RRect: A Novel Server-Centric Data Center Network with High Power Efficiency and Availability

Zhenhua Li and Yuanyuan Yang, *Fellow, IEEE*

Abstract—In this paper, we propose a novel server-centric network for data centers, called RRect. Compared to existing server-centric networks, RRect provides a more graceful degradation performance in the presence of component failure. Meanwhile, RRect enjoys a linear diameter to the network order and multiple parallel paths. We present algorithms to find paths and to construct all parallel paths between any pair of servers in RRect. We also show that without using any power-aware routing algorithm, RRect behaves more power efficient in its interconnection network. To meet today's stringent high availability requirement, RRect can be configured into redundancy and failover scheme. In particular, RRect can be configured into both symmetric and asymmetric redundancy modes to cater different applications' needs, which cannot be implemented in existing server-centric networks. Moreover, RRect gives more flexibility to adjust the network size. Given the same switches, by fine tuning the parameters, RRect provides numerous structures with different sizes, while preserving the properties of short diameter and multiple parallel paths. Our comprehensive simulations show that RRect gives much better graceful degradation performance in the presence of component failure, and RRect saves much energy in the interconnection network. Meanwhile, RRect can maintain the same performance on many critical metrics as BCube, including short diameter and excellent aggregate throughput. All these features make RRect a very empirical structure for enterprise dater center network products.

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

1 Introduction

Cloud computing has drawn significant attentions recently as it provides an innovative way to organize computing resources which brings tremendous benefits to both industry and academia. Data center, as the cornerstone of cloud computing, has a critical impact on the performance and further on the economical ecosystem of cloud computing. Thus, due to the importance and driven by technology, considerable network structures for data center have been proposed recently, which can be mainly divided into two categories: switch-centric networks and server-centric networks. In a switch-centric network, servers only send packets to and receive packets from the interconnection network, while switches are responsible for the routing task. The classical FatTree [10], VL2 [11] and Portland [14] 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 transferring packets for each other, and switches act simply as crossbars. Thus the routing task of a server-centric network is assigned to servers. DCell [8], BCube [6] and BCCC [9] belong to this category. A significant advantage of server-centric networks is that the network hardware cost can be reduced drastically, as inexpensive commodity switches, such as layer-2 switches, are sufficient given that routing tasks have been shifted to servers where computing resources are abundant. In addition, switch-centric networks consist of more interconnection switches than server-centric networks, thus given a similar network size, switch-centric networks consume significantly more power than server-centric networks. Moreover, since servers are much more programmable than switches, server-centric network structures can accelerate the process of network innovation.

Compared to the one-time construction cost of a DCN, energy consumption acts more as the main contribution to the cost in the long term operation of the DCN. It was reported that the nationwide DCNs consume 100 billion kWh energy at a cost of $7.4 billion in 2011 [5]. Therefore, energy consumption plays a critical role in designing a DCN and choosing the location of the target DCN. In general, power consumption in a switch is usually modeled as a variate part, which is linear to the number of ports, plus a fixed part for fabrics overhead. An observation from current device market is that although switches with different numbers of ports have different power consumption overhead, the overhead is relatively fixed, provided that the numbers of switch ports are in the same order of magnitude. Thus three 32-port switches consume drastically more power than two 48-port switches. Therefore, to get a better power efficiency, it is desired to use fewer switches with as many ports as possible.

• *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 27 May 2017; revised 30 Jan. 2018; accepted 17 Feb. 2018. Date of publication 16 Mar. 2018; date of current version 8 Sept. 2020.
(Corresponding author: Zhenhua Li.)
Recommended for acceptance by L. Navarro.
Digital Object Identifier no. 10.1109/TCC.2018.2816650

Also, many data centers today have extremely stringent availability requirements, as the downtime of services running on them incurs a huge capital cost and reputation loss for both service providers and service costumers. 6-nine(99.9999 percent) or almost error free is a common availability target for DCNs. However, as the network size continues to increase, the component failure becomes more and more common. Thus, how to achieve this almost error free target on a network platform with frequent component failures is a challenge. To alleviate this problem, redundancy scheme is implemented into DCNs [4]. When normal servers fail, either hardware malfunction or software crash, the backup servers will work on the tasks left by the failed servers. Usually, since servers only send or receive packets and each server works independently of others in terms of traffic, switch-centric networks can easily adopt redundancy scheme. However, server-centric network structures are not suitable to deploy redundancy as every server takes the responsibility of forwarding packets for each other. Moreover, if the structure is set to active-passive mode, putting some servers to standby mode may cut off paths for some pairs of servers, which further impacts the throughput of the entire system. As introduced earlier, server-centric networks have great potential to be a promising candidate. However, the availability issue impedes existing server-centric networks from being used in a reliable DCN product.

In this paper, we propose a novel server-centric network structure, called RRect, which is a recursively built structure and has many good properties. RRect has a diameter that increases linearly to the network order, and has multiple near-equal parallel paths between any pair of servers within it. We also show that BCube is a special case of RRect. On the other hand, given the same network size and order, RRect consumes significantly less power than BCube. In addition, RRect can be configured into redundancy and failover scheme where the system can be configured to either symmetric redundancy mode or asymmetric mode, and making servers standby will not affect the performance of remaining servers.

The main contributions of this paper are as follows. First, we propose the recursive way to construct RRect, and the related addressing scheme to facilitate routing. We also provide an efficient routing algorithm for RRect. Second, we theoretically analyse the power efficiency of RRect, and deploy PCube framework into RRect. Third, we show how RRect is highly service available. Finally, we conduct comprehensive simulations which reveal convincing evidence that RRect gives a more graceful performance degradation and significantly less power consumption in its interconnection network than BCube, while preserving similar performance to BCube in many other critical metrics, such as average path length, path distribution, aggregate throughput, and so forth.

The rest of this paper is organized as follows. Section 2 discusses the related work concerning server-centric networks. Section 3 describes the recursive construction procedure of RRect, and presents routing algorithms and related properties of RRect. Section 4 analyses the power consumption of RRect and the flexibility of network size RRect provides. Section 5 shows how to deploy active/active and active/passive redundancy models into RRect. Section 6 gives comprehensive performance evaluations for RRect. Finally, Section 7 concludes this paper.

2 Related Work

Among existing server-centric network structures, BCube is a typical structure that provides many good properties, such as short network diameter, large network size, multiple parallel paths between any pair of servers, graceful degradation, and so forth. However, the power consumption caused by the interconnection network of BCube still occupies a relatively large proportion of the total power consumption in a data center built on BCube, which should be kept as low as possible. Details will be discussed in later sections. On the contrary, DCell consumes significantly less power than BCube by allowing servers directly connecting to each other without via any switch. Comparisons in [12] show that DCell has a drastically smaller bisection bandwidth, a key factor to throughput performance, than BCube. Besides, DCell behaves more sensitively to component failure, such as server or link failure. To reduce power consumption, PCube [7] was proposed. PCube acts more as a scheme, which dynamically turns off the unused switches in BCube based on a series of rules, than a new structure. However, the bisection bandwidth of PCube is still small. Besides, turning switches on and off efficiently requires traffic load estimation, either in hardware or software. Given that in a server-centric network, most tasks are fulfilled in servers, workload in servers is heavy and layer-2 switches are used for the purpose of low cost. Thus, both software estimator and hardware one will generate a huge capital expenditure for a server-centric network.

The proposed RRect differs from structures discussed above in that RRect saves power without sacrificing the good properties enjoyed by existing server-centric network structures. In addition, RRect provides the capability of deploying redundancy scheme, which is not achievable for existing structures. Next, we discuss how to construct RRect.

3 RRect Network Structure

To make this paper self-contained, in this section we first introduce the structure of BCube. Then we describe the structure of RRect and provides the recursive way to construct an RRect. We also discuss several critical topological properties of RRect and propose two algorithms for routing and constructing parallel paths respectively.

3.1 Preliminaries

BCube is a leveled and recursively constructed network structure. First, $BCube(n,0)$ is simply composed of n servers connected by an n-port switch. Then $BCube(n,k)$, where $k>0$, is constructed by n $BCube(n,k-1)$s along with n^k n-port switches. Thus there are n^{k+1} servers and $(k+1)n^k$ switches which are evenly divided into $k+1$ levels. Each level has n^k switches and the newly added n^k switches are in the top level. To construct $BCube(n,k)$, each server in these n $BCube(n,k-1)$s is connected to a corresponding switch in the top level. BCube structure is closely related to

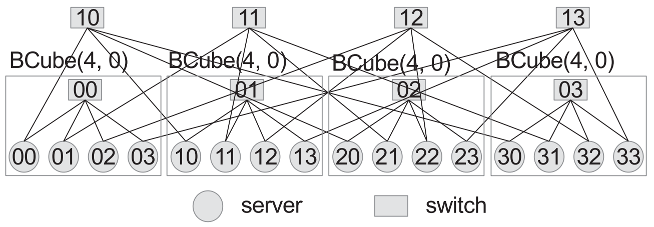


Fig. 1. Structure of $BCube(4,1)$, which is composed of four $BCube(4,0)$s and four switches.

a k-ary n-cube. Fig. 1 presents an example of $BCube(4,1)$ which consists of four $BCube(4,0)$s and four 4-port switches. BCube has many enjoyable features for DCN, such as a large accommodation of servers while keeping a very short network diameter, a wide bisection bandwidth, and an excellent aggregate throughput, which make BCube a very promising structure for DCN products.

