A Novel Network Structure with Power Efficiency and High Availability for Data Centers

Zhenhua Li and Yuanyuan Yang, *Fellow, IEEE*

Abstract—Designing a cost-effective network for data centers that can deliver sufficient bandwidth and provide high availability has drawn tremendous attentions recently. In this paper, we propose a novel server-centric network structure called RCube, which is energy efficient and can deploy a redundancy scheme to improve the availability of data centers. Moreover, RCube shares many good properties with BCube, a well known server-centric network structure, yet its network size can be adjusted more conveniently. We also present a routing algorithm to find paths in RCube and an algorithm to find multiple parallel paths between any pair of source and destination servers. In addition, we theoretically analyze the power efficiency of the network and availability of RCube under server failure. Our comprehensive simulations demonstrate that RCube provides higher availability and flexibility to make trade-off among many factors, such as power consumption and aggregate throughput, than BCube, while delivering similar performance to BCube in many critical metrics, such as average path length, path distribution and graceful degradation, which makes RCube a very promising empirical structure for an enterprise data center network product.

Index Terms—Data center networks, diameter, parallel paths, server-centric, power efficiency, high availability

1 Introduction

Cloud computing has been widely deployed in both industry and academia as it provides an innovative way to organize computing resources, which generates tremendous benefits. Data centers, as the cornerstone of cloud computing, play a key rule in determining the performance and further the economical ecosystem of cloud computing. Hence, regarding the importance and technology advance, several network structures for data centers have been studied within the decade, which are mainly divided into two categories: switch-centric networks and server-centric networks. In a switch-centric network, servers are only responsible of sending packets to and receiving packets from the network, while switches find paths and relay packets for communications between servers. The classical FatTree [12], Portland [17], and newly emerging Jupiter [13] belong to this group. On the other hand, in a server-centric network, servers act not only as the source or destination nodes, but also as the relay nodes for each other, and switches act simply as crossbars transferring packets from one side to another. Therefore the routing task of a server-centric network is assigned to servers. DCell [8], BCube [6] and BCCC [9] belong to this category. A direct benefit derived from server-centric networks is that the network hardware cost can be drastically reduced, as inexpensive commodity switches, such as layer-2 switches, are sufficient provided that routing tasks have been taken by servers which hold abundant computing resources. In addition, server-centric networks are established with fewer interconnection switches than switch-centric networks, given a similar network size, hence a server-centric network consumes much less energy than a switch-centric network. Moreover, as servers are much more programmable than switches, server-centric network structures greatly help accelerate the process of network innovation.

As a server-centric network, BCube is well known for several good properties, including large network size, short network diameter, graceful degradation and multiple parallel paths between any pair of servers. These features make BCube a very popular structure for data center networks (DCNs). However, the power consumed by the interconnection network of BCube still occupies a relatively large fraction of the total power consumption of the data center built on BCube. On the contrary, DCell costs much less power than BCube, but it is implemented in a way of allowing servers to connect directly to each other without via any switches. Hence, as compared in [15], DCell has much poorer bisection bandwidth, a key factor for throughput performance, than BCube. Besides, DCell behaves much more sensitively to component failure, in case of both server and link failures. To reduce power consumption, PCube [7] was proposed, which improves BCube by systematically turning off the unused switches in BCube based on a series of rules. However, the bisection bandwidth of PCube is poor. Besides, turning switches on and off in an on-demand manner requires estimating and predicting the traffic load, either in hardware or software. Given that in a server-centric network, most tasks are executed in servers, workload in servers is heavy and only layer-2 switches are used

• *The authors are with the Department of Electrical and Computer Engineering, Stony Brook University, Stony Brook, NY 11794. E-mail: {zhenhua.li, yuanyuan.yang}@stonybrook.edu.*

Manuscript received 18 Feb. 2017; revised 16 Sept. 2017; accepted 5 Oct. 2017. Date of publication 12 Oct. 2017; date of current version 12 Jan. 2018. (Corresponding author: Yuanyuan Yang.)
Recommended for acceptance by X. Wang.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TPDS.2017.2762297

for the purpose of low cost. Hence, either a software estimator or a hardware one would potentially generate an extra capital expenditure for a server-centric network.

Meanwhile, as the size of DCNs continues to increase, the inevitable component failure becomes normal. To alleviate this situation, redundant servers are added into DCNs as backups [4]. When normal servers fail, due to either hardware malfunction or software crash, the backup servers will warm up and start to work on the tasks left by the failed ones. Usually, since putting some servers into standby mode will not affect other servers, switch-centric networks can easily adopt a redundancy scheme. However, it is more difficult to deploy redundancy in server-centric networks as every server takes the responsibility of forwarding packets for each other. Putting some servers to standby mode may cut off paths for some pairs of servers, which further impacts the throughput of the entire system. Moreover, it is stated in [13] that servers are more prone to malfunction than switches. Hence high availability becomes an even crucial challenge to server-centric network structures. However, none of existing work concerning server-centric network achieves this goal. This impedes server-centric networks from being used in a reliable DCN product.

In this paper, we propose a novel server-centric network structure, called RCube, which is a recursively built structure with many good properties. First of all, RCube has a diameter that increases linearly to the network order, and has multiple parallel paths between any pair of servers within it. We also show that BCube is a special case of RCube. On the other hand, given the same network size and order, RCube consumes significantly less power than BCube. In addition, RCube can be configured to the active/passive redundancy mode such that putting servers into standby will not affect the performance of the remaining servers while the availability of the entire system can be significantly enhanced. To the best of our knowledge, RCube is the only server-centric network that can support this feature. Our comprehensive simulations demonstrate that RCube provides higher availability and flexibility to make trade-off among many factors, such as power consumption and aggregate throughput, than BCube, while delivering similar performance to BCube in many critical metrics, such as average path length, path distribution and graceful degradation.

The rest of this paper is organized as follows. Section 2 describes the recursive construction of RCube and provides the routing algorithms, along with some related properties of RCube. Section 3 analyses the power consumption of RCube. Section 4 discusses how to deploy the active/passive redundancy model into RCube and theoretically analyses the improvement on system availability. Section 5 gives comprehensive performance evaluations for RCube. Finally, Section 6 debriefs the state-of-the-art studies in the related field and our future work, and Section 7 concludes this paper.

2 RCube Network Structure

In this section, we introduce the structure of BCube first. Then, we present the procedure of constructing an RCube, along with some related properties and we also provide routing algorithms for finding paths in RCube.

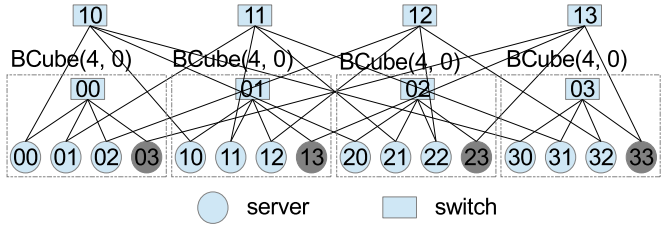


Fig. 1. Structure of $BCube(4,1)$ which is composed of 4 $BCube(4,0)$s and 4 switches. All the eight switches are evenly divided into two levels and each server connects to one switch in each level through one of its two ports.

2.1 Structure of BCube

To facilitate the further discussion on high availability in server-centric networks in the following section, we first introduce some background of BCube structure. BCube is well known for its high aggregate throughput, low diameter and abundant parallel paths. BCube structure can be represented with two parameters n and k. A $BCube(n,k)$ consists of n^{k+1} $(k+1)$-port servers and $(k+1)n^k$ n-port switches which are evenly divided into $k+1$ levels. Therefore, in each level, there are n^k switches, and each server connects one switch in every level through one of its ports based on dedicatedly designed principles. Meanwhile, BCube can also be interpreted in a recursive way. The basic construction element for BCube is $BCube(n,0)$, which is composed of n servers connected by an n-port switch. Given this, $BCube(n,k)$ is constructed with n $BCube(n,k-1)$s and n^k switches. More details on the connection rule between switches and servers in BCube can be found in [6]. Fig. 1 depicts an example of $BCube(4,1)$.

As introduced previously, BCube has many luring properties, such as low network diameter, high aggregate throughput, multiple parallel paths. However, BCube lacks the flexibility in adjusting its network size. In addition, it houses numerous switches that cost huge power consumption. Also, it has limited ability to support services with high availability requirement, which will be described later in this paper.

2.2 Structure of RCube

RCube is a recursively constructed network using servers and switches equipped with multiple line cards and network interface controller (NIC) ports respectively. Let $RCube(n,m,0)$ denote the basic building block within which there are $n+m$ servers connected by an $(n+m)$-port switch. Then an $RCube(n,m,k)$, where $k \in \aleph^+$, is constructed from n $RCube(n,m,k-1)$s with n^k $(n+m)$-port switches. Denote the network size of $RCube(n,m,k)$, the number of servers an $RCube(n,m,k)$ holds, as $S_{RCube(n,m,k)}$. Thus based on the construction process described above, we have the following property

$$\begin{cases} S_{RCube(n,m,0)} = n+m \\ S_{RCube(n,m,k)} = n \cdot S_{RCube(n,m,k-1)}. \end{cases} \quad (1)$$

Hence, the size of $RCube(n,m,k)$ is $(n+m)n^k$.

Next, we describe the detailed construction procedure of an $RCube(n,m,k)$ structure. In an $RCube(n,m,0)$, we identify each server by a symbol a_0, where $a_0 \in \{0,1,\ldots,n+m-1\}$. As aforementioned, an $RCube(n,m,k)$ is composed of n $RCube(n,m,k-1)$s. These n $RCube(n,m,k-1)$s can be indexed by a_k, where $a_k \in \{0,1,\ldots,n-1\}$. Then n $RCube(n,m,k-2)$s in an $RCube(n,m,k-1)$ can be

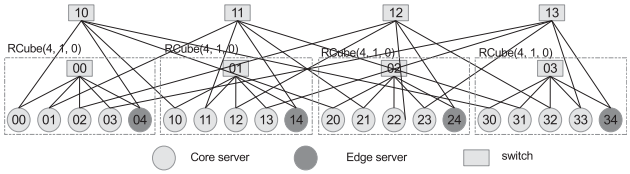


Fig. 2. Structure of $RCube(4,1,1)$ which is composed of 4 $RCube(4,1,0)$s and 4 switches.

identified by a_{k-1}, $a_{k-1} \in \{0,1,\ldots,n-1\}$. This process continues until k decreases to 0. Thus in this way, each server in an $RCube(n,m,k)$ can be uniquely determined by a $(k+1)$-tuple, $a_k a_{k-1} \ldots a_1 a_0$. A server with address $a_k a_{k-1} \ldots a_0$ can be located by searching the postfix $a_{k-1} a_{k-2} \ldots a_0$ in the a_k^{th} $RCube(n,m,k-1)$ and so forth. We further say that a server is a core server if $a_0 < n$, otherwise, we call it an edge server.

