Computer Networks 56 (2012) 2132–2147

Contents lists available at SciVerse ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet

DPillar: Dual-port server interconnection network for large scale data centers ☆

Yong Liao [a], Jiangtao Yin [a], Dong Yin [b], Lixin Gao [a,*]

[a] Department of Electrical and Computer Engineering, University of Massachusetts at Amherst, Amherst, MA 01003, United States
[b] Automation Department, Northwestern Polytechnical University, Xi'an, ShanXi 710072, China

ARTICLE INFO

Article history:
Received 2 May 2011
Received in revised form 5 February 2012
Accepted 24 February 2012
Available online 6 March 2012

Keywords:
Data center network
Multi-path routing
Network topology

ABSTRACT

To meet the huge demands of computation power and storage space, a future data center may have to include up to millions of servers. The conventional hierarchical tree-based data center network architecture faces several challenges in scaling a data center to that size. Previous research effort has shown that a server-centric architecture, where servers are not only computation and storage workstations but also intermediate nodes relaying traffic for other servers, performs well in scaling a data center to a huge number of servers. This paper presents a server-centric data center network called *DPillar*, whose topology is inspired by the classic butterfly network. DPillar provides several nice properties and achieves the balance between topological scalability, network performance, and cost efficiency, which make it suitable for building large scale future data centers. Using only commodity hardware, a DPillar network can easily accommodate millions of servers. The structure of a DPillar network is symmetric so that any network bottleneck is eliminated at the architectural level. With each server having only two ports, DPillar is able to provide the bandwidth to support communication intensive distributed applications. This paper studies the interconnection features of DPillar, how to compute routes in DPillar, and how to forward packets in DPillar. Extensive simulation experiments have been performed to evaluate the performance of DPillar. The results show that DPillar performs well even in the presence of a large number of server and switch failures.

1. Introduction

The prevalence of cloud computing is driving the deployment of data centers to host various network applications and services [1,23,3]. In order to better support computation and storage demanding large scale distributed applications, as well as provide the multi-tenancy ability for efficient resource utilization and elastic resource allocation, data centers have to accommodate a large number of interconnected servers. A typical data center today has thousands of servers; a data center in the future can have hundreds of thousands or even millions of servers [17,15]. Although the availability of inexpensive commodity PCs has made it possible to expand a data center to a huge number of servers, efficiently interconnecting the servers is still a challenging task. The interconnection network has to provide high bandwidth to facilitate distributed applications requiring frequent data accessing and shuffling [11,9,13]. It is also often desired to use cost efficient commodity hardware in order to meet the budget constraints. The interconnection network should provide high availability as well because more and more enterprise-class mission critical applications are migrating into data centers. The conventional way of connecting multiple levels of switches into a tree and attaching servers as leaves of the tree [4] faces difficulties in scaling a data center to more than a few thousands servers [5]. The high

☆ Part of this work has been published in Proceedings of ICCCN 2010: the 19th International Conference on Computer Communications and Networks [21].

* Corresponding author. Tel.: +1 413 545 4548; fax: +1 413 545 1993.
E-mail addresses: yliao@ecs.umass.edu (Y. Liao), jyin@ecs.umass.edu (J. Yin), yindong@mail.nwpu.edu.cn (D. Yin), lgao@ecs.umass.edu (L. Gao).

1389-1286/$ - see front matter © 2012 Elsevier B.V. All rights reserved.
doi:10.1016/j.comnet.2012.02.016

speed core switches close to the tree's root will soon become the cost and performance bottleneck as the number of servers grows.

Researchers are actively searching for new network architectures to build cost efficient and high performance data centers. The VL2 network [15] uses a tree-based switch fabric to connect servers and provides a design with better agility property than conventional data center networks. The fat-tree network [5,22] is also a tree structure network. Instead of having expensive high-end switches at the tree's root, fat-tree uses identical switches at all levels of the tree. The number of servers connected by a fat-tree is determined by the number of ports on each switch.