3.2 Structure of RRect

RRect is also a recursively constructed network structure, for which off-the-shelf commodity servers and switches are sufficient. An $RRect(n,m,0)$ is simply constructed by mn servers connected by an mn-port switch. Then when $k \geq 1$, an $RRect(n,m,k)$ is built by n $RRect(n,m,k-1)$s along with n^k mn-port switches. Denote the size of $RRect(n,m,k)$, the number of servers an $RRect(n,m,k)$ accommodates, as $S(n,m,k)$. Then we have

$$\begin{cases} S(n,m,0) = mn \\ S(n,m,k) = n \cdot S(n,m,k-1). \end{cases} \tag{1}$$

Therefore, the size of $RRect(n,m,k)$ is mn^{k+1}.

Next, we describe the detailed procedure of constructing an $RRect(n,m,k)$. In the $RRect(n,m,0)$, we identify each server by a symbol a_0, where $a_0 \in \{0,1,\ldots,mn-1\}$. As aforementioned, an $RRect(n,m,k)$, $k \geq 1$, consists of n $RRect(n,m,k-1)$s. These n $RRect(n,m,k-1)$s can be identified by a_k, where $a_k \in \{0,1,\ldots,n-1\}$. Then n $RRect(n,m,k-2)$s in each $RRect(n,m,k-1)$ can be represented by a_{k-1}, where $a_{k-1} \in \{0,1,\ldots,n-1\}$. This process continues until k decreases to 0. Thus, in this way, we can identify each server in $RRect(n,m,k)$ by a $(k+1)$-tuple, $a_k a_{k-1} \ldots a_0$, where $a_i \in \{0,1,\ldots,n-1\}$, $i \in \{1,2,\ldots,k\}$ and $a_0 \in \{0,1,\ldots,mn-1\}$. We call this $(k+1)$-tuple the address of a server, which uniquely manifests the location of the server. 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_kth $RRect(n,m,k-1)$ and so on.

Besides the mn^{k+1} servers in $RRect(n,m,k)$, there are also $(k+1)n^k$ switches, of which n^k switches are newly added and the remaining kn^k switches are among the n $RRect(n, m, k-1)$s. Following similar rules to servers, we can identify a switch by a $(k+1)$-tuple too, which 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 $s_k \in \{0,1,\ldots,k\}$. Here we say that switch $s_k s_{k-1} \ldots s_0$ with $s_k = k$ is one of the n^k newly added switches and others are within n $RRect(n,m,k-1)$s.

To facilitate routing in RRect, we further let formats $a_k a_{k-1} \ldots a_{i+1}\{a_i + \lfloor \frac{a_0}{n} \rfloor n\} a_{i-1} \ldots \{a_0 - \lfloor \frac{a_0}{n} \rfloor n\}$, where $i \in \{1, 2,\ldots,k\}$, represent the same server. For example, server 14 can also be represented as 50 in $RRect(4,2,1)$. Now suppose

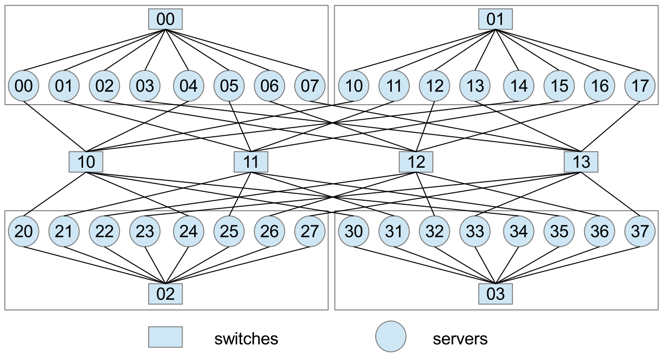


Fig. 2. Structure of $RRect(4,2,1)$, which is composed of four $RRect(4,2,0)$s and four switches.

we have n $RRect(n,m,k-1)$s and n^k switches and have assigned them their own addresses. Then the procedure of constructing an $RRect(n,m,k)$ is as follows. First, change the addresses of all servers to format $\{a_k + \lfloor \frac{a_0}{n} \rfloor n\} a_{k-1} \ldots a_1 \{a_0 - \lfloor \frac{a_0}{n} \rfloor n\}$. Then connect each server to one of the n^k switches, $s_k s_{k-1} \ldots s_0$, where $s_k = k$, according to the following equation:

$$a_{k-1}a_{k-2}\ldots\left\{a_0 - \left\lfloor \frac{a_0}{n} \right\rfloor n\right\} = s_{k-1}s_{k-2}\ldots s_0. \tag{2}$$

The process recursively continues until k reduces to 0 and this concludes the construction of an $RRect(n,m,k)$. An example of $RRect(4,2,1)$ is shown in Fig. 2. To get a better understanding of the structure of $RRect(n,m,k)$, we also provide a geometrical view of $RRect(4,2,1)$, which is presented in Fig. 3. Clearly, it can be observed that RRect is a symmetric topology, where using any node as the original point will give the same structure.

From the structure described above, it can also be easily demonstrated that $BCube(n,k)$ is a special case of an $RRect(n,m,k)$ when $m=1$. In addition, from Fig. 3, it can be observed that in $RRect(4,2,1)$, servers $\{00, 01, 02, 03, 10, 11, 12, 13, 20, 21, 22, 23, 30, 31, 32, 33\}$ form a $BCube(4,1)$, while servers $\{04, 05, 06, 07, 14, 15, 16, 17, 24, 25, 26, 27, 34, 35, 36, 37\}$ form the other $BCube(4,1)$. This observation illustrates that $RRect(n,m,k)$ can be considered as m $BCube$ (n,k)s, one overlapping another. As a result, $RRect(n,m,k)$ can also be considered as constructed from m $BCube(n,k)$s by binding the corresponding servers together, for instance, servers 00 and 04 in $RRect(4,2,1)$. However, by simply binding servers

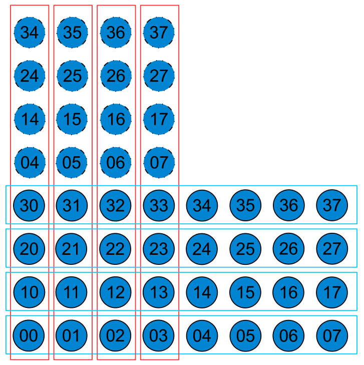


Fig. 3. Geometrical view of $RRect(4,2,1)$, where servers in a horizontal rectangle are within the same $RRect(4,2,0)$ and servers in a vertical rectangle are connected by the same newly added switch.

to form the new structure, RRect is entitled much better properties, regarding power efficiency and service availability, which we will discuss in Sections 4 and 5, respectively.

Next, we discuss routing algorithms and related properties of RRect.

3.3 Routing in RRect

Note that switches in server-centric networks act simply as crossbars, thus we say that a server has only one-hop distance to its neighbors, the servers connected by the same switch, which is also the assumption in other works about server-centric networks. Denote the Hamming distance between two servers $A = a_k a_{k-1} \dots a_0$ and $A' = a'_k a'_{k-1} \dots a'_0$ as $h(A, A')$, which is the number of different digits in the addresses between A and A'. The routing algorithm of RRect is actually to find a way to correct every digit of the addresses from the source server to the destination server. In other words, it reduces the Hamming distance to 0 between the source server and the destination server. This algorithm is presented as RRectRouting in Algorithm 1, which operates as follows. First, RRectRouting checks the format of the source and destination to find the one that gives the shortest Hamming distance. For example, server 14 in $RRect(4, 2, 1)$ has two address formats 14 and 50. The Hamming distance between servers 00 and 14 is 2. However, if we represent server 14 in the format of 50, the Hamming distance to server 00 can be shortened by 1 hop. Therefore, RRectRouting checks the formats first to ensure the path generated by it is the shortest one. Meanwhile, this also helps balance the workload among all the switches. Next, RRectRouting corrects one digit difference between the designated formats of source and destination servers chosen in the first step. Which digit will be corrected first is determined by the permutation of array $[0, 1, \dots, k]$. Clearly, the maximum Hamming distance between any pair of servers in $RRect(n, m, k)$ is $k + 1$. Therefore, we have the following theorem.

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

Proof. RRectRouting corrects one digit difference between source server and destination server in each iteration by one hop. Since there are $k + 1$ digits, after at most $k + 1$ hops, RRectRouting can find a path between any pair of servers. Next, we show that it is also necessary to spend $k + 1$ hops to find a path from a server to anther one that is farthest to the source. This can be easily shown by an example where all the digits in the address of the source server are 0's and all the digits in the address of the destination server are 1's. In this case, $k + 1$ hops are necessary. Thus, the diameter of $RRect(n, m, k)$ is $k + 1$. □

Theorem 1 indicates that the diameter of RRect increases only linearly to its network order k. Compared to the size of RRect, which increases exponentially to k, servers in RRect enjoy excellent communication latency. This feature favors the cloud computing, as one of the requirements in cloud computing is that a task can be assigned to any part of the data center network. Moreover, different permutations in RRectRouting generate different paths. For example, in $RRect(4, 2, 1)$ in Fig. 2, whose diameter is 2, server 37 can be reached from server 00 by 2 hops. By using different permutations, this path can be either {00, 07, 37} or {00, 30, 37}. Additionally, by changing the format of server 37 to 73, we can also get two parallel paths, which are {00, 03, 37} and {00, 34, 37}. This observation inspires following theorems.