In an $RCube(n,m,k)$, each server connects to $(k+1)$ switches and each switch connects to $(n+m)$ servers, thus there are $(k+1)n^k$ switches in total. Following a similar rule, the address of switches in $RCube(n,m,k)$ can be denoted as $s_k s_{k-1} \ldots s_0$, where $s_i \in \{0,1,\ldots,n-1\}$, $i \in \{0,1,\ldots,k-1\}$ and $s_k \in \{0,1,\ldots,k\}$. These $(k+1)n^k$ switches are evenly divided into $k+1$ levels and s_k indicates which level a switch is located in. In addition, the n^k newly added switches are in the top level and the remaining kn^k switches are in lower levels. Each core server is equipped with $k+1$ NIC ports numbered by 0 to k, which are used to connect to a switch in the corresponding level under the condition that

$$a_k a_{k-1} \ldots a_{s_k+1} a_{s_k-1} \ldots a_0 = s_{k-1} s_{k-2} \ldots s_0. \quad (2)$$

Edge servers are the same as core servers except that they are connected to switches under the following condition:

$$a_k a_{k-1} \ldots a_1 = a_{k-1} s_{k-2} \ldots s_0. \quad (3)$$

To unify Eqs. (2) and (3), we define that the address of an edge server can be presented in different forms. In other words, $a_k a_{k-1} \ldots a_{i+1} a_0 a_i a_{i-1} \ldots a_1$, where $i \in \{0,1,\ldots,k\}$, refers to the same edge server and it connects to the switch in the ith level. Therefore, Eq. (3) can be included into Eq. (2). Then the procedure of constructing an $RCube(n,\ m,k)$, $k > 0$, is as follows. First, we have n $RCube(n,m,k-1)$s and n^k switches. Then connect each server to the corresponding switch among the n^k switches based on Eq. (2). That concludes the construction procedure.

Fig. 2 shows an example of $RCube(4,1,1)$, which is composed of 4 $RCube(4,1,0)$s and 4 switches. In $RCube(4,1,0)$, server 04 is an edge server and it has two address forms, 04 and 40. It connects to switch 00 in the form of 04, while connecting to switch 10 in the form of 40. To better understand the structure of RCube, we also provide a topological view of $RCube(4,1,1)$ in Fig. 3, where rectangles stand for switches connecting to servers within them. In particular, blue rectangles are used to construct $RCube(4,1,0)$s, while red ones are deployed to build $RCube(4,1,1)$.

It can be clearly observed that $BCube(n,k)$ is a special case of $RCube(n,m,k)$ where $m=0$.

2.3 Finding Paths in RCube

Next, we discuss algorithms and related properties concerning the routing in RCube.

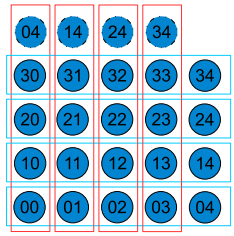


Fig. 3. Geometrical view of $RCube(4,1,1)$.

We say that a server has only one-hop distance to its neighbors, the servers connected by the same switch, since switches in server-centric networks act simply as a crossbar. Therefore, a path between a source and a destination usually includes multiple hops via intermediate servers.

Suppose source server A, with an address $a_k a_{k-1} \ldots a_0$, wants to communicate with destination server A$'$, denoted as $a'_k a'_{k-1} \ldots a'_0$. Building a path between A and A$'$ can be treated as correcting the digit differences between $a_k a_{k-1} \ldots a_0$ and $a'_k a'_{k-1} \ldots a'_0$. This algorithm is summarized as RCubeRouting in Table 1. RCubeRouting finds the path iteratively and in each iteration it corrects one digit difference between A and A$'$. RCubeRouting uses a permutation of array $[k, k-1, \ldots, 0]$ to decide in which sequence these different digits should be corrected. Moreover, we constrain that a_0 of a source edge server should be corrected in the first iteration and a'_0 of a destination edge server should be corrected in the last iteration. In other words, edge servers can only appear at the two ends of a path. We make such a constraint to facilitate routing, as allowing edge nodes to appear as intermediate nodes in a path will increase the complexity of the routing rules. Recall that an edge server has an address in $k+1$ forms, therefore we should revise

TABLE 1
RCube Routing Algorithm for Finding a Path from a Single Source to a Single Destination

Input: source A, destination A$'$,
and Π **which is a permutation of** $[k, k-1, \ldots, 0]$
Output: the list of intermediate servers of the path from A to B, path(A, B)
RCubeRouting(A, A$'$**,** Π**)**

```
path(A, A') = {A};
if A[0] ≥ n //change source address format
  tmp_node[π_k] = A[0];
  for (i = 0; i < π_k; i++)
    tmp_node[i] = A[i + 1];
  for (i = π_k + 1; i <= k; i++)
   tmp_node[i] = A[i];
 else
  tmp_node = A;
if A'[0] ≥ n
  destin[π_0] = A'[0];
  for (i = 0; i < π_0; i++)
   destin[i] = A'[i + 1];
  for (i = π_0 + 1; i <= k; i++)
   destin[i] = A'[i];
 else
  destin = A';
 for (i = k; i > 0; i- -)
  tmp_node[π_i] = destin[π_i];
  append tmp_node to path(A, A');
 return path;
```

the source and destination to their corresponding forms depending on the permutation in use. For example, in $RCube(4,1,1)$, server 04 wants to communicate with server 14 and the permutation in use is [1, 0]. Server 14 should be revised to 41 while 04 keeps its representation. Then the path between them would be {04, 01, 41}, which can also be represented as {04, 01, 14}.

Note that RCubeRouting in Table 1 does not guarantee that the resulted path is the shortest one for two reasons. The first reason is that an improper permutation may cause a longer path. For example, in an $RCube(4,1,2)$, servers 012 and 124 are neighbors directly connected by a switch based on the construction rule, thus one hop is sufficient for them. However, if we use a permutation where $\pi_0 \neq 2$, the path generated by RCubeRouting will have a length of 3. The other reason lies in the nature of RCube structure where the fact of edge servers being intermediate nodes helps shorten path lengths between some pairs of servers. Use $RCube(4,1,2)$ as the example again, in which server 012 wants to communicate with server 123. With the help of server 124, packets can be transferred from 012 to 123 within 2 hops. On the contrary, RCubeRouting will generate a path with a length of 3, no matter which permutation is used. This situation becomes even more complicated when k increases to some large numbers. However, finding the shortest path is a computing hungry process, which is unacceptable especially for server-centric networks, because the calculation workloads have already been very stressful on them. On the other hand, RCubeRouting efficiently checks every digit between source and destination servers and then decides the routing path without any extra calculation, which is more suitable for server-centric networks. Besides, the difference of the lengths between the shortest path and the path generated by RCubeRouting is minimal, which we will show later, such that it has only a negligible impact on the network performance.

Next, we study an important topological property of $RCube(n,m,k)$ regarding its diameter, and present it as the following theorem.

Theorem 1. *The diameter of $RCube(n,m,k)$ is $k+1$.*

Proof. From RCubeRouting in Table 1, we can see that finding a path between source and destination servers is actually to correct every digit of their addresses from the source server to the destination server. In an $RCube(n, m, k)$, the address of a server has $k+1$ digits and RCubeRouting corrects one digit in each iteration. Thus, in at most $k+1$ iterations, RCubeRouting can correct the address from the source to the destination. Hence, $k+1$ hops are sufficient for a packet to travel from the source server to the destination server. Next we show that $k+1$ hops are also necessary to send packets to the destination server that is farthest from the source. This can be shown by an example where every digit in the address of the source server is 0 and every digit in the address of the destination server is 1. In this case, edge servers do not help shorten the distance between the source and destination servers. Thus $k+1$ hops are necessary. □

Based on Theorem 1, we can see that the diameter of $RCube(n,m,k)$ increases only linearly to its network order k. Thus applications running on RCube can enjoy an excellent short communication latency. This is important as one of the requirements for cloud computing is that applications or tasks can be assigned to any servers in the data center. Another observation is that by using different permutations in RCubeRouting we can obtain different paths. For example, in an $RCube(4,1,1)$ in Fig. 2, we want to find a path from server 00 to 11. By using a permutation as [1, 0], we will find a path {00, 01, 11}, while we will find another path {00, 10, 11} by applying a different permutation which is [0, 1]. Thus there are several parallel paths between a pair of source and destination servers. Next, we show this feature by the following theorems and corollaries.

Theorem 2. *Given that two servers $A = a_k a_{k-1} \dots a_0$ and $A' = a'_k a'_{k-1} \dots a'_0$, where $a_0, a'_0 \in \{0,1,\dots,n-1\}$, are different in every digit, $a_i \neq a'_i$ and $i \in \{0,1,\dots,k\}$, by using two different permutations $\Pi_0 = [i_0, (i_0-1) \bmod (k+1), \dots, (i_0-k) \bmod (k+1)]$ and $\Pi_1 = [i_1, (i_1-1) \bmod (k+1), \dots, (i_1-k) \bmod (k+1)]$, where $i_0 \neq i_1$ and $i_0, i_1 \in \{0,1,\dots,k\}$, RCubeRouting generates two parallel paths between the source and destination servers.*

Proof. First, we show that servers in path P_0 generated by Π_0 cannot appear in path P_1 generated by Π_1. In RCubeRouting, a path between A and A' must go through k intermediate nodes, or servers. Thus, denote P_0 and P_1 as $\{A, N_1^0, N_2^0, \dots, N_k^0, A'\}$ and $\{A, N_1^1, N_2^1, \dots, N_k^1, A'\}$ respectively, where N_i^j represents an intermediate node in path P_j after i hops from server A. RCubeRouting corrects 1 digit difference between source and destination servers on the π_i^{th} digit of their addresses. Thus an intermediate node N_i^0 cannot appear in path P_1 with a distance to server A other than i hops. In other words, $N_i^0 \neq N_j^1$ when $i \neq j$, as in this case N_i^0 has i digit difference to server A in their addresses while N_j^1 has j digit difference to server A. Next, node N_i^0 cannot appear in path P_1 with i hop distance to server A, $N_i^0 \neq N_i^1$, either. This is because that A arrives at N_i^0 in path P_0 by correcting digits at positions $i_0, (i_0-1) \bmod (k+1), \dots, (i_0+1-i) \bmod (k+1)$ of the address of server A, but A arrives at N_i^1 in path P_1 by correcting digits at positions $i_1, (i_1-1) \bmod (k+1), \dots, (i_1+1-i) \bmod (k+1)$ of the address of server A. Since $i_0 \neq i_1$, there must be at least one digit difference between N_i^0 and N_i^1, thus $N_i^0 \neq N_i^1$. Similarly, intermediate node N_i^1 cannot appear in path P_0 either.