To scale a data center without relying on switches with more ports, a second thread of research work proposes moving the networking intelligence from switches to servers, i.e., each server can forward traffic for other servers. Such a data center network is often referred to as a *server-centric* network. Besides the ability to easily scale the network size, a server-centric network offers other advantages in building large scale data centers. As the networking intelligence is moved to servers, switches can be simple layer-2 plug-and-play devices. Using dummy low-end switches can greatly reduce the cost of building a data center. Secondly, as servers are easier to program than switches, it is more convenient to develop and deploy routing and management mechanisms in server-centric data center networks. The DCell network [17] uses small switches to connect multi-port servers into "cells" and scales the network by recursively connecting smaller cells into larger cells. The network topology of DCell is not symmetric. Some links are more likely to be saturated and failures of those links are of greater impact to the network performance. The FiConn network [20] adopts similar idea as DCell to recursively expand the network size. Each server in FiConn needs to have only two ports. Because most off-the-shelf servers already integrate two ports, one primary and one backup, FiConn can use commodity servers in an as-is manner. FiConn is not a symmetric structure either and it has similar issues as DCell, e.g., some links will be more loaded than others. The BCube network [16] has symmetric network topology and it performs very well in terms of network bandwidth. In order to scale the network size, each server in BCube needs to have more ports.

In this paper, we propose a new server-centric data center network architecture called *DPillar*. DPillar uses only commodity hardware and can easily scale to a huge number of servers. No matter how many servers exist in a DPillar network, the number of ports in each server is always fixed. More specifically, with each server having only two ports, we can build a DPillar network to accommodate any number of servers. In contrast, both DCell and BCube require additional ports from servers in order to connect more servers into the networks, which could be a practical issue because off-the-shelf commodity servers always have a fixed number of ports. DPillar has a symmetric structure and therefore it eliminates any network bottleneck at the architectural level. The symmetric structure of DPillar facilitates the development of high performance routing mechanisms. In DPillar, servers forwarding traffic does not incur much overhead to them because there is no routing table lookup in packet forwarding. Instead, a server computes the next hop in $O(1)$ time based on its own address and destination address of the packet being forwarded. Even though each server has only two ports, a DPillar network offers rich connections between servers, which we have leveraged in designing an efficient multi-path routing scheme. This scheme produces multiple paths between a source–destination pair in a DPillar network. The disjointness of the yielded paths is formally proved in this paper. We also provide a design to utilize the disjoint paths to tolerate failures and balance load in a DPillar network.

DPillar is closely related to the classic wrapped butterfly network [19]. The design of our routing schemes is inspired by previous work on thewrapped butterfly network as well. It is worthy to highlight here a few essential differences between DPillar and thewrapped butterfly network. DPillar requires only two ports for each server but a node in a wrapped butterfly network needs to have four ports. More importantly, because of using switches instead of direct server-to-server links, a DPillar network has much smaller network diameter than a wrapped butterfly network of the same size. A DPillar network also provides much better bandwidth than a wrapped butterfly network of the same size. More details on comparing DPillar to a wrapped butterfly network will be presented in Section 2.6.

In summary, in this paper we make the following technical contributions in data center network design: (1) We propose a new server-centric data center network architecture with symmetric topological structure. A DPillar network can scale to a huge number of servers with each server having a fix number of ports. (2) We provide a comprehensive study of DPillar's interconnection properties. (3) We design a simple yet high performance multi-path routing scheme, and have provable path disjointness result applicable to a large set of related network topologies.

The rest of this paper is organized as follows. Section 2 presents the network structure of DPillar in detail and studies its topological properties. Sections 3 and 4 are devoted to the discussion of routing and packet forwarding in DPillar networks. Section 5 presents performance evaluation results. The background of interconnection networks and a discussion of related work on data center networks are presented in Section 6. Section 7 concludes this paper.

2. DPillar interconnection

This section presents the interconnection of DPillar. We first present how servers in DPillar are addressed and connected. Then we study the mathematical principles behind DPillar's structure and its topological properties, which serve as the foundation of designing packet routing and forwarding mechanisms for DPillar. A comparison between DPillar and the closed related wrapped butterfly network [19] is also presented in this section.

2.1. Logical representation of DPillar network

A DPillar network is built from two kinds of devices, dual-port servers and n-port switches. The servers and

switches are logically arranged into k equal-size server columns and k equal-size switch columns. Each server column has $\left(\frac{n}{2}\right)^k$ servers and each switch column has $\left(\frac{n}{2}\right)^{k-1}$ switches. The server columns and switch columns are numbered as $H_0 \sim H_{k-1}$ and $S_0 \sim S_{k-1}$, respectively. The server columns and switch columns are alternately placed into a logical circle, as shown in Fig. 1. Visually, the $2k$ columns of servers and switches are attached to the cylindrical surface of a *pillar*. For ease of exposition, in the rest of this paper we call server column $H_{(i+1)\%k}$ a *clockwise neighboring column* of H_i and $H_{(i+k-1)\%k}$ a *counter-clockwise neighboring column* of H_i. As we can see, a DPillar network is uniquely defined by two parameters, n, the number of ports in each switch, and k, the number of server columns. We call such a DPillar network an (n,k) DPillar network.