Algorithm 1. RRect Routing Algorithm to Find a Path from a Single Source to a Single Destination

```
Require:
 1: source A, destination A′,
 2: and Π which is the permutation of $[k, k-1, \dots, 0]$
Ensure:
 3: list of intermediate servers of the path from A to A′, path(A, A′)
 4: function RRectRouting(A, A′, Π)
 5:   path(A, A′) = {A};
 6:   src_offset = (A[0] $/n) * n$; //change source address format
 7:   dst_offset = (A′[0] $/n) * n$; //change source address format
 8:   A[0] -= (A[0] $/n) * n$;
 9:   A′[0] -= (A′[0] $/n) * n$;
10:   if A[0] == A′[0] then
11:     //path could be shortened by one hop
12:     randomly choose a position idx such that A[idx] $\neq$ A′[idx]
13:   end if
14:   A[idx] += src_offset;
15:   A′[idx] += dst_offset;
16:   tmp_node = A;
17:
18:   for $i = k \to 0$ do
19:     tmp_node[$\pi_i$] = A′[$\pi_i$];
20:     inter_node = tmp_node;
21:     if tmp_node[idx] $\geq n$ then
22:       // change address format back
23:       inter_node[0] += (tmp_node[idx] $/n) * n$;
24:       inter_node[idx] -= (tmp_node[idx] $/n) * n$;
25:     end if
26:     append inter_node to path(A, A′);
27:   end for
28:   return path;
29: end function
```

Theorem 2. *Given that two servers $A = a_k a_{k-1} \dots a_0$ and $A' = a'_k a'_{k-1} \dots a'_0$, in the formats of $a_k \dots a_{i+1}\{a_i + \lfloor \frac{a_0}{n} \rfloor n\}$ $a_{i-1} \dots \{a_0 - \lfloor \frac{a_0}{n} \rfloor n\}$ and $a'_k \dots a'_{i+1}\{a'_i + \lfloor \frac{a'_0}{n} \rfloor n\} a'_{i-1} \dots \{a'_0 - \lfloor \frac{a'_0}{n} \rfloor n\}$ respectively are different in every digit of their address formats, 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 $i_0, i_1 \in \{0, 1, \dots, k\}$, RRectRouting generates two parallel paths between A and A'.*

Proof. We denote the two paths generated by the two different permutations Π_0 and Π_1 as P_0 and P_1 respectively. Since P_0 and P_1 consist of multiple intermediate nodes, they can be represented 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 $N_i^{(j)}$, $i \in \{1, 2, \dots, k\}$ and $j \in \{0, 1\}$, represents the ith intermediate node in P_j. We show that P_0 and P_1 are parallel to each other in two aspects, intermediate servers and switches.

First, we show that $N_i^{(0)}$, $i \in \{1, 2, \dots, k\}$, cannot appear in P_1. In other words, $N_i^{(0)} \neq N_j^{(1)}$, where $i, j \in \{1, 2, \dots, k\}$. When $i \neq j$, source server A routes to

$N_i^{(0)}$ by correcting i digit differences and it routes to $N_j^{(1)}$ by correcting j digit differences. Since $i \neq j$, there must be at least one digit difference between addresses of $N_i^{(0)}$ and $N_j^{(1)}$. Hence $N_i^{(0)} \neq N_j^{(1)}$. In other words, $N_i^{(0)}$ cannot appear in P_1 at a position other than i. Moreover, as P_0 and P_1 are generated from Π_0 and Π_1 respectively by correcting different digits of the address of server A, the digits corrected in $N_i^{(0)}$ are at positions of $a_{i_0}, a_{i_0-1}, \ldots, a_{(i_0-i) \bmod (k+1)}$, while in $N_i^{(1)}$, the digits corrected are at positions of $a_{i_1}, a_{i_1-1}, \ldots, a_{(i_1-i) \bmod (k+1)}$. Since $i_0 \neq i_1$, there is also at least one digit in $N_i^{(0)}$ different from $N_i^{(1)}$. Thus, $N_i^{(0)} \neq N_i^{(1)}$. Therefore, $N_i^{(0)}$ cannot appear in P_1. Similarly, $N_j^{(1)}$ cannot appear in P_0 either.

Next, we show that switches in P_0 and P_1 are also different. First, we show that each switch in a single path, either P_0 or P_1, cannot appear in that path more than once. Based on the construction rule of RRect, all the servers connected by the same switch have only one digit difference at a specified position. In a single path, if a switch appears more than once, there must be at least three intermediate servers connected to it. However, these servers have addresses different at different positions, which contradicts the construction rule. Hence, switches cannot appear in a single path more than once. Next, we show that switches in P_0 cannot appear in P_1. Denote $S_i^{(j)}$ as the switch at the ith position in path P_j. If a switch $S_i^{(0)}$ appears in both P_0 and P_1, since servers in P_0 and P_1 are different, there are four different servers connected by $S_i^{(0)}$, which can be separately represented as $N_{i-1}^{(0)}$, $N_i^{(0)}$, $N_{i'-1}^{(1)}$ and $N_{i'}^{(1)}$ respectively. However, based on RRectRouting and the construction rule of RRect, we will have $N_i^{(0)} = N_{i'}^{(1)}$, which contradicts that intermediate servers cannot appear in both P_0 and P_1. Hence switches in one path cannot appear in the other path. Therefore, paths P_0 and P_1 are parallel to each other. □

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

Theorem 3 can be shown by constructing such $k+1$ paths, which can be built by the algorithm in Algorithm 2. The paths between a pair of servers A and A' constructed by RRectParaPaths are divided into two groups generated by DirectRouting and IndirRouting respectively. $h(A, A')$ paths belong to the group generated by DirectRouting and $k+1-h(A, A')$ paths belong to the other group. Clearly, the $h(A, A')$ paths from DirectRouting are parallel to each other by Theorem 2.

Next, we show that the $k+1-h(A, A')$ paths from IndirRouting are also parallel to each other. Suppose after transforming the forms of addresses of servers A and A' to the forms where $h(A, A')$ is minimized, the ith digits between addresses of A and A' are the same. Then this path will be created by IndirRouting which will add a neighbor B, $b_k b_{k-1} \ldots b_0$, of source server A into the path, where $b_i \neq a_i$. Next, IndirRouting chooses such a permutation that b_i will be corrected to a'_i in the last iteration. In other words, all the intermediate nodes in this path have addresses where the ith digits are different from that of both source and destination servers. Similarly, another path generated based on that $a_j = a'_j$ and $j \neq i$, will have all its intermediate nodes, $b_k b_{k-1} \ldots b_0$, with $b_j \neq a_j$ and $b_j \neq a'_j$. Since $i \neq j$, there are at least two digit differences between an intermediate node in the first path and an intermediate node in the second path. Therefore, the intermediate nodes in the first path cannot appear in the second path and vice versa. In addition, since there are at least two digit differences between nodes in the first path and the second path, based on the construction rule discussed earlier, they cannot be connected by the same switches. Thus switches in one path cannot appear in the other path either. Therefore, two paths generated by IndirRouting are parallel to each other.

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

```
Require:
 1: source $A = a_k a_{k-1} \ldots a_0$ and destination $A' = a'_k a'_{k-1} \ldots a'_0$
Ensure:
 2: $k+1$ parallel paths between $A$ and $A'$
 3: function RRECTPARAPATHS(A, A′)
 4:     pathSet = { };
 5:     src_offset = (A[0] $/n$) $* n$; //change source address format
 6:     dst_offset = (A′[0] $/n$) $* n$; //change source address format
 7:     A[0] -= (A[0] $/n$) $* n$;
 8:     A′[0] -= (A′[0] $/n$) $* n$;
 9:     if A[0] == A′[0] then
10:         // path could be shortened by one hop
11:         randomly choose a position idx such that A[idx] $\neq$ A′[idx]
12:     end if
13:     A[idx] += src_offset;
14:     A′[idx] += dst_offset;
15:     tmp_node = A;
16:     for $i = 0 \rightarrow k$ do
17:         if A[$i$] $\neq$ A′[$i$] then
18:             $P_i$ = DirectRouting(A, A′, $i$, idx);
19:         else
20:             $P_i$ = IndirRouting(A, A′, $i$, idx);
21:         end if
22:         add $P_i$ to pathSet;
23:     end for
24:     return pathSet;
25: end function
```

Finally, we show that paths generated by IndirRouting are also parallel to paths built by DirectRouting, which can be shown by a similar proof discussed above. Therefore, there are $k+1$ parallel paths between any pair of servers in an $RRect(n, m, k)$.

For example, in an $RRect(4, 2, 2)$, there should be three parallel paths. The Hamming distance between servers 000 and 025 is 2. Thus there are two parallel paths generated by DirectRouting which are {000, 020, 025} and {000, 005, 025}. In addition, there is another path generated by IndirRouting which is {000, 100, 120, 125, 025}. Clearly, it can be observed that all the three paths have nearly equal distances and are parallel to each other. The abundance of near-equal parallel paths guarantees RRect with an excellent graceful degradation in the presence of component failures, such as server crashes and switch malfunctions.

Next, we discuss the energy efficiency and flexibility of network size of RRect.