Next, denote the switch in path P_j at position i as S_i^j. Since in each iteration, RCubeRouting corrects one digit difference via a switch assigned by π_i, any switch S_i^j cannot appear in path P_j more than once. We now show that switches in path P_0 cannot appear in path P_1. Assume switch S_i^0 also appears in path P_1, $S_i^0 = S_j^1$ and denote A and A' as N_0^0 and N_{k+1}^0 in path P_0 and N_0^1 and N_{k+1}^1 in path P_1 respectively. Thus, servers N_i^0, N_{i+1}^0, N_j^1 and N_{j+1}^1,, $i, j \in \{0,1,\dots,k\}$, are connected by switch S_i^0. As a result, due to the topological properties of RCube, the addresses of these four servers only differ in the π_i^{th} digit a_{π_i}. However, in this case, RCubeRouting will correct a_{π_i} to a'_{π_i}. As a result, N_{i+1}^0 and N_{j+1}^1 should be the same node, which contradicts with the assumption above. Thus switches in path P_0 cannot appear in path P_1.

Similarly, switches in path P_1 cannot appear in path P_0 either. □

Corollary 1. *Given that two servers $A = a_k a_{k-1} \dots a_0$ and $A' = a'_k a'_{k-1} \dots a'_0$, where $a_0 \in \{0, 1, \dots, n-1\}$ and $a'_0 \in \{n, n+1, \dots, n+m-1\}$, are different in every digit of addresses between A and any representation form of A', $a'_k a'_{k-1} \dots a'_{\ell+1} a'_0 a'_\ell \dots a'_1$ and $\ell \in \{0, 1, \dots, k\}$, by using two different permutations $\Pi_0 = [i_0, (i_0 - 1) \bmod (k+1), \dots, (i_0 - k) \bmod (k+1)]$ and $\Pi_1 = [i_1, (i_1 - 1) \bmod (k+1), \dots, (i_1 - k) \bmod (k+1)]$, where $i_0 \neq i_1$ and $i_0, i_1 \in \{0, 1, \dots, k\}$, RCubeRouting generates two parallel paths between A and A'.*

Proof. Denote the two paths generated by RCubeRouting separately using Π_0 and Π_1 as P_0 and P_1 respectively. Similar to the proof for Theorem 2, we can show that the intermediate servers in P_0 cannot appear in path P_1 and vice versa. Next we show that switches in path P_0 cannot appear in path P_1. In each iteration i, RCubeRouting corrects one digit difference via a switch specified by π_i, thus any switch S_i^j cannot appear in path P_j more than once. Next, suppose switch S_i^0 in P_0 also appears in path P_1, then there will be four different intermediate nodes connected by switch S_i^0, denoted as N_i^0, N_{i+1}^0, N_j^1 and N_{j+1}^1. By the construction principle of RCube described earlier, these four nodes should have one digit different in their addresses. To be more specific, by representing their addresses in the form of $a_k a_{k-1} \dots a_0$, these four nodes only differ in digit $a_{(i_0-i) \bmod (k+1)}$. However, due to the process of RCubeRouting, before correcting $a_{(i_0-i) \bmod (k+1)}$, node N_i^0 has corrected $(i_0 - (i_0 - i)) \bmod (k+1)$ digits from the source server and node N_j^1 has corrected $(i_1 - (i_0 - i) \bmod (k+1)$ digits from the source server. Since $i_0 \neq i_1$, N_i^0 and N_j^1 differ in more than one digits from each other, which contradicts with the assumption. Thus switches in path P_0 cannot appear in path P_1. Similarly, switches in path P_1 cannot appear in path P_0 either. □

Corollary 2. *Given two servers $A = a_k a_{k-1} \dots a_0$ and $A' = a'_k a'_{k-1} \dots a'_0$, where $a_0, a'_0 \in \{n, n+1, \dots, n+m-1\}$, and two permutations $\Pi_0 = [i_0, (i_0 - 1) \bmod (k+1), \dots, (i_0 - k) \bmod (k+1)]$ and $\Pi_1 = [i_1, (i_1 - 1) \bmod (k+1), \dots, (i_1 - k) \bmod (k+1)]$, where $i_0, i_1 \in \{0, 1, \dots, k\}$ and $i_0 \neq i_1$, such that $a_k a_{k-1} \dots a_{i_0+1} a_0 a_{i_0} \dots a_1$ is different in every digit from $a'_k a'_{k-1} \dots a'_{i_0+2} a'_0 a'_{i_0+1} \dots a'_1$ and $a_k a_{k-1} \dots a_{i_1+1} a_0 a_{i_1} \dots a_1$ is different in every digit from $a'_k a'_{k-1} \dots a'_{i_1+2} a'_0 a'_{i_1+1} \dots a'_1$, by using Π_0 and Π_1, RCubeRouting generates two parallel paths between the source and destination servers.*

Corollary 2 can be easily proved by a similar proof to Corollary 1, and we omit it here. By Theorem 2 and its corollaries, we have the following theorem.

Theorem 3. *There are $k+1$ parallel paths between any pair of servers in an RCube(n, m, k).*

Theorem 3 can be shown by constructing such $k+1$ parallel paths, which can be built by the algorithm provided in Table 2. It can be observed that for any pair of servers, paths built by Table 2 can be divided into two groups: one by DirectRouting and the other one by IndirRouting. Let $h(A, A')$ be the number of digits different between addresses of A and A'. Then if both source and destination servers are core servers, there are $h(A, A')$ paths generated by DirectRouting and $k+1-h(A, A')$ paths constructed by IndirRouting. Otherwise, all paths are generated by IndirRouting. Clearly, the $h(A, A')$ paths from DirectRouting are parallel to each other by Theorem 2.

Next, we show that paths generated by IndirRouting are parallel to each other. Suppose one path is built from the ith digit, which means that $a_i = a'_i$ or at least one of A and A' is an edge server. Based on the procedure, all the intermediate nodes, denoted as $B = b_k b_{k-1} \dots b_0$, in this path generated by IndirRouting have the ith digit different from A and A', that is, $b_i \neq a_i$ and $b_i \neq a'_i$. Similarly, for another path where $a_j = a'_j$ or at least one of the A and A' is an edge server,

TABLE 2
Algorithm for Finding $k+1$ Parallel Paths Between a Pair of Source and Destination Servers

```
Input: source A = a_k a_{k-1} ... a_0 and destination
A' = a'_k a'_{k-1} ... a'_0
Output: the k+1 parallel paths between A and A'.
RCubeParaPaths(A, A')
    pathSet = { };
    for (i = 0; i ≤ k; i++)
      if A[0] ≥ n //change source address format
        source[i] = A[0];
        for (j = 0; j < i; j++)
          source[j] = A[j + 1];
        for (j = i + 1; j ≤ k; j++)
          source[j] = A[j];
      else
        source = A;
      if A'[0] ≥ n
        destin[i] = A'[0];
        for (j = 0; j < i; j++)
          destin[j] = A'[j + 1];
        for (j = i + 1; j <= k; j++)
          destin[j] = A'[j];
      else
        destin = A';
      if source[i] ≠ destin[i] && source[i] < n &&
      destin[i] < n
        P_i = DirectRouting(A, A', i);
      else
        P_i = IndirRouting(A, A', source, i);
      add P_i to pathSet;
    return pathSet;
IndirRouting(A, A', source, i)
    path = {A};
    tmp_node = source;
    while(true)
      value = random() mod n;
      if value ≠ a_i && value ≠ a'_i
      tmp_node[i] = value;
      break;
    idx = k;
    for (j = i − 1; j ≤ i − k − 1; j- -)
      Π[idx] = j mod (k + 1);
      idx- -;
    path += RCubeRouting(tmp_node, A', Π);
    return path;
DirectRouting(A, A', i)
    idx = k;
    for (j = i; j ≤ i − k; j- -)
      Π[idx] = j mod (k + 1);
      idx- -;
    return RCubeRouting(A, A', Π);
```

$b_j \neq a_j$ and $b_j \neq a'_j$. Since $i \neq j$, there is at least one digit different from intermediate nodes in the first path to those in the second path and vice versa. Thus nodes in one path constructed by IndirRouting cannot appear in another path generated by IndirRouting. For the switches, since nodes from two different paths have at least two digit differences in their addresses, b_i and b_j, based on the construction rule of RCube, switches in one path cannot appear in another path.

Finally, we show that paths generated by IndirRouting are also parallel to paths generated by DirectRouting. This can be shown by the same proof discussed above. Therefore, Theorem 3 holds.

Using $RCube(4,1,2)$ as an example, by applying function RCubeParaPaths, there are three parallel paths between servers 000 and 014, which are {000, 002, 012, 014}, {000, 020, 021, 014} and {000, 200, 201, 014}, respectively.

Since component failure occurs more often as the size of DCNs becomes larger, graceful degradation plays an important role in today's DCNs. The abundance of parallel paths between any pair of servers in RCube guarantees excellent graceful degradation performance in the presence of component failure, such as link failure and server failure. This makes RCube a very promising structure for today's DCN products.

Finally, we analyze the average length of paths between any pair of servers by applying RCubeRouting.

Theorem 4. *The average path length between any pair of servers within $RCube(n,m,k)$ using RCubeRouting is*

$$\frac{2m}{n+m}+\frac{n(n-1)(k+1)}{(n+m)^2}+\frac{2(n-1)mk}{(n+m)^2}+\frac{m^2(n-1)(k-1)}{(n+m)^2 n}.$$

Proof. The paths between any pair of servers within $RCube(n,m,k)$ are categorized into four groups: core server to core server, core server to edge server, edge server to core server and edge server to edge server.

For paths between core servers and core servers, RCubeRouting is merely finding a way to correct every different digit between addresses of source and destination servers. Assume i digits are different, then the path length between them is i. There are $\binom{k+1}{i}$ cases of assigning i different digits in the $(k+1)$-tuple address. In each of the i corresponding positions, there exist $n-1$ choices different from the source address. Thus the total path lengths from a single core server to all other core servers are $\sum_{i=1}^{k+1} i\binom{k+1}{i}(n-1)^i$.

For paths between core servers and edge servers, RCubeRouting will present the edge server to a proper address format first. Note that for an edge server, addresses $a_k a_{k-1}\ldots a_{i+1}a_0 a_i \ldots a_1$, where $i \in \{0,1,\ldots,k\}$, represent the same server. Thus for the total path lengths from a single core server to all edge servers, we have

$$\frac{\sum_{i=1}^{k+1} i(k+1)\binom{k}{i-1}m(n-1)^{i-1}}{k+1}=\sum_{i=1}^{k+1} i\binom{k}{i-1}m(n-1)^{i-1}.$$

For paths between edge servers and core servers, by RCubeRouting, the total path lengths from each core server to each edge server equal those from each edge server to each core server. Thus we have $\frac{n}{m}\sum_{i=1}^{k+1} i\binom{k}{i-1}m(n-1)^{i-1}$ for the total path lengths from an edge server to all core servers.

Finally, for paths between edge servers and edge servers, since the $k+1$ address formats represent the same server and RCubeRouting adopts a random permutation, the average paths from a single edge server to all other edge servers will be $\sum_{i=2}^{k+1} i\binom{k-1}{i-2}mn(n-1)^{i-2}$.