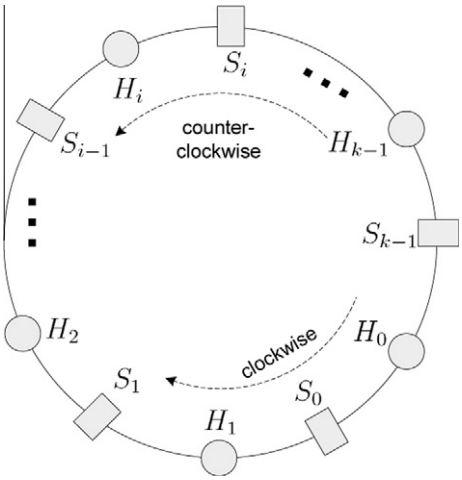

Fig. 1. A vertical view of the pillar. The blocks represent switch columns and the circles represent server columns. The server columns and switch columns are alternately placed into a large circle. It looks like they are attached to the cylindrical surface of a pillar.

2.2. Addressing servers in DPillar

For the $\left(\frac{n}{2}\right)^k$ servers in any server column H_i in an (n,k) DPillar network, each of them is assigned with a k-symbol label $(v^{k-1} \ldots v^0)$, where each symbol v^i is an integer number between 0 and $\left(\frac{n}{2}-1\right)$. Under this labeling scheme, one server in DPillar can be uniquely identified as $(C, v^{k-1} \ldots v^0)$, meaning a server with label $(v^{k-1} \ldots v^0)$ at server column H_C. We call $(C, v^{k-1} \ldots v^0)$ the address of the server.

Fig. 2 shows an $(8,2)$ DPillar network in a two-dimensional view. There are two server columns (H_0 and H_1) and two switch columns (S_0 and S_1). The label of a server has two digits, and each digit can be of value [0,3]. For

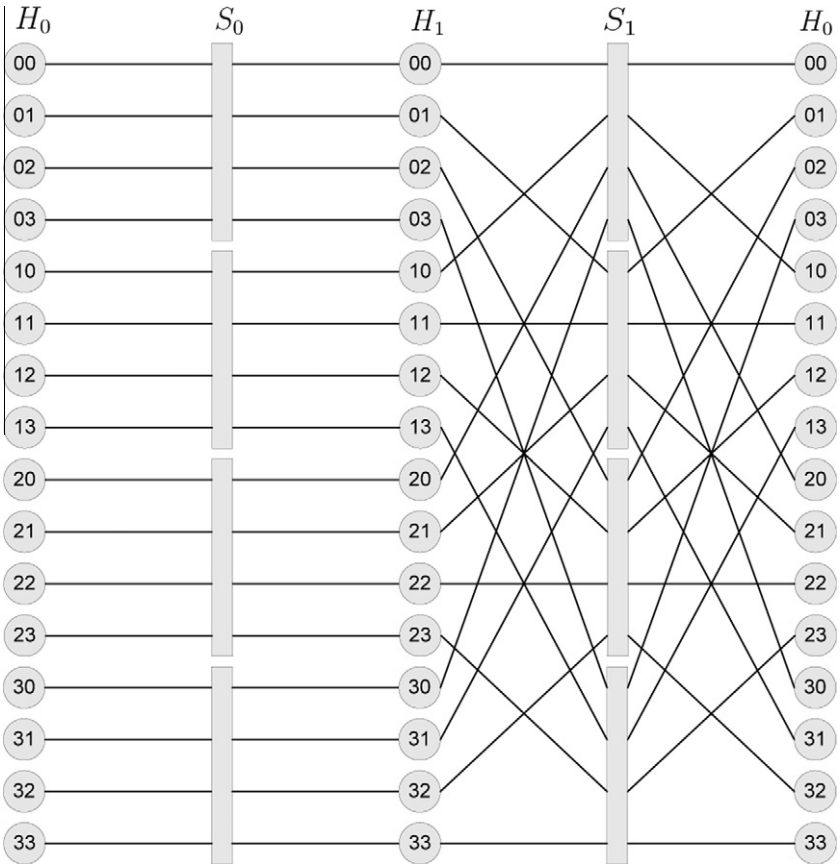

Fig. 2. Two-dimensional view of an $(8,2)$ DPillar network. The number in each circle is the label of that server. Note that server column H_0 is duplicated in this figure to better show the connection between server column H_0 and switch column S_1.

instance, the top left circle is a server at column H_0 and its label is (00), therefore its address is (0,00); the bottom left circle is a server of address (0,33).