Algorithm 3. Functions IndirRouting and DirectRouting

```
function IndirRouting(A, A′, source, $i$, idx)
    path = {A};
    tmp_node = A;
    // randomly choose a value other than $a_i$ or $a'_i$
    tmp_node[$i$] = random() $\bmod n$;
    inter_node = tmp_node;
    if tmp_node[idx] $\geq n$ then
        // change address format back
        inter_node[0] += (tmp_node[idx] $/n) * n$;
        inter_node[idx] -= (tmp_node[idx] $/n) * n$;
    end if
    append inter_node to path(A, A′);
    $\ell = k$;
    for $j = i - 1 \rightarrow i - k - 1$ do
        $\pi_\ell = j \bmod (k+1)$; //build up permutation
        $\ell$- -;
    end for
    for $j = k \rightarrow 0$ do
        tmp_node[$\pi_j$] = A′[$\pi_i$];
        inter_node = tmp_node;
        if tmp_node[idx] $\geq n$ then
            // change address format back
            inter_node[0] += (tmp_node[idx] $/n) * n$;
            inter_node[idx] -= (tmp_node[idx] $/n) * n$;
        end if
        append inter_node to path(A, A′);
    end for
    return path;
end function
function DirectRouting(A, A′, $i$, idx)
    path = {A};
    tmp_node = A;
    $\ell = k$;
    for $j = i \rightarrow i - k$ do
        $\pi_\ell = j \bmod (k+1)$;
        $\ell$- -;
    end for
    for $j = k \rightarrow 0$ do
        tmp_node[$\pi_j$] = A′[$\pi_i$];
        inter_node = tmp_node;
        if tmp_node[idx] $\geq n$ then
            // change address format back
            inter_node[0] += (tmp_node[idx] $/n) * n$;
            inter_node[idx] -= (tmp_node[idx] $/n) * n$;
        end if
        append inter_node to path(A, A′);
    end for
    return path;
end function
```

4 Energy Efficiency and Size Flexibility

It is stated that the interconnection network of a DCN occupies around 15-20 percent of the total power consumption, within which the fabric overhead of switches, including chip set and cooling system, contributes most to the energy consumption. On the contrary, turning off an unused switch port only saves 1-2Watts [16]. In addition, we have studied the power consumption of switches provided by some well-known vendors [1], [2], [3] and summarize some representative results in Table 1, 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, three 32-port switches consume much more power than two 48-port switches. Therefore, it is desirable that a DCN with a targeted size should be constructed using as few switches as possible.

TABLE 1
Power Consumption of Switches

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

BCube uses numerous switches, that cost considerable amount of energy. Now we show that given RRect and BCube with the same network size and diameter, without using any power-aware routing algorithm, RRect consumes less power in the interconnection network than BCube. Denote the power of a switch at time t as $P(t)$ which is composed of a fixed overhead power p_0 and power consumed by each port p_1. Thus

$$P(t) = p_0 + i(t)p_1, \tag{3}$$

where $i(t)$ represents the number of active ports in a switch at time t. Suppose we have $BCube(n_1, k)$ and $RRect(n_2, m, k)$ such that $n_1^{k+1} = mn_2^{k+1}$. Thus they consume $(k+1)n_1^k$ and $(k+1)n_2^k$ switches respectively. Then the ratio of interconnection network power consumption of $RRect(n_2, m, k)$ to that of $BCube(n_1, k)$ can be represented as follows:

$$\begin{aligned}\frac{P_{RRect}}{P_{BCube}} &= \frac{(k+1)n_2^k \cdot p_0 + \alpha_2(k+1)mn_2^{k+1} \cdot p_1}{(k+1)n_1^k \cdot p'_0 + \alpha_1(k+1)n_1^{k+1} \cdot p_1} \\ &= \frac{(\frac{1}{m})^{\frac{k}{k+1}} p_0 + \alpha_2 p_1 n_1}{p'_0 + \alpha_1 p_1 n_1},\end{aligned} \tag{4}$$

where α_1 and α_2 denote the proportion of ports that are active in BCube and RRect structures respectively, and p_0 and p'_0 are the overhead power of switches in each structure respectively. As discussed above, $p_0 \approx p'_0$, given that n_1 and mn_2 are in the same order of magnitude. Meanwhile, due to the structure similarity between BCube and RRect, provided a similar traffic load, the numbers of active ports in $RRect(n_2, m, k)$ and $BCube(n_1, k)$ should be similar, if not exactly the same. Therefore, $\alpha_1 \approx \alpha_2$. Denote them as α. Then we have

$$\lim_{k \to +\infty} \frac{P_{RRect}}{P_{BCube}} = \frac{p_0 \frac{1}{m} + \alpha p_1 n_1}{p_0 + \alpha p_1 n_1}. \tag{5}$$

Clearly, it can be observed that, without using any power-aware routing algorithm, naturally RRect consumes significantly less energy than BCube. As reported in [16], usually $\frac{p_1 n_1}{p_0 + p_1 n_1} \leq 25\%$. Hence, when $m = 2$, $RRect(n_2, 2, k)$ saves

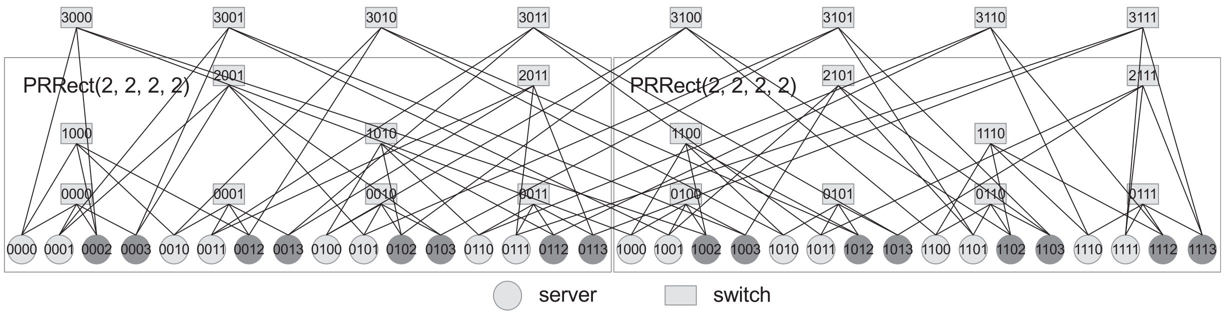


Fig. 4. Structure of $PRRect(2,2,3,3)$, consisting of two $PRRect(2,2,2,2)s$ and eight switches.

37.5-50 percent energy compared to $BCube(n_1, k)$ depending on the traffic load.

Note that as explained earlier, given a similar traffic load, the numbers of switch ports used to support the traffic in each structure should be similar, if not exactly the same. Similarly, the numbers of Network Interface Card (NIC) ports in servers active for relaying traffic in these two structures should also be similar. Hence, the difference of power consumed by servers for the purpose of transiting traffic load in these two structures should be trivial and negligible. As a result, we focus on the power consumption in the interconnect switches.

Moreover, as introduced in Section 2, PCube can be deployed on BCube to reduce power consumed by interconnect switches in way of systematically turning off several switches based on dedicatedly designed rules. In this section, we show that the scheme of PCube can also be applied to RRect if needed, denoted as PRRect, and hence the power consumption from the interconnect network can be further saved.

Compared to RRect, PRRect introduces one more parameter, q, to represent the number of ports used in each server. The structure of PRRect can be interpreted in two perspectives. On one hand, PRRect is also a recursively constructed structure. The base structure is denoted as $PRRect(n, m, k, 2)$, which has a similar structure to $RRect(n, m, k)$. To be specific, $PRRect(n, m, k, 2)$ and $RRect(n, m, k)$ have exactly the same number of servers. For switches, following the addressing scheme discussed in Section 3, the switches kept in $PRRect(n, m, k, 2)$ are those with addresses regarding the following rule:

$$\begin{cases} s_k s_{k-1} \dots s_0, \text{ if } s_k = 0 \\ s_k s_{k-1} \dots s_{s_k} e_{s_k - 1} o_{s_k - 2} \dots o_0, \text{ if } s_k \in \{1, \dots, k-1\} \\ s_k s_{k-1} o_{k-2} \dots o_0, \text{ if } s_k = k \end{cases}. \tag{6}$$

In the expression above, o_i denotes an odd integer that is within the set $\{0, 1, \dots, n-1\}$. Similarly, e_i represents an even number within the same set. Thus, there are mn^{k+1} servers and $2n^k$ switches in a $PRRect(n, m, k, 2)$. An instance of $PRRect(2, 2, 2, 2)$ is depicted in the left rectangle in Fig. 4. Within this $PRRect(2, 2, 2, 2)$, all the four switches in the first level, $s_k = 0$, are reserved. In the second level, only switches with the last digit in the address being even numbers are kept, namely switches 1000 and 1010. On the other hand, in the third level, also the top level, switches with the last digit being odd numbers are left, which are switches 2001 and 2011.

Given the definition of the base structure $PRRect(n, m, k, 2)$, a $PRRect(n, m, k, q)$ is constructed with n $PRRect(n, m, k-1, q-1)$s and n^k switches. The connections between those servers and switches follow the same principle as constructing an $RRect(n, m, k)$. This construction process recursively continues until $q = 2$. Fig. 4 also shows an example of $PRRect(2, 2, 3, 3)$ that is composed of two $PRRect(2, 2, 2, 2)$s and eight switches.

