based server-centric network for data centers," IEEE Trans. Parallel Distrib. Syst., vol.27, no.10, pp.2895–2910, Dec. 2015.
[24] T. Wang, Z. Su, Y. Xia, B. Qin, and M. Hamdi, "NovaCube: A low latency Torus-based network architecture for data centers," Proc. 2014 IEEE Global Communications Conference (GLOBECOM), pp.2252–2257, Dec. 2014.

[25] F. Yan, X. Xue, and N. Calabretta, "HiFOST: A scalable and low-latency hybrid data center network architecture based on flow-controlled fast optical switches," IEEE/OSA J. Opt. Commun. Netw., vol.10, no.7, pp.1–14, July 2015.
[26] F. Yan, W. Miao, and N. Calabretta, "On the cost, latency, and bandwidth of LIGHTNESS data center network architecture," Proc. 2015 Int. Conf. on Photonics in Switching (PS), pp.130–132, Sept. 2015.
[27] Arista, "Cloud networking: Scaling out datacenter networks," https://www.arista.com/assets/data/pdf/Whitepapers/Cloud_Networking__Scaling_Out_Data_Center_Networks.pdf, Jan. 2020.

**Moeen Al-Makhlafi** received the B.E. degree in Mathematical and Computer Science from IBB University of Engineering and Technology, in 2010 and an M.E. degree in Computer Science and Technology from Xidian University in 2019. He is currently working toward the Ph.D. degree in the data center, cloud computing, future Internet architecture in the State key laboratory of ISN, Xidian University.

**Huaxi Gu** received the B.E. degree, M.E. and Ph.D. in Telecommunication Engineering and Telecommunication and Information Systems from Xidian University, Xidian in 2000, 2003 and 2005 respectively. He is full Professor in the State key lab of ISN, Telecommunication Department, Xidian University, Xidian, China. His current interests include interconnection networks, networks on chip and optical intrachip communication. He has more than 100 publications in refereed journals and conferences. He has been working as a reviewer of IEEE Transaction on Computer, IEEE Transactions on Dependable and Secure Computing, IEEE system Journal, IEEE Communication letters, Information Sciences, Journal of Supercomputing, Journal of System Architecture, Journal of Parallel and Distributed Computing, Microprocessors and Microsystems etc.

**Xiaoshan Yu** received the M.E. degree in Electronics and Communications Engineering from Xidian University in 2013 and the Ph.D. degree in Telecommunication and Information System from Xidian University in 2016. Now he is doing postdoctoral programme in the State key lab of ISN, Xidian University. His main research interests are related to optical interconnected networks, data center networks.

**Yunfeng Lu** received the B.E. degrees in Information Technology from Jilin University, in 2016. Since 2016, he has been working toward the M.E. degree in telecommunication and information systems in the State key laboratory of ISN, Xidian University. The main research interests of him include: optical interconnected networks and high performance computing networks.