2.3. Connecting servers via switches

Let us consider server columns H_i and $H_{(i+1)\%k}$. There are totally $2\left(\frac{n}{2}\right)^k$ servers in those two columns. Under the addressing scheme described in Section 2.2, those $2\left(\frac{n}{2}\right)^k$ servers can be divided into $\left(\frac{n}{2}\right)^{k-1}$ equal-size groups so that for those n servers in the same group, their labels are the same except the ith symbol (i.e., symbol v^i). The n servers in the same group are connected to the same switch in switch column S_i. It is easy to see that among the n servers in a group, half of them are in server column H_i and the other half are in $H_{(i+1)\%k}$.

For example, Fig. 2 shows the connection of servers and switches in an (8,2) DPillar network. For ease of illustration, we duplicate server column H_0 and plot the cylindrical surface of the logical pillar as a two-dimensional view. In this example, each server column has $\left(\frac{8}{2}\right)^2 = 16$ servers and the label of each server has two symbols. The address of each server can be represented as (C, v^1v^0). The first switch in S_1 is connected with eight servers, including four servers in H_1 and four servers in H_0. The labels of those servers are (00), (10), (20), and (30), respectively. That is, their labels are of form (v^10), with $0 \leqslant v^1 \leqslant 3$. Note that those labels are the same if v^1 is removed. Similarly, the labels of those eight servers connected to the first switch at S_0 are (00), (01), (02), and (03). They are of form $(0v^0)$ $0 \leqslant v^0 \leqslant 3$, and they are the same if v^0 is removed.

The rule of connecting servers via switches can be summarized as following. For any label $(v^{k-1} \ldots v^i \ldots v^0)$, there are $\frac{n}{2}$ servers in H_i whose labels are $(v^{k-1} \ldots v^i_* \ldots v^0)$, where $0 \leqslant v^i_* \leqslant \frac{n}{2} - 1$; there are $\frac{n}{2}$ such servers in $H_{(i+1)\%k}$ too. Those n servers are connected to the same n-port switch in switch column S_i. Given the way of how servers and switches are connected, the following Proposition 2.1 is obvious.

Proposition 2.1. *For two servers in any server column H_i, they are connected to the same switch if their labels differ at the ith symbol only (i.e., symbol v^i) or differ at the $((i-1)\%k)$th symbol only (i.e., symbol $v^{(i-1)\%k}$).*

2.4. Topological properties

For building large scale data centers, the interconnection network must accommodate a huge number of servers. The network should also provide sufficient bandwidth to support traffic intensive applications running in a data center. DPillar's topological structure does not impose any limit on the number of servers connected into the network. Hence, it is possible to scale a DPillar network into a huge number of servers. DPillar's topological structure also has good bisection width because there are rich connections among servers.

2.4.1. Number of servers

For an (n,k) DPillar network, since each server column has $\left(\frac{n}{2}\right)^k$ servers, there are $k\left(\frac{n}{2}\right)^k$ servers in total. In the rest of this paper, we use N to represent the total number of servers in an (n,k) DPillar network and we have Proposition 2.2.

Proposition 2.2. *An (n,k) DPillar network has $N = k\left(\frac{n}{2}\right)^k$ servers.*

Because the total number of servers, N, grows exponentially as the number of server columns, k, grows, DPillar structure scales well in terms of accommodating a large number of servers. Here we provide some examples on how many servers an (n,k) DPillar network can support. Considering that 48-port Gbit Ethernet switches are widely available now and relatively inexpensive, we assume 48-port switches are used in building DPillar networks. A (48,3) DPillar network has 41,472 servers. The number of servers can be about 1.3 million in a (48,4) DPillar network. If we build a (48,5) DPillar network, it has around 40 million servers.

2.4.2. Bisection width

Bisection width is an important factor to quantify an interconnection network's bandwidth. It is defined as the smallest number of edges removal of which divides the nodes in the network into two parts of equal size. Larger bisection width means that the network can sustain more communications between nodes in the network. Because servers in a data center usually have lots of interactions among them when running distributed applications, it is desirable for a data center network to have large bisection width.