On the other hand, it can be easily observed that $PRRect(n, m, k, q)$ is a sub-graph of $RRect(n, m, k)$. This observation inspires the second perspective to understand the structure of PRRect, which is also more important than the former one. That is $PRRect(n, m, k, q)$ is derived by dynamically turning on/off switches in $RRect(n, m, k)$ with respect to the restriction in Eq. (6). Hence, the power consumption of the interconnect network in the original $RRect(n, m, k)$ can be reduced in an on-demand manner. When traffic is heavily loaded in the network, large q would be necessary. In contrast, in the case where the traffic load is light, small q in each server suffices to handle the traffic on the fly, such that a notable energy can be saved. To be more specific, the ratio of the power consumption of the interconnect network in $PRRect(n, m, k, q)$ to that in $RRect(n, m, k)$ can be represented as follows:

$$\frac{P_{PRRect}}{P_{RRect}} = \frac{qn^k \cdot p_0 + \alpha qmn^{k+1} \cdot p_1}{(k+1)n^k \cdot p_0 + \alpha(k+1)mn^{k+1} \cdot p_1}. \tag{7}$$

Different from previous analysis on power efficiency, where the same traffic pattern will result in different routes separately in BCube and RRect, in this comparison, the traffic in PRRect can have exactly the same routes as in RRect, leaving those switches not deployed in PRRect idle. Hence, the above equation can be revised as follows:

$$\begin{aligned} \frac{P_{PRRect}}{P_{RRect}} &= \frac{qn^k \cdot p_0 + \alpha qmn^{k+1} \cdot p_1}{(k+1)n^k \cdot p_0 + \alpha qmn^{k+1} \cdot p_1} \\ &= \frac{q + \alpha q \frac{mnp_1}{p_0}}{k + 1 + \alpha q \frac{mnp_1}{p_0}}. \end{aligned} \tag{8}$$

As discussed previously, in a typical switch, $\frac{p_1 mn}{p_0 + p_1 mn} \leq 25\%$. Hence, the expression above can be further simplified as

$$\frac{P_{PRRect}}{P_{RRect}} \approx \frac{q(1 + 0.33\alpha)}{k + 1 + 0.33\alpha q}. \tag{9}$$

Clearly, $q \leq k + 1$. Thus, the power efficiency in PRRect outperforms that in RRect, given the same m, n and k. The efficiency totally depends on the traffic load. Therefore, by deploying PRRect scheme on RRect, the power efficiency of the interconnect network can be further improved in the on-demand manner.

TABLE 2
Network Sizes in Different Configurations

	$RRect(4,12,2)$	$RRect(6,8,2)$	$RRect(8,6,2)$	$RRect(12,4,2)$	$RRect(16,3,2)$	$RRect(24,2,2)$	$RRect(48,1,2)$
No. of servers	768	1,728	3,072	6,912	12,288	27,648	110,592
No. of switches	48	108	192	432	768	1,728	6,912

Note that all the analysis conducted above is based on a simplified power model in which each port works in an on/off pattern. In other words, each port either works at its full power or stays idle. Therefore, it may not reflect the traffic traversing through the ports. However, as pointed out in [16], the overhead part dominates the power consumption of the entire switch, and per port power consumption is only 1-2 Watts. As a result, the conclusion that RRect is more power efficient than BCube, regarding the interconnect network, still holds, even when a more elaborate model is adopted, as long as the overhead in each switch still makes the major contribution of the power consumption.

In a nutshell, RRect achieves better power efficiency without sacrificing any important properties enjoyed by BCube, which will be shown later. There is only one minor concern. By using the mn_2-port switches adopted in $RRect(n_2, m, k)$, $BCube(mn_2, k)$ could accommodate much more servers than $RRect(n_2, m, k)$. However, since $RRect(n_2, m, k)$ holds a sufficient number of servers in empirical scenarios, sacrificing the network size is an acceptable trade-off for better power efficiency.

Meanwhile, the size of RRect enjoys more flexibility. Compared to BCube, RRect provides more options for network sizes, which is more suitable for an empirical DCN scenario. For example, by using switches with 48 ports, $BCube(48, 1)$ accommodates 2,304 servers, while $BCube(48, 2)$ will hold 110,592 servers which increases very fast. Therefore, the target size in the middle will be quite difficult to be implemented by BCube. On the other hand, by providing one more parameter m, RRect gives much more options for network sizes, given that the diameter of the network and the number of switch ports are specified. Table 2 demonstrates examples of what sizes RRect can provide in the scenario that the number of switch ports is 48 and the diameter is 3. It is true that BCube also provides a solution for building a partial BCube [6]. However, this solution still needs improvements. For example, $BCube(8, 3)$ provides 4,096 servers. If 2,000 servers are target size, we can construct a partial $BCube(8, 3)$ by using only 4 $BCube(8, 2)$s, which gives 2,048 servers. However, to keep the bisection bandwidth and other properties, unlike servers, all the switches needed in the third level in the complete $BCube(8, 3)$ have to be kept the same as in the partial $BCube(8, 3)$. Hence, it requires 1,280 8-port switches. On the contrary, RRect spends switches in proportion to the parameter m. Thus, by fine tuning the parameters, RRect can give a similar network size to the partial BCube, but much less switches used for construction than BCube. In the scenario discussed above, $RRect(6, 2, 3)$ accommodates 2,592 servers while only using 864 12-port switches. Therefore, RRect consumes significantly less energy for the interconnection network than BCube. Thus, RRect is more energy efficient and more empirical for enterprise DCN products.

Next, we discuss the availability of RRect in the presence of server failure.

5 Failover and Availability

The traditional solution to achieving high availability is to deploy server redundancy scheme, which can be divided into two categories: active-passive or asymmetric, and active-active or symmetric. In the active-passive mode, one sever is active and doing all the tasks, while its partner is a dedicated standby and going back to work only when the active one fails. On the contrary, in an active-active mode, both servers are doing 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 asymmetric redundancy mode can be further divided into $1+1$ mode and $\alpha+\beta$ mode. In $1+1$ mode, each working server is backed up by its own standby server, while in $\alpha+\beta$ mode, every α servers are backed up by β standby servers. In the presence of a working server failure, one of the β corresponding servers will warm up. The capital expenditure of deploying active-passive mode increases tremendously, since every working server requires a backup that is in sleep in most cases. Thus the asymmetric mode is not cost efficient, and the symmetric mode is more favorable. It is natural for switch-centric networks to enjoy high availability by deploying redundancy scheme, as the routing task is implemented in switches and any malfunctioning server will not impact other working servers in most cases. Thus the backup server can easily take over all the tasks left by the crashed server, and other servers continue their tasks. However, this is not true for server-centric networks. One drawback of server-centric networks, compared to switch-centric networks, is that servers in a server-centric network have to participate in the tasks of traffic routing. If a server malfunctions, all the packets that originally travel through that server will have to be re-routed and all the tasks that were worked on the malfunctioning servers should be restarted at other servers. As a result, a crashed server inevitably has a profound impact on other working servers, especially for servers that have packets traveling through that crashed server. A failed server may even damage parallel paths of other working servers. Particularly, structures like DCell which has parallel paths between a pair of servers with significantly different lengths will give an even worse performance in the presence of component failure [12]. On the contrary, RRect is more suitable for the symmetric redundancy mode. Although a crashed server will also have impact on other working servers in RRect, this impact can be minimal.

In an $RRect(n, m, k)$, every m servers form a sub-group which share the same properties. The sub-group can be identified as follows. Two servers with addresses $a_k a_{k-1} \dots a_0$ and $a'_k a'_{k-1} \dots a'_0$ respectively, belong to the same sub-group if $a_i = a'_i$, $i \in \{1, 2, \dots, k\}$ and $(a_0 \bmod n) = (a'_0 \bmod n)$. Servers within the same sub-group can be mirrors to each other. Therefore, if any server fails, caused by either system crash or hardware malfunctioning, the corresponding server in the same sub-group can take over all the tasks left by the failed

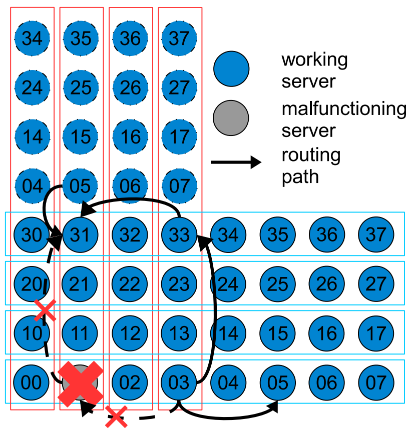


Fig. 5. Failover in $RRect(4,2,1)$ where server 01 fails and server 05 takes over all the tasks left by server 01.