Thus, in total, we obtain the average path length from any server to any other server as follows:

$$\begin{aligned}
&\frac{1}{(m+n)n^k}\left\{\left[\sum_{i=1}^{k+1} i\binom{k+1}{i}(n-1)^i+\sum_{i=1}^{k+1} i\binom{k}{i-1}m(n-1)^{i-1}\right]n^{k+1}\right.\\
&\quad\left.+\left[\frac{n}{m}\sum_{i=1}^{k+1} i\binom{k}{i-1}m(n-1)^{i-1}+\sum_{i=2}^{k+1} i\binom{k-1}{i-2}mn(n-1)^{i-2}\right]mn^k\right\}\\
&=\frac{1}{(m+n)n^k}\left\{\left[(k+1)n^k(n-1)+mk(n-1)n^{k-1}+mn^k\right]n^{k+1}\right.\\
&\quad\left.+\left[(n-1)kn^k+n^{k+1}+m(k-1)(n-1)n^{k-1}+2mn^k\right]mn^k\right\}\\
&=\frac{2m}{n+m}+\frac{n(n-1)(k+1)}{(n+m)^2}+\frac{2(n-1)mk}{(n+m)^2}+\frac{m^2(n-1)(k-1)}{(n+m)^2 n}.
\end{aligned}$$

□

Theorem 4 indicates that the average path length in RCube also increases linearly to the network order k. Next, we discuss the energy efficiency of deploying an RCube.

3 Power Consumption

Power consumption plays a critical role in designing a DCN. Compared to the one-time expenditure for construction, the nationwide DCNs cost 100 billion kWh power at a cost of $7.4 billion in 2011 [5], which is why DCNs are usually built in a place where the energy price is low. In a DCN, the interconnection network, which includes switches, line cards and NICs, occupies 10-20 percent of the total power consumption [19]. Thus, it is desirable to reduce the power consumption of interconnection networks [20] in data centers. In this section, we study the power consumption of RCube in the perspective of both its inborn property and exterior policy.

3.1 Power Saving Due to Inherent Properties of RCube

A typical approach to increasing the power efficiency of interconnection networks is to turn off the unused ports. If all the ports within a switch are idle, turn off the entire switch. Thus the consumed power can be in proportion to the traffic load. However, due to the fact that there are fixed overheads in switches such as fans and switch chips, turning off ports leads to only negligible power saving. Report in [19] indicates that over 75 percent of the total power consumption in a switch is generated by the overhead. Thus turning off a switch yields the most energy benefits, while turning off an unused port saves only 1-2 Watts [19].

We have studied the power consumption of switches provided by some well-known vendors [1], [2], [3], and some representative results are summarized in Table 3, which shows that although switches with different numbers of ports have different power consumption, the power consumption for the fabric overhead part in each switch is relatively fixed given that the numbers of ports are in the same order of magnitude. Hence, two 48-port switches consume

TABLE 3
Power Consumption of Switches

Vendor	Model	No. of ports	Power	
			Typical	Max.
Cisco	Nexus 5548UP	up to 48	390 W	600 W
	Nexus 5596UP	up to 96	660 W	882 W
Arista	7050TX-48	up to 48	305 W	-
	7050TX-64	up to 64	315 W	-
Huawei	CE5855-24T4S2Q-EI	24	-	≤ 90 W
	CE5855-48T4S2Q-EI	48	-	≤ 130 W

much more power than a 96-port switch. As a result, when designing a DCN with a target size, it is preferable to keep the total number of interconnection switches as small as possible. Given that BCube and RCube have the same size with the same network order, without using any power aware routing algorithms, the interconnection part of RCube naturally consumes less power than that of BCube. Note that given RCube and BCube under the same network order and similar network size, the difference of power consumption of NICs, including both the per port power consumption and the overhead, on servers in these two structures should be trivial and negligible, due to similar traffic load. Hence, we focus on the power consumed by the interconnected switches. Denote the power consumption of a switch at time t as $P(t)$, the power consumption of the overhead including fans and fabrics as p_0 and the power consumption caused per port as p_1. Then we have

$$P(t) = p_0 + i(t)p_1, \tag{4}$$

where $i(t)$ represents the number of active ports in a switch, or ports not in idle state, at time t.

Suppose we have a $BCube(n_1, k)$ and an $RCube(n_2, m, k)$ with the same network size $n_1^{k+1} = (n_2 + m)n_2^k$. The numbers of switches used in each structure are $(k+1)n_1^k$ and $(k+1)n_2^k$, respectively. Denote the ratio, or percentage, of working switch ports to the total number of switch ports in each structure as r_1 and r_2 respectively. Then the ratio, $R(n_1, n_2, m, k)$, of power consumption of the interconnection network in $RCube(n_2, m, k)$ to that in $BCube(n_1, k)$ is

$$\begin{aligned} &R(n_1, n_2, m, k) \\ &= \frac{p_0 \times (k+1)n_2^k + r_2 p_1 \times (k+1)(n_2+m)n_2^k}{p_0 \times (k+1)n_1^k + r_1 p_1 \times (k+1)n_1^{k+1}} \\ &= \frac{p_0}{p_0 + r_1 p_1 n_1}\left(\frac{n_2}{n_2+m}\right)^{\frac{k}{k+1}} + \frac{r_2 p_1 n_1}{p_0 + r_1 p_1 n_1}. \end{aligned} \tag{5}$$

Due to the similarity between $BCube(n_1, k)$ and $RCube(n_2, m, k)$, given similar traffic load, the number of active ports in each structure should be similar. Hence, $r_1 \approx r_2$, denoted as r. Then we have

$$\lim_{k \to +\infty} R(n_1, n_2, m, k) = \frac{p_0}{p_0 + r p_1 n_1}\,\frac{n_2}{n_2+m} + \frac{r p_1 n_1}{p_0 + r p_1 n_1}. \tag{6}$$

Eq. (6) shows that without using any power saving scheme, for a similar network size, RCube behaves more power efficiently than BCube, as the network order k becomes larger. Based on the data in [19], over 75 percent of the total power in a switch is consumed by the fixed overhead. In other words, $\frac{p_0}{p_0 + n_1 p_1} \geq 75\%$. Thus when $m = n_2$, $RCube(n_2, m, k)$ saves about 37.5-50 percent energy compared to $BCube(n_1, k)$ depending on the traffic load. Note that we use a simplified power model to analyze the power efficiency. However, the conclusion that RCube is more power efficient than BCube, with respect to interconnect network, still holds as long as the overhead part consumes a fixed amount of energy and it dominates the power consumption in a switch.

In addition, compared to BCube, RCube can provide more flexibility in design choices. For example, suppose we have switches with α ports. BCube can build a network with size α^{k+1}. As a result, we can only modify the network size by adjusting k. On the other hand, RCube can construct a network with size $\alpha(\alpha - m)^{k+1}$. Thus we can have more choices for network size by fine tuning both k and m. It is true that we can also build an incomplete BCube, or partial BCube, to meet the network size requirements [6]. For example, $BCube(8, 3)$ consists of 4,096 servers and the target size is 2,000. This goal can be achieved by constructing a partial $BCube(8, 3)$ using only 4 $BCube(8, 2)$s which consist of 2,048 servers. However, this solution has some drawbacks. Based on the discussion in [6], though BCube can be partially constructed, the interconnection structure of switches should be kept the same as the complete BCube so that its throughput will not be impacted too much and future expansion can be easily fulfilled. Unfortunately, these switches inevitably increase the capital expenditure and power consumption. On the contrary, using the same switches and servers, $RCube(6, 2, 3)$ has a similar size, which holds 1,728 servers, and the same network diameter, while only consuming 864 8-port switches, much less than 2,048 8-port switches in the partial $BCube(8, 3)$, leading to significant energy saving.

3.2 Power Consumption Reduction Due to External Policies

In this section, we further reduce power consumption of RCube by systematically turning off irrelevant switches depending on real-time traffic load. As introduced in the introduction section, PCube greatly improves power efficiency based on BCube. Since BCube is a special case of RCube, we will show in the following that PCube scheme can also be deployed to RCube, denoted as PRCube.

In addition to the existing three parameters, the number of core servers n, the number of edge servers m and the network order k, one more parameter q is needed to define the number of NIC ports in each server. Hence, denote PRCube as $PRCube(n, m, k, q)$. Clearly, $q \leq k + 1$. PRCube is also a recursively defined structure, and the base PRCube structure is $PRCube(n, m, k, 2)$. Specifically, compared to $RCube(n, m, k)$, $PRCube(n, m, k, 2)$ houses exactly the same number of servers, while the switches available on the s_kth level in $PRCube(n, m, k, 2)$ are those restricted by the following rules:

$$\begin{cases} s_k s_{k-1} \ldots s_0, \text{ if } s_k = 0 \\ s_k s_{k-1} \ldots s_{s_k} e_{s_{k-1}} o_{s_{k-2}} \ldots o_0, \text{ if } s_k \in \{1, \ldots, k-1\} \\ s_k s_{k-1} o_{k-2} \ldots o_0, \text{ if } s_k = k. \end{cases} \tag{7}$$

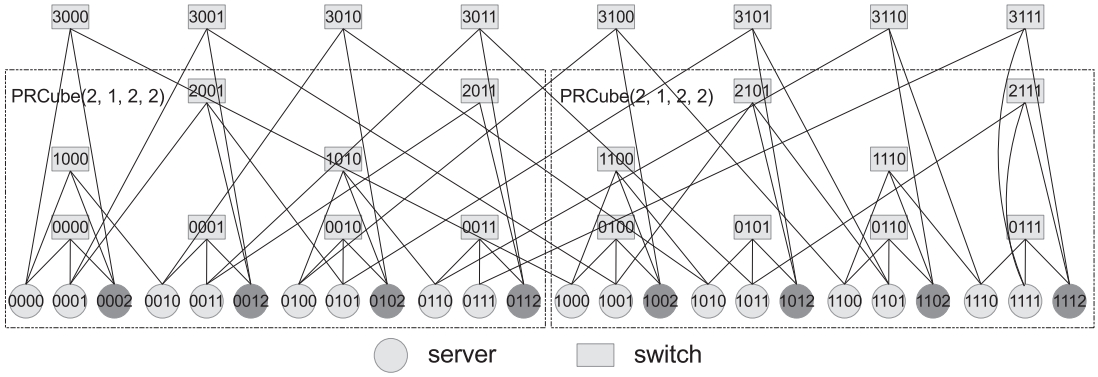


Fig. 4. Example of $PRCube(2,1,3,3)$, which is composed of two $PRCube(2,1,2,2)$s and eight switches.

In the above constraint, e_i stands for an even number in the ith digit of switch address, while o_i represents an odd number in the ith digit. An example of $PRCube(2,1,2,2)$ is shown in the left rectangle in Fig. 4, which is composed of 12 servers. In the first level, $s_k = 0$, all the four switches, 0000, 0001, 0010 and 0011, are kept. However, in the second level, $s_k = 1$, only switches with s_0 being even numbers are kept, namely, 1000 and 1010. Similarly, in the third level, also the top level of $PRCube(2,1,2,2)$ where $s_k = 2$, only switches with s_0 being odd numbers are left, which are switches 2001 and 2011.