We use the example shown in Fig. 2 to study how to cut a DPillar network into two halves. In Fig. 2, the servers in this (8,2) DPillar network can be divided into a top half and a bottom half so that for all servers in the top half, the v^1 symbol in their labels satisfies $0 \leqslant v^1 \leqslant 1$; for servers in the bottom half, the v^1 symbol in their labels satisfies $2 \leqslant v^1 \leqslant 3$. This scheme cuts four links for each switch in S_1 and the size of the cut is 16. In general, we can cut an (n,k) DPillar network into a top half and a bottom half so that for all servers in the top half, the v^{k-1} symbol in their labels satisfies $0 \leqslant v^{k-1} \leqslant \frac{n}{4} - 1$; for servers in the bottom half, the v^{k-1} symbol in their labels satisfies $\frac{n}{4} \leqslant v^{k-1} \leqslant \frac{n}{2} - 1$. This scheme cuts $\frac{n}{2}$ links for each n-port switch in switch column S_{k-1}. As there are $\left(\frac{n}{2}\right)^{k-1}$ switches in S_{k-1}, the size of the cut is $\left(\frac{n}{2}\right)^k$. Therefore, the upper bound of the bisection bandwidth of an (n,k) DPillar network is $\left(\frac{n}{2}\right)^k$. We can prove that $\left(\frac{n}{2}\right)^k$ is also the lower bound. This is stated as Proposition 2.3. The proof of Proposition 2.3 is presented in Appendix A.

Proposition 2.3. *The bisection width of an (n,k) DPillar network is $\left(\frac{n}{2}\right)^k$.*

2.5. Cost efficiency

DPillar network is cost-efficient as it uses commodity hardware. Here we provide some budget examples of building DPillar networks. We ignore the cost of servers and focus on the networking devices, including switches

Table 1
Example budget for networking cost when building a DPillar network with four columns of servers.

Switch type	8-Port	16-Port	24-Port	48-Port
Switch unit price ($)	50	150	180	600
Number of servers	1024	16,384	82,944	1,327,104
Total networking cost ($)	14,848	339,968	1,410,048	35,831,808
Per server cost ($)	14.5	20.8	17	27

and Ethernet cables. As most off-the-shelf servers already integrate dual-port network interfaces, there is no need to invest on network interfaces for servers.

An (n,k) DPillar network has k switch columns and each one includes $\left(\frac{n}{2}\right)^{k-1}$ n-port switches. The total number of switches is $k\left(\frac{n}{2}\right)^{k-1}$. There are $k\left(\frac{n}{2}\right)^{k}$ servers and each one has two ports, so the total number of cables used in an (n,k) DPillar network is $2k\left(\frac{n}{2}\right)^{k}$. Let U_s be the unit price of an n-port switch and U_c be the unit price of an Ethernet cable, the total networking cost of an (n,k) DPillar network is $\left(U_s \times k\left(\frac{n}{2}\right)^{k-1} + U_c \times 2k\left(\frac{n}{2}\right)^{k}\right)$. The average cost of connecting one server in a DPillar network is $2(U_s/n + U_c)$.[1]

The unit prices we get from an online retailing store are $150 for a 16-port Gbit Ethernet switch and $1 for an Ethernet cable. We expect the wholesale price of the switches and cables would be even lower. To build a (16,4) DPillar network, the cost of all switches is $307,200. The cost of cables is $32,768, as we need two cables for each server. The total cost of networking devices is about 0.34 million USD. On average it costs about $20 to connect one server. Table 1 shows the total cost and the per server cost of building DPillar networks with four columns of servers, when different types of switches are used.

2.6. Contrasting DPillar with wrapped butterfly network

DPillar is closely related to the *wrapped butterfly network* [19], which is one of the multi-stage interconnection networks studied before. The "column" in DPillar is equivalent to the "stage" in the wrapped butterfly network. DPillar can be viewed as an extension of the wrapped butterfly network, but DPillar has a set of unique properties which make it more suitable in building large scale data centers. Here we highlight the important differences between DPillar and the wrapped butterfly network.

2.6.1. Number of ports in servers

In order to connect servers via a wrapped butterfly network, the servers must have four ports. Because most commodity servers and servers in existing data centers integrate only two ports, we have to physically upgrade the servers if using a wrapped butterfly network to interconnect them. Although the cost of additional network interfaces is not an issue, installing those interfaces in a large number of servers can be quite time and manpower consuming. By connecting the servers via switches, DPillar can use off-the-shelf and existing dual-port servers to build a scalable data center network.