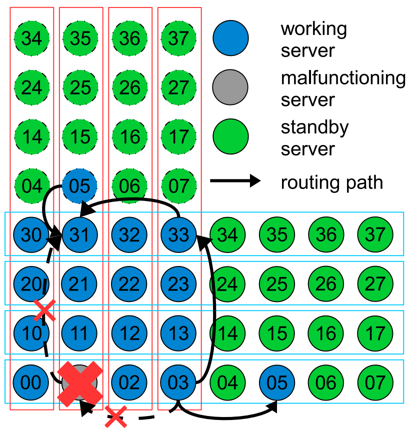


Fig. 6. Failover in $RRect(4,2,1)$ that is configured into active-passive mode and servers with addresses $a_0 \geq 4$ are set to standby mode. In this scenario, server 01 fails and server 05 warms up taking over all tasks left by server 01.

server without too much impact on other working servers. For example, in an $RRect(4,2,1)$, a pair of servers form a sub-group and they can fully take over the tasks from each other in case of server failure. Fig. 5 shows a case in $RRect(4,2,1)$ where servers 01 and 05 are mirrors to each other. Then server 01 fails and server 05 takes over all the tasks left by server 01. When server 01 was working, server 03 was able to communicate with server 31 through two parallel paths with equal distance. After server 01 crashes down, one of the two parallel paths is cut off. However, since server 05 takes over all the tasks which were assigned to server 01 at the beginning, by routing through server 05, there are still two parallel paths with equal distance between server 03 and 31, which are {03, 33, 31} and {03, 05, 31} as shown in Fig. 5. Moreover, this feature applies to not only server 03, but also all the working servers. All the packets, no matter where they come from, that are relayed by server 01, can all be relayed by server 05, while all topological properties are still reserved. To the best of our knowledge, RRect is the only server-centric network structure where every server is backed up with another server that can be interchanged for each other without impacting topology properties. Thus RRect is more suitable for today's cloud computing applications than any other existing server-centric network.

In addition, though the active-passive redundancy mode is very expensive, in some scenarios where cost is not the concern but an almost error free environment is the top priority, asymmetric redundancy mode becomes more favorable. In this case, other existing server-centric networks give even worse performance than in symmetric mode, since making some servers into standby will reduce the number of parallel paths for numerous pairs of servers and hence will impact the performance of other working servers. Moreover, in existing structures, when a working server crashes and its backup server warms up, the backup server may still have a limited routing ability, restricted by the configuration of its neighbors. Using $BCube(4,1)$ in Fig. 1 as an example, suppose we set servers 03, 13, 23 and 33 into standby. If server 00 fails and server 03 warms up, all routing jobs left by server 00 cannot be transferred to server 03. This is because the remaining servers connected by switch 13 are all set to standby, and there is only one path for server 03 to send packets out. As a result, even server 03 warms up and intends to act as the backup of server 00, it loses the relaying capability to take this responsibility, unless all the standby servers warm up together, which nullifies the benefits brought by asymmetric redundancy scheme. On the contrary, RRect can be configured into active-passive mode and the working servers keep their performance without any impact from standby servers. Also, when a working server fails and the corresponding backup server warms up, taking place of the failed server, the backup server shares the same property as the failed server. Therefore, the properties of the entire network remain the same as if there is no failure occurred. RRect can be set to this active-passive mode by setting RRect to $RRect(n,2,k)$ where servers with addresses $a_0 < n$ are configured as working servers and other servers will be set to standby mode. Fig. 6 shows an example of $RRect(4,2,1)$ configured into active-passive mode where all the servers with addresses $a_0 \geq 4$ are set to standby mode. It can be observed that all the working servers keep all their properties, including diameter, multiple parallel paths and so forth. When server 01 crashes, the corresponding standby server 05 warms up and takes over all the tasks left by server 01. We can see that by re-routing all the packets, which originally go to server 01, to server 05, all the working servers still enjoy their near-equal multiple parallel paths. In Fig. 6, due to the failure of server 01, one path between servers 03 and 31 is cut down. However, by re-routing packets to server 05, path {03, 01, 31} can be replaced by path {03, 05, 31} with equal path distance. Hence, the property of multiple near-equal parallel paths between any pair of servers can be kept as if the original working server is still working. Hence, from Fig. 6 we can see that failed servers can be fully replaced by their corresponding standby servers without any impact to other working servers. Any server that detects the failure of a working server will broadcast the failure message to other servers. Once other working servers get the message, they update their routing table and then send packets, which should go to the failed server, to the updated newly warmed up server. This part can be easily implemented, thus we omit the protocol details due to limited space.

Meanwhile, RRect suits not only for $1+1$ mode, but also for $\alpha + \beta$ mode, such that the capital expenditure of deploying asymmetric redundancy scheme can be greatly lowered. Let $m = \alpha + \beta$, then in an $RRect(n,m,k)$, there are αn^{k+1} servers

TABLE 3
Topological Comparison between RRect and BCube

	servers (N)	diameter	server ports	switch ports	parallel paths	bisection bandwidth
$BCube(n,k)$	n^{k+1}	$k+1$	$k+1$	n	$k+1$	$\frac{N}{2}$
$RRect(n,m,k)$	mn^{k+1}	$k+1$	$k+1$	mn	$k+1$	$\frac{N}{2}$

in normal mode and βn^{k+1} servers in standby mode. This can be simply configured as follows. If a server has an address with $a_0 < \alpha n$, set it as working server. Otherwise, set it to standby mode. Hence, each sub-group has α working servers and β standby servers, and once a working server fails, one of the standby servers in the same sub-group will start up and take over all the tasks left by the failed server. Similarly, in this configuration, all the working servers work independently of the standby servers and they work in the same way as all the standby servers are also set to working mode. Also, the standby server which warms up in case of working server failure shares all the features that the failed server has. Thus the standby server can fully take the place of the failed server without any impact on other working servers.

Provided these, we could further enhance network availability by simultaneously deploying both symmetric and asymmetric redundancy modes. In the $RRect(n,m,k)$, where $m = \alpha + \beta$ as discussed above, if $\alpha > 1$, every α working servers form a group following the rules discussed previously. In this way, servers in the same group share exactly the same features, and act as mirrors for each other. Hence, these working servers are organized in way of symmetric redundancy mode, and servers within a group behave as backup for each other. At the same time, the extra servers with addresses $a_k a_{k-1} \dots a_0$, where $a_0 \geq \alpha n$, form the $\alpha + \beta$ asymmetric redundancy mode within each $RRect(n,m,0)$ in $RRect(n,m,k)$. As a result, both asymmetric and symmetric redundancy modes can be achieved simultaneously. Once a working server fails, either a backup server can warm up or a mirror of the failed server can do the failover process. If the runtime workload is light, a server in the same group of the failed server can take the responsibility without warming up a standby server for the purpose of keeping power efficiency. On the other hand, if current load is heavy, a standby server should take the responsibility in order to maintain throughput performance. This flexibility of the hardware configuration sufficiently guarantees the availability of services provided by the DCN provider.

Therefore, RRect meets the availability requirements in numerous ways and it can be configured into different modes dynamically without any alteration on existing hardware systems. To the best of our knowledge, RRect is the first server-centric network structure that can be configured into either active-passive redundancy mode or active-active redundancy mode and the backup server can fully take place of the original server with little impact on other working servers. Thus, this feature makes RRect very empirical for enterprise DCN products where availability is a major concern.

Moreover, the PRRect structure introduced in the previous section naturally enjoys a similar availability as well. As in Fig. 4, it can be observed that making servers marked in dark gray into standby mode would not generate side effect on the remaining working servers. Servers falling into the same sub-group also share exactly the same properties. Hence, PRRect also holds a high availability.

Next, we study the performance of RRect with respect to several factors, that are critical to DCNs, through extensive simulations.

6 Performance Evaluations

In this section, we evaluate the performance of RRect by conducting comprehensive simulations. A comparison between BCube, DCell and several other structures has been studied in [12], which indicates that BCube gives an excellent performance in many critical metrics. Thus, in this section, we mainly compare RRect to BCube which has represented itself as the most typical server-centric network structure. We compare RRect with BCube in several critical metrics, including average path distance, graceful degradation in the presence of component failure, aggregation throughput and power consumption of the interconnection network.

A theoretical topological comparison, regarding critical metrics between RRect and BCube, is summarized in Table 3, which reveals that $BCube(n,k)$ and $RRect(n,m,k)$ share similar properties. The only difference is that $RRect(n,m,k)$ accommodates more servers by utilizing switches with more ports. From another point of view, constructed on the same switches, $RRect(n/m,m,k)$ holds less servers than $BCube(n, k)$. These are expected, since $RRect(n,m,k)$ generalizes $BCube(n,k)$ by introducing parameter m. Next, we evaluate the performance of RRect through extensive simulations.

Since we focus on topological properties of the structure in this paper, we ignore the detailed protocols and assume that packets are always routed along the shortest path among all the paths which are available between the source and destination servers. By saying a path is available, we mean that all the components along that path, including switches, links and intermediate servers, are working normally. We also assume that packets will not be dropped due to congestion control, hence a packet will always arrive at its target server as long as there is at least one available path. Unless specially noted, all the following simulations follow the same assumption. To focus on the topological performance, we find the shortest available path in each structure by breadth first search algorithm (BFS). We first evaluate the average path distance between all pairs of servers in both RRect and BCube constructed by switches with the same number of ports, n. We separately set these two structures to $RRect(\frac{n}{2},2,4)$ and $BCube(n,4)$, both of which have a diameter of 5 hops, and vary n from 4 to 32. Fig. 7a depicts the resulted average shortest path between all pairs of servers in each structure. It can be observed from Fig. 7a that given that each structure is constructed by switches with the same number of ports, RRect gives a shorter average path distance compared to BCube. Therefore, all the servers in RRect enjoy shorter communication latency. Meanwhile, we can see that the average path

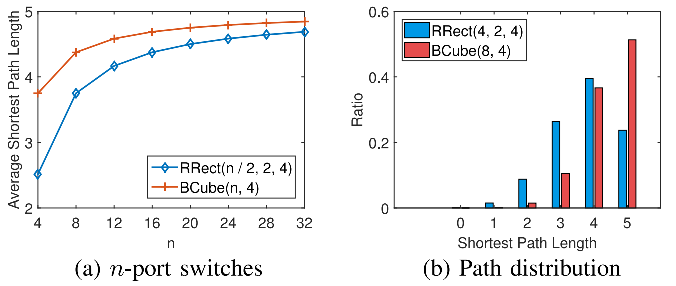


Fig. 7. Average path length.