Provided this, a $PRCube(n,m,k,q)$ is constructed with n $PRCube(n,m,k-1,q-1)$s and n^k $(n+m)$-port switches. The connection between servers and switches follows the same rules as constructing RCube. Fig. 4 also demonstrates an instance of $PRCube(2,1,3,3)$, which consists of 2 $PRCube(2,1,2,2)$s and 8 switches. The construction process of PRCube implicitly infers that the structure of $PRCube(n, m, k, q)$ is a sub-graph of $RCube(n,m,k)$. However, servers in $PRCube(n,m,k,q)$ requires q NIC ports, independent of network order k, while servers in $RCube(n,m,k)$ are fixed and equipped with $k+1$ NIC ports. Moreover, although most properties of original RCube are kept in PRCube, some critical ones change slightly. In particular, instead of $k+1$ near-equal parallel paths, there are only q parallel paths, whose lengths diverse greatly. This will increase traffic congestion in PRCube, which further lowers the aggregate throughput, as shown in Section 5 later.

Meanwhile, compared to $RCube(n,m,k)$, $PRCube(n, m, k, q)$ uses fewer switches. Given that $PRCube(n,m,k,q)$ and $RCube(n,m,k)$ hold the same number of servers, naturally, $PRCube(n,m,k,q)$ further reduces power consumption without deploying any power-aware routings. To be specific, let $R(q)$ be the ratio of power consumption of the interconnecting network in $PRCube(n,m,k,q)$ to that in $RCube(n,m,k)$. It can be easily shown that there are qn^k switches in $PRCube(n,m,k,q)$. Then

$$R(q) = \frac{p_0 \times qn^k + rp_1 \times q(n+m)n^k}{p_0 \times (k+1)n^k + rp_1 \times (k+1)(n+m)n^k}. \quad (8)$$

Different from the case in the previous section, where the same traffic pattern would result in different routings in RCube and BCube respectively, even though they house a similar number of servers, in the case above, the same traffic could result in exactly the same routing in both RCube and PRCube by leaving the switches in RCube that are not used in PRCube, in idle state. Therefore, Eq. (8) can be converted as follows:

$$R(q) = \frac{p_0 \times qn^k + rp_1 \times q(n+m)n^k}{p_0 \times (k+1)n^k + rp_1 \times q(n+m)n^k}. \quad (9)$$

Similarly, given that $\frac{p_0}{p_0+(n+m)p_1} \geq 75\%$, the expression above could be simplified to

$$R(q) = \frac{q + rq\frac{(n+m)p_1}{p_0}}{k+1+rq\frac{(n+m)p_1}{p_0}} \approx \frac{(1+0.33r)q}{k+1+0.33rq}. \quad (10)$$

Hence, the power efficiency of PRCube, compared to RCube, totally depends on the traffic load. In case of light traffic load, smaller q would be sufficient, which could achieve the best power reduction. On the other hand, in case of heavy traffic load, which requires more throughput bandwidth, we could choose a larger q. Even in the case where $q = k+1$, the power efficiency of PRCube would be equal to that of RCube. Hence, the power consumed by the interconnect switches in PRCube is no more than that in RCube. Therefore, deploying PCube scheme into RCube greatly benefits the power efficiency of RCube.

Next, we discuss and analyze the availability of RCube by deploying a redundancy scheme.

4 Availability and Failover in RCube

Many data centers today have extremely stringent availability requirements. However, as the size of the DCNs gets larger and larger, it becomes more and more challenging to meet this target. For example, suppose the failure rate of a server is once every 10 years on average, the mean time to failure, or MTTF, for a DCN with a size of 1,000, is 3.65 days. When the size of a DCN reaches 10,000, its MTTF will sharply reduces to 8.76 hours. In other words, there will be around 3 servers malfunctioning each day in a DCN with a size of 10,000. In addition, in [21], it is stated that high availability cannot be achieved by merely installing failover software and walking away. Thus to achieve the near-perfect service availability, many DCN implementations deploy server redundancy to eliminate the impact of a single node failure [4].

Traditionally, the redundancy scheme can be divided into two categories: active-passive or asymmetric mode, and active-active or symmetric mode. In the active-passive mode, one server is active and running all the tasks, while its partner server is a dedicated standby server and goes

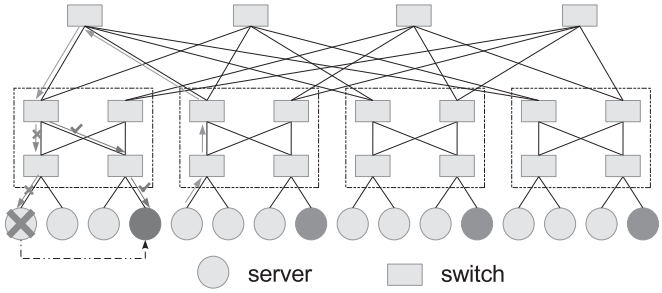


Fig. 5. Failover in FatTree. In the case where server marked with cross fails and the server in dark gray in the same pod warms up and takes all the jobs left by the failed one. All the remaining servers will not be affected due to the failure.

back to work only when the active node fails. On the contrary, in the active-active mode, both servers are running independent tasks and if any one fails, the other will fully take over the tasks the failed server is working on before it goes down. The redundancy scheme can also be divided to $1+1$ model and $N+M$ model. In $1+1$ model, each server has its own dedicated backup server in standby mode, while in $N+M$ model, every N servers are covered by M backup servers. One drawback of the active/passive mode is its cost. In particular, in the $1+1$ redundancy model, making one server not working and standby for every other server doubles the capital expenditure. On the other hand, $N+M$ model can help lower the cost. Besides, for some critical applications, high availability is the priority. For those applications and structures, it is reasonable to trade off some extra cost for high availability [4].

Redundancy technique has been commonly applied to switch-centric networks, see, for example, [10], [11], as putting some servers into standby mode will not affect the performance and behavior of the remaining servers. One example of failover in a switch-centric network is depicted in Fig. 5, where circles in dark gray are redundant servers used as backups. It shows that making a failover from the failed server to the backup one will not interfere the behavior of other servers. On the contrary, due to their special properties, usually server-centric networks cannot adopt the active-passive redundancy scheme. The nature of server-centric networks needs servers to act as the relay nodes and participate in the routing task. Thus, making some servers into standby mode will inevitably impact the performance of other working servers, including bisection bandwidth, etc.

Using $BCube(4,1)$ in Fig. 1 as an example, suppose we choose server 03 to be the backup server and set it to standby mode. Then the original two parallel paths between servers 00 and 33, one of which goes through server 03, will reduce to a single path. Moreover, suppose servers 03, 13, 23 and 33 are all in standby mode separately for the four $BCube(4,0)$s, then switch 13 is completely idle. If we do not put it into sleep, it will consume a tremendous amount of energy. (In fact, as discussed earlier, even a switch in an idle state still consumes a considerable amount of energy.) If we put it to sleep, the standby servers could not provide full parallel paths. For example, if server 00 is dead and server 03 is back to work, since switch 13 is sleeping, server 03 can only relay packets via switch 00, which sharply reduces the network throughput and reliability. Even after we wake

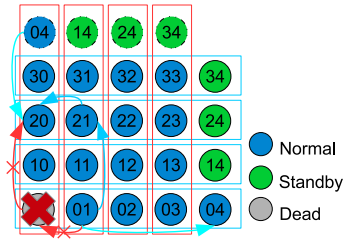
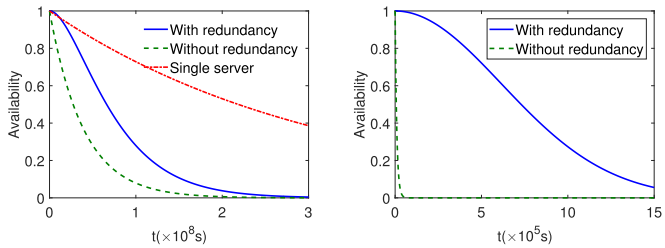


Fig. 6. Failover: Server 00 crashed and server 04 warms up and takes over the tasks left by server 00.

up switch 13, this problem remains, because all the other servers connected by switch 13 are still in the standby mode.

It is possible to install special purpose NICs, which have routing ability, to these standby servers, so that even when their main computing components are sleeping, their NICs remain working and relaying packets for others. However, this will drastically increase capital expenditure and power consumption. On the other hand, RCube provides a cost-efficient solution that satisfies the redundancy requirement while keeping power consumption relatively low. Since we have discussed the power consumption issue in the previous section, in this section, we focus on the redundancy property. If we are designing a DCN with $N+M$ redundancy mode, we can choose $RCube(n,m,k)$, where $n=N$ and $m=M$, and set edge servers into standby mode. The active and passive server pair and backup scheme are implemented in $RCube(n,m,0)$s of the structure. Fig. 6 provides an example of $RCube(4,1,1)$ within which $4+1$ redundancy model is adopted and all edge servers are put into standby mode. It can be observed that without edge servers, core servers can still communicate with each other and all the properties, including diameter and $k+1$ parallel paths, still hold. Therefore, making edge servers standby will not affect the performance of core servers. When some core servers fail, the corresponding edge servers which belong to the same $RCube(n,m,0)$ will take over the jobs the failed servers were working on. Fig. 6 also shows a failover example, where server 00 goes down and then server 04 is back to working mode and taking over the jobs, including both computing jobs and packet relaying jobs, from server 00. All the remaining servers continue to work on their own jobs with only a little alteration: when sending or relaying packets, which should go to server 00, now they send to server 04 and let server 04 do the routing. In addition, all the normal servers still have $k+1$ parallel paths. For example, the original two parallel paths between servers 01 and 20 are {01, 00, 20} and {01, 21, 20}. Since 00 is not working and 04 takes the place, server 01 can reset its first path to {01, 04, 20} which is also parallel to the second path. Meanwhile, the newly coming server 04 also has $k+1$ parallel paths to any other working servers, which has been proved by Theorem 3. To the best of our knowledge, this is the feature that other existing server-centric networks cannot achieve. We can gossip this failover event to all the remaining servers, which can be easily fulfilled, so we ignore the technical details here.

In addition, $RCube(n,m,k)$ can also be set to normal working mode without standby servers, in which each server independently works on its own job. This increases computing resource while having a negligible impact on the



(a) Availability of $RCube(8,1,0)$. (b) Availability of $RCube(8,1,4)$.

Fig. 7. Availabilities of the system with and without redundancy.

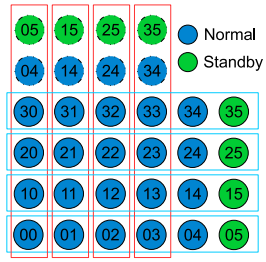


Fig. 8. Structure of $RCube(4,2,1)$ with $5+1$ redundancy.

interconnection power consumption as we described in previous sections. Thus RCube provides more options for DCN products and can be dynamically configured between fully working mode and active/passive mode based on customers' requirement on DCN availability.

Next, we theoretically analyze the availability of RCube. Let $R(t)$ denote the probability of a server working normally at time t. We assume that the failure of each server occurs independently from others and it follows exponential distribution, where $R(t)=e^{-\lambda t}$. Thus the probability of that server fails at time t is

$$f(t)=\frac{d(1-R(t))}{dt}=\lambda e^{-\lambda t}. \tag{11}$$

Next, consider the probability of the whole network of $RCube(n,m,k)$ in the active/passive mode, where all edge servers are set to standby mode, working normally at time t. As discussed earlier, when core servers have problems, such as system crash or hardware failure thus cannot work, the corresponding edge servers will warm up and take over the jobs left by the failed servers. Therefore, when the number of failed servers in each $RCube(n,m,0)$ of $RCube(n,m,k)$ is no greater than m, the single $RCube(n,m,0)$ can still work normally. Suppose i core servers independently fail at time $t_0,t_1,\ldots,t_{i-1}$, with failure probabilities of $f(t_0)$, $f(t_1),\ldots,f(t_{i-1})$, respectively, where $0\le i\le m$ and $0\le t_0\le t_1\le\cdots\le t_{i-1}\le t$, and i edges servers warm up immediately right after the corresponding core server fails. To ensure that the system is still working at time t, all the remaining $n-i$ core servers should be still working and the i backed up servers should also be working. Thus the probability that there are still n servers working at time t during which i original servers fail in an $RCube(n,m,0)$ can be expressed by the following equation:

$$\begin{aligned}R_i(t)&=\int_{t_{i-1}}\cdots\int_{t_0}\binom{n}{i}\prod_{j=0}^{i-1}f(t_j)R(t)^{n-i}\prod_{j=0}^{i-1}R(t-t_j)\prod_{j=0}^{i-1}dt_j\\&=\int_{t_{i-1}}\cdots\int_{t_0}\binom{n}{i}\lambda^i e^{-\lambda nt}\prod_{j=0}^{i-1}dt_j=\binom{n}{i}\lambda^i e^{-\lambda nt}\frac{t^i}{i!}.\end{aligned} \tag{12}$$

Thus, the probability of $RCube(n,m,0)$ surviving to time t is as follows:

$$R_{RCube0}(t)=\sum_{i=0}^{m}R_i(t)=e^{-n\lambda t}\sum_{i=0}^{m}\binom{n}{i}\frac{(\lambda t)^i}{i!}. \tag{13}$$

In particular, when $m=1$, since an $RCube(n,m,k)$ is composed of n^k $RCube(n,m,0)$s, by applying Eq. (13), the availability of an $RCube(n,m,k)$ can be expressed as follows:

$$R_{RCube}(t)=[R_{RCube0}(t)]^{n^k}=(1+\lambda nt)^{n^k}e^{-\lambda n^{k+1}t}. \tag{14}$$

The MTTF of $RCube(n,m,k)$ in the active/passive mode can be expressed by the following equation:

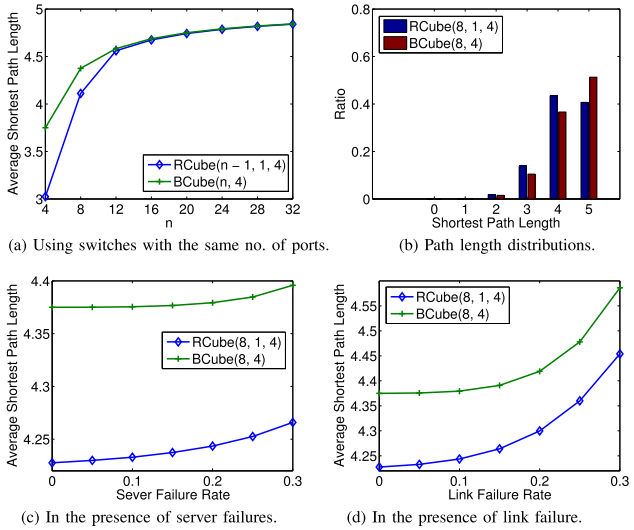
$$\begin{aligned}MTTF_{RCube}&=\int_t tf_{RCube}(t)dt=\int_t t\frac{d(1-R_{RCube}(t))}{dt}dt\\&=\frac{1}{\lambda n^{k+1}}\sum_{i=0}^{n^k}\frac{n^k!}{i!(n^k)^{n^k-i}}=\frac{1}{\lambda n^{k+1}}\frac{n^k!}{(n^k)^{n^k}}\sum_{i=0}^{n^k}\frac{(n^k)^i}{i!}.\end{aligned} \tag{15}$$

By applying Stirling's approximation, we have

$$\begin{aligned}MTTF_{RCube}&=\frac{1}{\lambda n^{k+1}}\frac{\sqrt{2\pi n^k}(\frac{n^k}{e})^{n^k}}{(n^k)^{n^k}}\frac{e^{n^k}}{2}\\&=\frac{\sqrt{\frac{\pi}{2}}}{\lambda n\sqrt{n^k}}.\end{aligned} \tag{16}$$

Fig. 7 shows the probabilities of $RCube(8,1,0)$ and $RCube(8,1,4)$ surviving to time t, respectively, which, together with Eq. (16), illustrates that simply adding one standby server for every n normal servers significantly enhances the availability of the whole system. In the example at the beginning of this section, suppose the DCN with 10,000 servers is constructed by $RCube(10,1,3)$. By applying the active/passive scheme, the MTTF of the system can be significantly increased from original 8.76 hours to 347.19 hours. The cost to achieve this improvement in $RCube(10,1,3)$ is that the capital expenditure has to increase by around 10 percent of the total budget, which is used for the backup servers. However, compared to the significant improvement of the system availability, the extra cost is worthy for two reasons. First, without the backup scheme, the entire system requires higher long term maintenance cost, including human power, since almost 3 servers fail every day. Besides, a high availability of the DCN helps the service provider set up a good reputation which in turn brings enormous potential customers and huge profits. Therefore, it is worth deploying the redundancy scheme.

Moreover, not all edge servers in an RCube need to be configured as standby nodes. In each $RCube(n,m,0)$ of an $RCube(n,m,k)$, we can set some of edge servers into passive mode while others remain in the active mode. As a result, RCube provides better flexibility which improves the power efficiency and availability at the same time. Fig. 8 shows an example, where one of the two edge servers in each



(a) Using switches with the same no. of ports. (b) Path length distributions.

(c) In the presence of server failures. (d) In the presence of link failure.

Fig. 9. Average shortest path length in different cases.

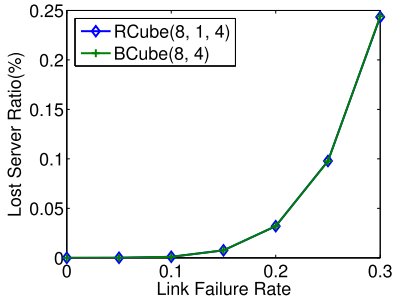


Fig. 10. Percentage of lost servers against link failure.

$RCube(4, 2, 0)$ is set to passive mode and the other one remains active. Following the same rule, when the active edge server fails, the corresponding backup edge server can also warm up and takes over all the tasks left by the failed server without affecting network performance too much.

Besides, PRCube introduced in Section 3 also enjoys a similar high availability to RCube discussed thus far. As the example of $PRCube(2, 1, 3, 3)$ in Fig. 4, suppose all the edge servers marked in dark circles are set to standby mode. All the core servers are working in normal mode and all the properties of the structure, such as diameter and multiple parallel paths, are remained. On the other hand, once some core servers malfunction, server 0000 for instance, the responsible edge servers, server 0002, will return to working mode and take the charge of jobs left by the crashed servers. The entire structure of the remaining network will still have the same properties. This redundancy scheme works in the same way as deployed in RCube. We skip the detailed discussion.

Next, we evaluate the performance of RCube with respect of several key factors through extensive simulations.

5 Performance Evaluation

In this section, we conduct comprehensive simulations to evaluate the performance of RCube. We mainly compare RCube with BCube, as BCube is known to be the most typical server-centric network structure.

5.1 Simulation Setup

For simplicity, we ignore the protocol details and assume the packets are always routed along the shortest available path between the source and destination servers, which is implemented by breadth first search (BFS). By saying a path is available we mean that there is no device failure along that path. We also assume a packet will not be dropped due to congestion control, thus a packet will always arrive at the destination as long as there is at least one available path. All simulations follow the same assumption unless specially noted.

5.2 Evaluation on Topological Properties

We first evaluate the average path length in RCube and BCube built by switches with the same number of ports, n. We set parameters to $RCube(n-1, 1, 4)$ and $BCube(n, 4)$ and vary n from 4 to 32. The results are shown in Fig. 9a. We can see that as n continues to increase, the average path length of $RCube(n-1, 1, 4)$ converges to that of $BCube(n, 4)$, however, the average path length of $RCube(n-1,\ 1, 4)$ is always shorter than $BCube(n, 4)$.

We also examine the detailed path length distribution in $RCube(8, 1, 4)$ and $BCube(8, 4)$, both of which have a diameter of 5. Fig. 9b shows that $RCube(8, 1, 4)$ provides a more balanced distribution than $BCube(8, 4)$. Around 40 percent path lengths are located at 4 and 5 respectively, while more than 50 percent paths in $BCube(8, 4)$ have the longest length.

Next, we evaluate the average lengths of the shortest paths between any pair of servers within RCube in the presence of component failure, including server failure and link failure. The topologies considered are also $RCube(8, 1, 4)$ and $BCube(8, 4)$ which accommodate 36,864 and 32,768 servers respectively. We first run simulations against server failure where we assume each server fails independently from each other with equal failure probability ranging from 0 to 0.3. The result is presented in Fig. 9c, from which we can observe that they have similar performance under server failure. The average path length of $RCube(8, 1, 4)$ grows from 4.228 to 4.266. Similarly, the average path length in $BCube(8, 4)$ rises from 4.375 to 4.396, as server failure ratio increases to 0.3.

Then we conduct simulations to evaluate the average path length under link failure. Similarly, we also assume that each link fails independently and the failure ratio ranges from 0 to 0.3. The results are presented in Fig. 9d, which indicates that $RCube(8, 1, 4)$ still shares similar performance to $BCube(8, 4)$. The average path length in $RCube(8, 1, 4)$ grows from 4.228 to 4.454. On the other hand, in $BCube(8, 4)$ it increases from 4.375 to 4.586.

Moreover, we also run simulations to evaluate the network connectivity under link failure and show the results in Fig. 10. We say that a server is lost if it cannot be reached from the source server. In other words, there are no available paths between the source server and the lost server. Fig. 10 reveals that both $RCube(8, 1, 4)$ and $BCube(8, 4)$ give the same good performance on connectivity. The ratio of the number of lost servers to the network size grows very slowly from 0 to 0.24 percent.