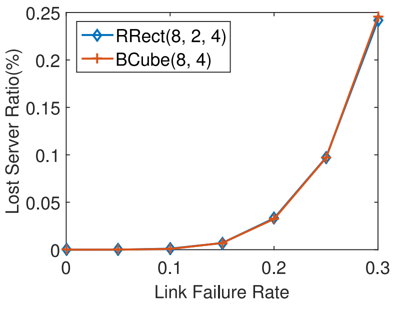


Fig. 9. Percentage of lost servers in the presence of link failure.

distance for either RRect or BCube increases as n grows. However, although the gap between RRect and BCube shrinks as n increases, RRect always gives the shorter average path distance.

We also study the shortest path distribution in $RRect(4, 2, 4)$ and $BCube(8, 4)$ which is presented in Fig. 7b. It can be observed that RRect gives more balanced path length distribution. In BCube, more than 51.29 percent paths have a length of 5, the diameter of $BCube(8, 4)$. On the other hand, in RRect, 23.73 percent paths have a length of 5 and 39.55 percent paths have a length of 4. This result indicates that although $RRect(4, 2, 4)$ shares the same diameter with $BCube(8, 4)$, it will give a shorter average communication latency than $BCube(8, 4)$.

Since component failure becomes normal as the network size of DCN continues increasing. Next, we evaluate the graceful degradation performance of RRect in the presence of component failure, including server failure and link failure. We first investigate the average path distance between all pairs of servers in both $RRect(8, 2, 4)$ and $BCube(8, 4)$ in the presence of server failure, where we assume that each server fails independently of others with equal failure rate ranging from 0 to 0.3. The simulation results are recorded in Fig. 8a, which indicates that the average path length in $RRect(8, 2, 4)$ increases much slower than that in $BCube(8, 4)$. The average path length in $RRect(8, 2, 4)$ increases from 4.375 to 4.3754, as server failure rate grows from 0 to 0.3. On the other hand, the average path length in $BCube(8, 4)$ increases from 4.375 to 4.3953. This result is expected. As discussed earlier, servers in the same sub-group in RRect behave as mirrors to each other. Thus, the distance between server A and any other server B is the same as that between server A', that belongs to the same sub-group as server A, and server B. Therefore, if server A fails, other servers can route via server A' to get a path with the same distance. Thus, RRect gives a more robust performance against server failure and all servers in RRect enjoy stabler communication latency than BCube.

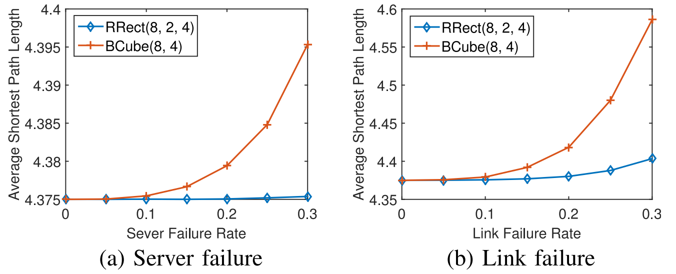
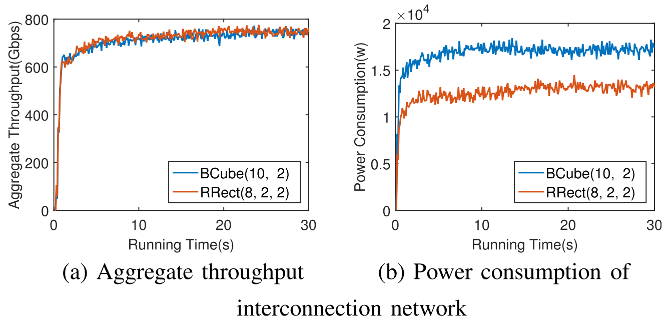


Fig. 8. Average path length in the presence of component failure.

Compared to server failure, degradation performance in the presence of link failure gives more insight, since server failure can be treated as that all the links connected to that server are cut down. Moreover, many other component failure can also be simulated into a link failure model. Due to its importance, next, we evaluate the graceful degradation performance of RRect in the presence of link failure. Similar to the simulations for server failure, we also assume that each link fails independently of each other with equal failure rate ranging from 0 to 0.3. We still set the network sizes to $RRect(8, 2, 4)$ and $BCube(8, 4)$, and investigate the average path distance of each structure against link failures. We present the simulation results in Fig. 8b, from which we can see that, the average path length in $RRect(8, 2, 4)$ increases only slightly from 4.375 to 4.4035, while that in $BCube(8, 4)$ increases much faster from 4.375 to 4.5859, as link failure rate grows from 0 to 0.3. Therefore, similar to the results for server failure, RRect gives a tremendous improvement on graceful degradation performance than BCube, which is expected for the same reason discussed above.

We also further study the structure connectivity in the presence of link failure. We say a server is lost if the destination server cannot be reached from the source. In other words, there are no available paths between the source server and the destination server. In this part, we compute the percentage of lost servers to the network size as link failure rate raises from 0 to 0.3. Fig. 9 presents the simulation results, which shows that $RRect(8, 2, 4)$ gives almost the same percentage of lost servers as $BCube(8, 4)$. The percentage of lost servers in both of RRect and BCube increases to 0.25 percent, as link failure rate grows from 0 to 0.3. The reason for these results is that although RRect provides more robust graceful degradation performance against component failures compared to BCube given the same network order k, the number of links connected to each server in RRect remains the same as that in BCube. In our experiments, each server in either $RRect(8, 2, 4)$ or $BCube(8, 4)$ is equipped with five NIC ports and five links connect to it from five different switches. In most scenarios, a server will be lost only if all of the links it has are cut off. Since each link fails independently of others, the probability of server being lost in BCube and RRect should be the same and small.

It should be pointed out that RRect achieves better performance, including shorter average path length, more balanced path distribution and more graceful degradation, by sacrificing its network size. In previous evaluations, $RRect(4, 2, 4)$ accommodates 2,048 servers while $BCube(8, 4)$ holds 32,768 servers, significantly larger than RRect. The comparisons we have conducted for graceful degradation,



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

Fig. 10. Aggregate throughput and power consumption of interconnection network under the traffic load generated by traffic characteristics.

on the other hand, assume that the two structures are constructed by switches with different number of ports. $RRect(8,2,4)$ uses 16-port switches and holds 65,536 servers, while $BCube(8,4)$ chooses 8-port switches and accommodates 32,768 servers. If we use the same 16-port switches to build a $BCube(16,4)$, it will hold 1,048,576 servers, significantly larger than that $RRect(8,2,4)$ holds. However, 1,048,576 servers are usually too large for most of today's DCN products, maybe even too large for DCN products in the future. On the other hand, $RRect(8,2,4)$ with 65,536 servers acts more suitable for an empirical DCN product. Besides, with current technology, a switch can be very cost efficiently integrated with enormous ports. Hence, a large network size of RRect can be easily derived by using switches with a large number of ports. Additionally, RRect has a network size increasing exponentially to its network order k. Thus, if needed, RRect can be scaled up to higher orders to meet the network size requirements. Moreover, since BCube is a special case of RRect, as discussed earlier, RRect can also be re-configured to a BCube for the purpose of network size. Therefore, given the same switches, although BCube could be much larger, RRect also provides a reasonable network size to meet the requirements.

Since throughput acts as a very critical metric for a DCN structure, finally we study the performance of aggregate throughput in RRect. We build our own simulator in C++ for the purpose of evaluating the throughput and power consumption of the interconnection network, which uses RRectRouting proposed in Algorithm 1 to find paths between any pair of servers. In this evaluation, we set the traffic speed of all components, including links, NIC ports and switch line cards, to 1 Gbps and set the buffer length of each link to infinite to eliminate the impact on the performance by the packet dropping algorithms and congestion control scheme. The structures are set to $RRect(8,2,2)$, accommodating 1,024 servers and constructed by 16-port switches, and $BCube(10,2)$, holding 1,000 servers and built from 10-port switches, so that both structures have a similar network size. We first evaluate the aggregate throughput of $BCube(10,2)$ and $RRect(8,2,2)$ under traffics, which are generated based on the traffic characteristics for DCNs summarized in [24]. The resulted throughput is presented in Fig. 10a which demonstrates that $BCube(10,2)$ and $RRect(8,2,2)$ give similar aggregate throughput. Both $BCube(10,2)$ and $RRect(8,2,2)$ provide an aggregate throughput around 740 Gbps. Therefore, when RRect and BCube are constructed in a similar network size with the same network order k, they give similar performance in aggregate throughput.