From the simulation results presented above, we can see that both RCube and BCube have excellent performance in the presence of component failures. To be more specific, RCube and BCube have almost the same behavior under both server failure and link failure. This is expected, since RCube and BCube share many properties. In this case,

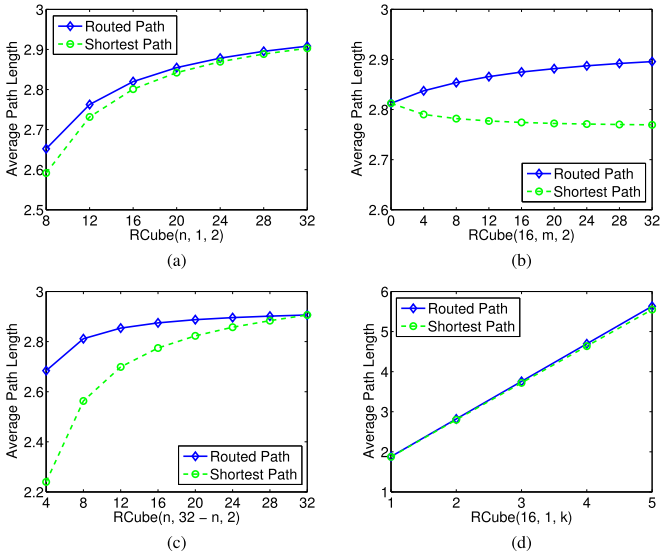


Fig. 11. Comparisons between paths generated by RCubeRouting and the shortest paths.

severs in either $RCube(8, 1, 4)$ or $BCube(8, 4)$ have 5 parallel paths to any other servers, which are unlikely all unavailable at the same time. Thus they are very robust against component failure.

5.3 Impact of RCubeRouting

So far, all the simulations harnesses the shortest paths in RCube. However, as discussed earlier, the overhead for finding a path in a server-centric network should be as low as possible, as each server has to handle tremendous amount of traffic loads for other servers. That is the main motivation for designing RCubeRouting. Next, we study the impact of using RCubeRouting in Table 1 on the path length by comparing paths generated by RCubeRouting with the shortest paths in RCube. To give a detailed evaluation, we separately conduct several simulations by varying each parameter, n, m, k and $n + m$, and present the results in Fig. 11. In Fig. 11a, we set the parameters to $RCube(n, 1, 2)$ where n varies from 8 to 32. It can be observed that the difference between paths generated by RCubeRouting and the shortest path strictly decreases as n grows. On the other hand, Fig. 11b, which uses $RCube(16, m, 2)$ and varies m from 0 to 32, indicates that as m grows, the gap between paths generated by RCubeRouting and the shortest path increases. However, this gap is very small, since even when m reaches 32, the path by RCubeRouting is only 4 percent longer than the shortest path. In Fig. 11c, we investigate the path lengths given that the number of ports in a switch is fixed to 32. Therefore, there are n core servers and $32 - n$ edge servers in an $RCube(n, m, 0)$. It can be observed that the more ports are used to connect core servers, the smaller the gap between the path by RCubeRouting and the shortest path. When all ports are used to connect core servers, where RCube is reduced to BCube, there will be no gap. This indicates that the routing algorithm provided in [6] is also a special case of RCubeRouting when $m = 0$. Then in Fig. 11d, we set the parameters to $RCube(16, 1, k)$ to investigate the impact of k on the path length, from which we can see that the average path length increases nearly linearly to the network order k, as illustrated in Theorem 4, and the difference between the path generated by RCubeRouting and the shortest path increases as k grows. However, the path generated by RCubeRouting has a length only under 4 percent longer than the shortest path, even when k reaches 5. Therefore, in total, Fig. 11 suggests that RCubeRouting gives similar performance to the shortest path with a negligible impact on the path length when n is relatively larger than m. Besides, RCubeRouting finds a path by calculating addresses of source and destination servers digit by digit which consumes only a small amount of computing time, even when the size of RCube is large, compared to finding the shortest path. Thus RCubeRouting is very computation efficient.

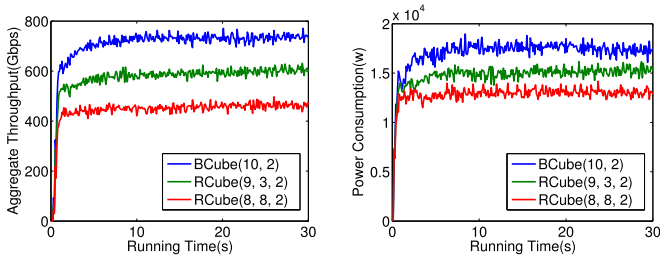
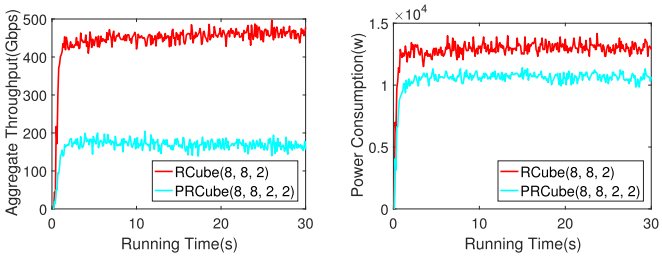


Fig. 12. Aggregate throughput and power consumption of interconnection network under the traffic pattern generated by traffic characteristics.

5.4 Throughput and Power Consumption

Finally, we evaluate the aggregate throughput of RCube. We build a simulator in C++ for the purpose of evaluating the throughput, which uses RCubeRouting in Table 1 to find paths. We set the speed of all links, NIC ports and switch line cards to 1 Gbps and ignore the packet dropping algorithms by setting the buffer length of each link to infinitely large so that the congestion control scheme will not affect the throughput. We generate traffics based on the traffic characteristics for DCNs summarized in [24]. We evaluate the aggregate throughput for $BCube(10, 2)$, $RCube(9, 3, 2)$ and $RCube(8, 8, 2)$ which have a similar network size under the same network order. To be more specific, $BCube(10, 2)$, $RCube(9, 3, 2)$ and $RCube(8, 8, 2)$ consist of 1,000, 972 and 1,024 servers respectively, and meanwhile, they are composed of 300, 243 and 192 switches respectively. The simulation results are presented in Fig. 12a, from which we can see that $BCube(10, 2)$ provides the highest aggregate throughput which is around 730 Gbps, and $RCube(8, 8, 2)$ gives the lowest throughput that is around 470 Gbps. On the other hand, $RCube(9, 3, 2)$ is in the middle, which presents a throughput of 600 Gbps. Clearly, it can also be observed that the throughputs of $RCube(8, 8, 2)$ and $RCube(9, 3, 2)$ are about 65 and 80 percent of that in $BCube(10, 2)$, respectively, which correspond to the number of switches in all these three structures. This result indicates that the more switches used in RCube, the higher throughput RCube will provide.

We also calculate the power consumption generated by the interconnection network of these three structures and present the results in Fig. 12b, in which we use the power consumption model discussed in the previous section and set overhead consumption and per port consumption to be 60 and 2 Watts respectively. For power efficiency, we also assume that any port can be turned off independently if there is no traffic on the corresponding link and a switch can be turned off if there is no traffic going through it.



(a) Aggregate throughput. (b) Power consumption of network.

Fig. 13. Aggregate throughput and power consumption of interconnection network under the traffic pattern generated by traffic characteristics.

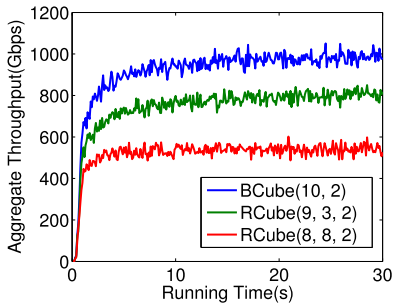


Fig. 14. Aggregate throughput under all-to-all communication pattern.

Fig. 12b shows that $BCube(10, 2)$ consumes most energy whose power stays around 18.5 KWatts. On the contrary, the energy consumed by $RCube(8, 8, 2)$ is significantly reduced to 12.5 KWatts, which is about 65 percent of the energy spent by $BCube(10, 2)$. On the other hand, $RCube(9, 3, 2)$ locates in the middle, which has 15.2 KWatts. These results are expected as they match the discussion on the power consumption in the previous section. Note that the data we are using may not be typical or representative for state-of-art switches. However, this result still holds since the fixed overhead power consumption in a switch acts as the main contributor.

As discussed earlier, RCube can also adopt PCube scheme, denoted as PRCube, to further improve power efficiency. Next, we study the performance of PRCube regarding the aggregate throughput and power consumption as well. To make the comparison consistent with the previous evaluation. We construct PRCube based on $RCube(8, 8, 2)$. Note that the reason why $RCube(9, 3, 2)$ is omitted is that due to the requirement of the construction, the n in $PRCube(n, m, k, q)$ should be an even number. The comparison between $RCube(8, 8, 2)$ and $PRCube(8, 8, 2, 2)$ is also conducted under the same traffic characteristics, and the corresponding throughput results are recorded in Fig. 13a. It can be observed that $RCube(8, 8, 2)$ has much higher throughput than $PRCube(8, 8, 2, 2)$. This is expected for two reasons. First, each server in $RCube(8, 8, 2)$ is equipped with one more NIC port than servers in $PRCube(8, 8, 2, 2)$. More NIC ports enable more capability to transit traffic. In addition, missing switches elongate paths among a considerable number of servers, which in turn exacerbates traffic congestion in some paths. As a result, $PRCube(8, 8, 2, 2)$ gives much worse throughput performance than $RCube(8, 8, 2)$.

On the other hand, the related power consumptions of the interconnected networks for $RCube(8, 8, 2)$ and $PRCube(8, 8, 2, 2)$ are presented in Fig. 13b. Compared to 12.5 KWatts consumed by $RCube(8, 8, 2)$, $PRCube(8, 8, 2, 2)$ costs only around 10 KWatts power, further saving 20 percent energy.

To obtain a more comprehensive view of the throughout, we further investigate the most stressful traffic pattern, all-to-all communication pattern, where every server sends a different packet to each of the remaining servers in the same network. We present the results in Fig. 14, which shows similar outcome to that in Fig. 12. $BCube(10, 2)$ gives the best throughput which achieves 1,000 Gbps, while $RCube(8, 8, 2)$ gives the worst one that only grows to above 620 Gbps. On the other hand, $RCube(9, 3, 2)$ is in the middle and provides a throughput of more than 800 Gbps. Thus the ratio to the throughput of $BCube(10, 2)$ by $RCube(8, 8, 2)$ and $RCube(9, 3, 2)$ are also about 65 and 80 percent respectively, which are the same as the case above.

Note that we have not evaluated the power consumption of the interconnection network in all-to-all pattern. However, the outcome can be easily calculated and should meet the expectation discussed in the previous section, since in this pattern, every switch including every port in it should be working all the time. We also omit the evaluation on $PRCube(8, 8, 2, 2)$ against $RCube(8, 8, 2)$ under all-to-all traffic pattern. This is because that according to PRCube policy, in this case, all internal switches should be turned on to provide more traffic capacity, which turns $PRCube(8, 8, 2, q)$ back to the original $RCube(8, 8, 2)$. As a result, there should be no difference between them. It should also be noted that we use RCubeRouting to generate the path for packets. Though it is efficient, it will make the traffic in RCube unbalanced, as core servers handle more traffic loads than edge servers. Therefore, theoretically, throughput for $RCube(8, 8, 2)$ and $RCube(9, 3, 2)$ can be further improved and potentially the gap between the throughput of $BCube(10, 2)$ and other two structures can be reduced. On the other hand, this potential throughput improvement should not affect the power consumption of the interconnection network much, as long as the overhead consumption still occupies a significant fraction of total power consumption in a switch.