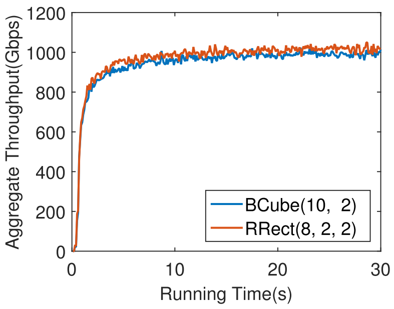


Fig. 11. Aggregate throughput in all-to-all communication pattern.

Meanwhile, we also calculate the power consumption generated by the interconnection network fabrics, the switches, of $BCube(10,2)$ and $RRect(8,2,2)$, which use 300 10-port switches and 192 16-port switches respectively. We calculate the energy consumption of switches based on the power consumption model discussed in Section 4. Since 10-port switches in $BCube(10,2)$ and 16-port switches in $RRect(8,2,2)$ are in the same order of magnitude in terms of number of ports, these two types of switches should have similar fixed energy cost for the fabrics overhead. Thus we separately set the overhead power consumption and per port consumption to 60 Watts and 2 Watts respectively. The simulations are also conducted under the same traffic environment as the simulations for the throughput above. To improve the power efficiency, we also assume that each port of a switch can be turned off independently if there is no traffic on it. In addition, if there is no traffic traveling through a switch, the entire switch can also be turned off to save more power. Fig. 10b shows the resulted power consumption, from which it can be observed that $RRect(8,2,2)$ consumes significantly less power than $BCube(10,2)$. $RRect(8,2,2)$ costs around 12.2 KWatts power, while $BCube(10,2)$ spends around 17.2 KWatts. Thus compared to $BCube(10,2)$, $RRect(8,2,2)$ saves almost 30 percent energy on interconnection networks, which slightly mismatches the conclusion from Section 4 for two reasons. First, Section 4 follows a traditional assumption that the overhead power in a switch occupies more than 75 percent of the total power, but in the simulations this proportion is less than 75 percent. Therefore, the value we choose for simulations may not be very representative. Moreover, the size of $RRect(8,2,2)$ is not exactly the same as that of $BCube(10,2)$. Hence, 30 percent is an expected ratio, which is still a considerable improvement.

To obtain a more comprehensive view of the throughput of RRect, we further investigate the most stressful traffic pattern, all-to-all communication pattern, in which every server separately sends different packets to each of the remaining servers in the same network. We present the throughput results in Fig. 11, from which we can observe that, similar to the throughput results above, $RRect(8,2,2)$ gives the same aggregate throughput performance as $BCube(10,2)$. Both of $BCube(10,2)$ and $RRect(8,2,2)$ provide an excellent throughput around 1,000 Gbps.

Note that we have not evaluated the power consumption of the interconnection network under all-to-all communication pattern, as in this pattern, all the switches including all the ports will be working all the time and they should not be turn off during the communication. Thus the power consumption of the interconnection networks in each structure should comply with Eq. (4).

Regarding the construction capital expenditure, based on the cost model in [21], the cost of each switch is linear to the number of its ports and the cost of an NIC is linear to the number of its ports too. On the other hand, the cost of CPU cores depends on how much traffic needs to be processed in each server. Since RRect and BCube are constructed in the same network size and network order, the total NIC ports and switch ports, as well as CPU cores needed in each server, are the same. Thus the capital cost to build an RRect should be the same as BCube.

Note that in this paper, only two basic traffic patterns are considered for evaluation. As pointed in [19], traffic load in an empirical data center could be highly dynamic. Also, cloud computing applications recently require data center networks to achieve bandwidth/performance guarantee [20]. Therefore, in our future work, we plan to study the performance of RRect under the aforementioned scenarios. Also, DCell is not used for comparison in this paper for two reasons. First, comparison results in [12] reveal that although DCell could be power efficient, its other performance, including throughput and graceful degradation, is significantly worse than BCube. Thus, DCell may not be suitable for an empirical product. Moreover, if we compare RRect with DCell, under a similar network size and order, the number of ports in a switch in RRect is drastically different from that in DCell. In other words, the number of switch ports in switches in RRect and DCell will not be in the same order of magnitude. Thus, their fixed overhead power consumption cannot be similar. However, how to distinguish the overhead power consumption in switches in RRect from that in DCell remains an open question and requires future investigation. Therefore, DCell is not considered for evaluation. We leave it in future work as well.

In summary, RRect is as good as BCube in throughput performance, while RRect tremendously reduces the energy cost caused by the interconnection switches, without using any power-aware routing algorithm. Meanwhile, RRect provides much more graceful degradation performance in the presence of both server and link failures. Besides, the capital cost for constructing an RRect is no more than that for a BCube, given the same network sizes and orders. Moreover, RRect shares many good topological properties, such as short network diameter and multiple near-equal parallel paths, with BCube. Taking all these factors into consideration, RRect is more practical for industry solutions.

7 Conclusions

In this paper, we propose a cube based DCN structure, called RRect, which is a recursively defined topology. RRect has an excellent energy efficiency of its interconnection network. Given the same network size and order, without adopting any power aware routing algorithm, RRect consumes significantly less power than BCube. PCube scheme can also be deployed on RRect to save more energy. Moreover, RRect can be configured into either symmetric or asymmetric redundancy mode to improve system availability, which cannot be conducted by existing popular server-centric networks. Also, RRect provides more flexibility, in which by fine tuning these parameters, RRect gives more options for practical DCN products, taking both network size and power consumption into consideration. We also present routing algorithms for servers in RRect. The simulation results show that RRect provides a significantly better energy efficiency and much more graceful degradation performance. Meanwhile, RRect presents a similar performance to BCube in other critical metrics, such as diameter, multiple parallel paths, aggregate throughput and so forth. These features meet today's requirements for DCNs, and thereby make RRect a very empirical topology for enterprise data center products.

Acknowledgments

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

References

[1] Cisco Nexus 5548P, 5548UP, 5596UP, and 5596T Switches. (2016). [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. (2016). [Online]. Available: https://www.arista.com/en/products/7050x-series

[3] CloudEngine CE5800 Series Data Center Switches. (2016). [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. (2006). [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 4th 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] 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.

[11] A. Greenberg, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. A. Maltz, and P. Patel, "VL2: A scalable and flexible data center network," in *Proc. ACM SIGCOMM Conf. Data Commun.*, Aug. 2009, pp. 51–62.

[12] Z. Li and Y. Yang, "ABCCC: An advanced cube based network for data centers," in *Proc. IEEE 35th Int. Conf. Distrib. Comput. Syst.*, Jul. 2015, pp. 547–556.

[13] 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

[14] R. Mysore, A. Pamboris, N. Farrington, N. Huang, P. Miri, S. Radhakrishnan, V. Subramanya, and A. Vahdat, "PortLand: A scalable fault-tolerant layer 2 data center network fabric," in *Proc. ACM SIGCOMM Conf. Data Commun.*, Aug. 2009, pp. 39–50.

[15] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat, "Hedera: Dynamic flow scheduling for data center networks," in *Proc. 7th USENIX Conf. Netw. Syst. Des. Implementation*, 2010, pp. 19–19.

[16] B. Heller, S. Seetharaman, P. Mahadevan, Y. Yiakoumis, P. Sharma, S. Banerjee, and N. McKeown, "ElasticTree: Saving energy in data center networks," in *Proc. 7th USENIX Conf. Netw. Syst. Des. Implementation*, 2010, pp. 17–17.

[17] 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.

[18] E. Marcus and H. Stern, *Blueprints for High Availability*, 2nd ed. Hoboken, NJ, USA: Wiley, 2003.

[19] F. Liu, J. Guo, X. Huang, and J. C. S. Lui, "eBA: Efficient bandwidth guarantee under traffic variability in datacenters," *Trans. Netw.*, vol. 25, no. 1, pp. 506–519, Feb. 2017.

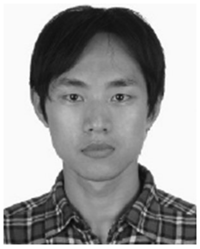
[20] J. Guo, F. Liu, T. Wang, and J. C. S. Lui, "Pricing intra-datacenter networks with over-committed bandwidth guarantee," in *Proc. USENIX Conf. Usenix Annu. Tech. Conf.*, 2017, pp. 69–81.

[21] 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.

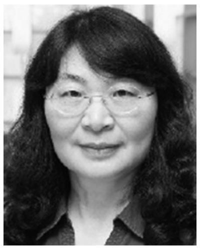
[22] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large cluster," in *Proc. Int. Parallel Process. Symp.*, Apr. 1996, pp. 789–795.

[23] S. Ghemawat, H. Gobioff, and S. Leung, "The Google file system," in *Proc. 19th ACM Symp. Operating Syst. Principles*, Oct. 2003, pp. 29–43.

[24] T. Benson, A. Akella, and D. Maltz, "Network traffic characteristics of data centers in the wild," in *Proc. 10th ACM SIGCOMM Conf. Internet Meas.*, Nov. 2010, pp. 267–280.



Zhenhua Li received the BEng and MS degrees in information and communication engineering from Zhejiang University, Hangzhou, China. He is working toward the PhD degree in 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 Academic Affairs with 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 the associate editor-in-chief of the *IEEE Transactions on Cloud Computing*. She has served as an associate editor-in-chief and associated editor of the *IEEE Transactions on Computers* and an associate editor of 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.