In summary, RCube has similar performance to BCube in many critical metrics, such as diameter, path distribution and graceful degradation, yet it provides more flexibility to make trade-off between interconnection network power consumption and aggregate throughput. Study in [24] implies that most traffic has a strong locality and is confined within the same rack. Thus, it may not be necessary to maintain high aggregate throughput for a practical scenario. On the contrary, power consumption has a key impact on choosing the location of a DCN and also plays an important role in long term operation of the DCN. Moreover, RCube can deploy the redundancy scheme to achieve high availability, which cannot be easily done in BCube. Therefore, RCube is more empirical than BCube taking all these factors into consideration.

6 Related Work and Future Work

BCube is proposed in [6] as a typical server-centric network, providing excellent throughput without the over-subscription problem suffered by a switch-centric network. To reduce the expandability cost of a traditional server-centric network, BCCC and its extension are introduced in [9], [15].

Meanwhile, extensive comparisons among the most typical server-centric network structures are also conducted in [9], [15] respectively as well. In addition, to improve the power efficiency of BCube, PCube scheme is provided in [7] in terms of systematically turning on/off switches in BCube. In light of availability, however, to the best of our knowledge, there is no existing work considering the availability issue from the perspective of topological structure as we study in this paper.

On the other hand, in switch-centric networks, a structure, called Jupiter, which extends Fat-tree structure, has been deployed at Google [13]. In [10], [11], redundancy and failover scheme is introduced into Fat-tree structure for the purpose of improving multicast performance. Regarding to the power consumption, ElasticTree in [19] dynamically manages and consolidates the set of active network elements to satisfy the changing traffic load. CARPO [20] takes advantage of both correlation-aware traffic consolidation and link adaptation to save more power. PowerNetS [25] further improves power efficiency by utilizing correlation among traffic load given the network structure variations caused by VM migrations. AggreFlow [26] introduces two novel routing solutions, denoted as Flow-set Routing and Lazy Rerouting to enhance power efficiency, in the way of achieving both load balance and QoS guarantee.

A comprehensive comparison between server-centric networks and switch-centric networks in terms of construction cost is given in [22]. Moreover, a comparison between them concerning power consumption is given in [27].

Finally, as discussed in Section 3, the power consumption model of the switch studied in this paper is simplified. Hence, the improvement analysis on power efficiency of RCube is not so accurate, although the conclusion that RCube is more power efficient in the interconnect network still holds, as long as the switch overhead plays a major role. We will deploy a more accurate model to conduct more detailed analysis in the future. In addition, since this paper focuses on the power efficiency improvement of RCube compared to BCube, the difference of power consumptions between RCube and a switch-centric network, such as Fat-tree, is not considered in this paper, and we leave it as future work. Another future work is related to availability. In this paper, we only theoretically analyze the availability of RCube under redundancy mode. We plan to build a prototype to empirically evaluate the availability in our future work. Moreover, we will also study more properties of RCube, such as performance under real-time traffic transmission, based on the prototype in the future.

7 Conclusions

In this paper, we propose a cube-based server-centric data center network, called RCube, which is a recursively defined network with many good properties. For the same network size and order, without adopting any power aware routing algorithms, RCube can consume less power than the typical existing server-centric network, BCube. Moreover, RCube can be configured into active/passive redundancy mode to achieve high availability, which cannot be easily implemented by existing server-centric networks. Also, RCube provides more flexibility in network design, by fine tuning some parameters to give more options for practical DCN products. We also present efficient routing algorithms for RCube. Our extensive simulation results show that RCube provides higher availability and flexibility to make trade-off among many factors, such as power consumption and aggregate throughput, than BCube, while delivering similar performance to BCube in many critical metrics, such as average path length, path distribution and graceful degradation. This makes RCube a very promising empirical topology for product data centers.

Acknowledgments

This research work was supported in part by the US National Science Foundation under grant numbers CCF-1526162 and CCF-1717731.

References

[1] Cisco Nexus 5548P, 5548UP, 5596UP, and 5596T Switches. [Online]. Available: http://www.cisco.com/c/en/us/products/collateral/switches/nexus-5000-series-switches/data_sheet_c78–618603.pdf

[2] Arista 7050X Series. [Online]. Available: https://www.arista.com/en/products/7050x-series

[3] CloudEngine CE5800 Series Data Center Switches. [Online]. Available: http://e.huawei.com/en/related-page/products/enterprise-network/switches/data-center-switches/ce5800/brochure/Switch_ce5800

[4] Data Center High Availability Clusters Design Guide. [Online]. Available: http://www.cisco.com/c/en/us/td/docs/solutions/Enterprise/Data_Center/HA_Clusters/HA_Clusters.pdf

[5] R. E. Brown, R. Brown, E. Masanet, B. Nordman, B. Tschudi, A. Shehabi, J. Stanley, J. Koomey, D. Sartor, P. Chan, and J. Loper, *Report to Congress on Server and Data Center Energy efficiency: Public law 109-431* (No. LBNL-363E). Ernest Orlando Lawrence Berkeley National Laboratory, Berkeley, CA (US), 2007.

[6] C. Guo, et al., "BCube: A high performance, server-centric network architecture for modular data centers," in *Proc. ACM SIGCOMM Conf. Data Commun.*, Aug. 2009, pp. 63–74.

[7] L. Huang, Q. Jia, X. Wang, and B. Li, "PCube: Improving power efficiency in data center networks," in *Proc. IEEE Int. Conf. Cloud Comput.*, Jul. 2011, pp. 65–72.

[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. ACM SIGCOMM Conf. Data Commun.*, Aug. 2008, pp. 75–86.

[9] 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.

[10] Z. Guo and Y. Yang, "Exploring server redundancy in nonblocking multicast data center networks," *IEEE Trans. Comput.*, vol. 64, no. 7, pp. 1912–1926, Jul. 2015.

[11] Z. Guo and Y. Yang, "On nonblocking multirate multicast fat-tree data center networks with server redundancy," in *Proc. IEEE 26th Int. Parallel Distrib. Process. Symp.*, 2012, pp. 1034–1044.

[12] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," in *Proc. ACM SIGCOMM Conf. Data Commun.*, Aug. 2008, pp. 63–74.

[13] A. Singh, et al., "Jupiter rising: A decade of clos topologies and centralized control in Google's datacenter network," in *Proc. ACM Conf. Special Interest Group Data Commun.*, Aug. 2015, pp. 183–197.

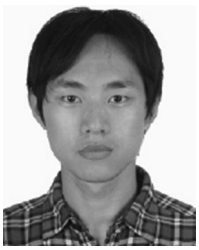
[14] A. Roy, H. Zeng, J. Bagga, G. Porter, and A. C. Snoeren, "Inside the social network's (datacenter) network," in *Proc. ACM Conf. Special Interest Group Data Commun.*, Aug. 2015, pp. 123–137.

[15] 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, Oct. 2016.

[16] D. Guo, T. Chen, D. Li, M. Li, Y. Liu, and G. Chen, "Expandable and cost-effective network structures for data centers using dual-port servers," *IEEE Trans. Comput.*, vol. 62, no. 7, pp. 1303–1317, Jul. 2013

[17] R. Mysore, et al., "PortLand: A scalable fault-tolerant layer 2 data center network fabric," in *Proc. ACM SIGCOMM Conf. Data Commun.*, Aug. 2009, pp. 39–50.

[18] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat, "Hedera: Dynamic flow scheduling for data center networks," in *Proc. ACM USENIX Conf. Netw. Syst. Des. Implementation*, 2010, pp. 19–19.
[19] B. Heller, et al., "ElasticTree: Saving energy in data center networks," in *Proc. ACM USENIX Conf. Netw. Syst. Des. Implementation*, 2010, pp. 17–17.
[20] X. Wang, Y. Yao, X. Wang, K. Lu, and Q. Cao, "CARPO: Correlation-aware power optimization in data center networks," in *Proc. IEEE INFOCOM*, 2012, pp. 1125–1133.
[21] E. Marcus and H. Stern, *Blueprints for High Availability*, 2nd ed. Hoboken, NJ, USA: Willey, 2003.
[22] L. Popa, S. Ratnasamy, G. Iannaccone, A. Krishnamurthy, and I. Stocia, "A cost comparison of data center network architectures," in *Proc. ACM 6th Int. Conf.*, Dec. 2010, Art. no. 16.
[23] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large cluster," *Commun. ACM*, vol. 51, no. 1, pp. 789–795, Jan. 2008.
[24] T. Benson, A. Akella, and D. Maltz, "Network traffic characteristics of data centers in the wild," in *Proc. ACM SIGCOMM Conf. Internet Meas.*, Nov. 2010, pp. 267–280.
[25] K. Zheng, X. Wang, L. Li, and X. Wang, "Joint power optimization of data center network and servers with correlation analysis," in *Proc. IEEE INFOCOM*, 2014, pp. 2598–2606.
[26] Z. Guo, S. Hui, Y. Xu, and H. J. Chao, "Dynamic flow scheduling for power-efficient data center networks," in *Proc. IEEE/ACM 24th Int. Symp. Quality Service*, 2016, pp. 1–10.
[27] Y. Shang, D. Li, and M. Xu, "A comparison study of energy proportionality of data center network architectures," in *Proc. IEEE Distrib. Comput. Syst. Workshops*, 2012, pp. 1–7.



Zhenhua Li received the BEng. and MS degrees in information and communication engineering from Zhejiang University, Hangzhou, China. He is currently working toward the PhD degree in the Electrical and Computer Engineering Department, Stony Brook University, Stony Brook, New York. His research interests include data center network structures, SDN, multicast, cloud computing, and network virtualization.

Yuanyuan Yang received the BEng and MS degrees in computer science and engineering from Tsinghua University, Beijing, China, and the MSE and PhD degrees in computer science from Johns Hopkins University, Baltimore, Maryland. She is a SUNY distinguished professor of computer engineering and computer science and the associate dean for the Academic Affairs in the College of Engineering and Applied Sciences, Stony Brook University, New York. Her research interests include data center networks, cloud computing, and wireless networks. She has published more than 380 papers in major journals and refereed conference proceedings and holds seven US patents in these areas. She is currently an associate editor-in-chief for the *IEEE Transactions on Cloud Computing* and an associate editor for the *ACM Computing Surveys*. She has served as an associate editor-in-chief and an associated editor for the *IEEE Transactions on Computers* and an associate editor for the *IEEE Transactions on Parallel and Distributed Systems*. She has also served as a general chair, program chair, or vice chair for several major conferences and a program committee member for numerous conferences. She is a fellow of the IEEE.