2.6.2. Network diameter

A wrapped butterfly network with k' column can accommodate $N' = k' \times 2^{k'}$ servers, while the number is $N = k \times \left(\frac{n}{2}\right)^{k}$ for an (n,k) DPillar network. Assuming $N' = N$, i.e., connecting the same amount of servers in a wrapped butterfly network and a DPillar network, we have

$$\frac{k'}{k} \approx \frac{\log\left(\frac{n}{2}\right)}{\log(2)} = \log_2(n) - 1.$$

We will show later in Section 3 that the diameter of DPillar (the max path length between two servers in DPillar) is a linear function of k, the number of server columns in the network. The linear relationship between k' and the network diameter also holds for a wrapped butterfly network. Therefore, to interconnect the same amount of servers, the diameter of wrapped butterfly is about $(\log_2(n) - 1)$ times the diameter of DPillar. We see that because of using switches, a DPillar network can have much more servers in one server column than the wrapped butterfly network. Hence, DPillar network scales the number of servers with much less server columns than the wrapped butterfly network does and results in much shorter paths when forwarding traffic between servers in the network.

2.6.3. Network bisection width

Using switches to connect servers also yields larger bisection width in DPillar, as compared to wrapped butterfly. The bisection width of an (n,k) DPillar network is equal to the number of servers in each server column. A wrapped butterfly network also has this property [19]. Each server column has $2^{k'}$ servers in a wrapped butterfly; while each server column has $\left(\frac{n}{2}\right)^{k}$ servers in DPillar. If connecting the same amount of servers in a wrapped butterfly network and a DPillar network, i.e., $N' = N$, we have

$$\frac{\left(\frac{n}{2}\right)^{k}}{2^{k'}} = \frac{k'}{k} \approx \log_2(n) - 1.$$

In other words, DPillar's bisection width is about $(\log_2(n) - 1)$ times the bisection width of a wrapped butterfly. For example, if using 48-port switches, a DPillar network's bisection width is about 4.5 times the bisection width of a wrapped butterfly network accommodating the same amount of servers.

3. Single-path routing

By considering the network structure and addressing the nodes in an intelligent way, one can design efficient routing in a network with symmetric structure. We will show in this section that routing and packet forwarding in DPillar can be quite straightforward because of the

[1] If we ignore the cost of the cables, the average cost of connecting a server in a DPillar network is two times the per-port cost of the switches used in this DPillar network.

way of interconnecting servers. A server can determine the nexthop by simply replacing some bits in its address with the corresponding bits in the destination address of the packet being forwarded. Therefore, servers can avoid expensive table lookup operations when forwarding packets. The routing scheme described in this section computes a unique path from a source server to a destination server in DPillar. This single-path scheme serves as the basis for our DPillar multi-path routing scheme presented later in Section 4.

3.1. Two-phase packet forwarding

As discussed in Section 2.3, each switch in an (n,k) DPillar directly connects n servers in its two neighboring server columns, and the labels of those n servers are the same if one of the $\frac{n}{2}$ symbols is removed. This property ensures that a server u in DPillar can always directly reach another server v in its neighboring server column, where the label of v has $\frac{n}{2}-1$ symbols in common with u's label and the other symbols of v can be of any value in $[0,\frac{n}{2}-1]$. Assuming u is at server column H_i, u can forward a packet to server v at $H_{(i+1)\%k}$, where the ith symbol of server v's label and the destination server's label are the same. Similarly, v can forward the packet to another server w at $H_{(i+2)\%k}$, where the $(i+1)\%k$th symbol of w is the same as that of the destination server's label. By keeping doing this, the packet can be forwarded to a server whose label is the same as the destination's label within $\frac{n}{2}$ hops. Note that in two neighboring columns, servers of the same label are also directly connected by switches, the packet can be sent to its destination by always forwarding to a nexthop server with the same label and whose column is closer to the destination server's column.

Based on the rationale described above, packet forwarding in DPillar can be divided into two phases. In the first phase, the packet is forwarded from the source server to an intermediate server whose label is the same as the destination server's label. In the second phase, the packet is forwarded from that intermediate server to the destination. In the following, we consider a scenario where a source server s sends a packet to a destination server d. The addresses of those two servers are (C_s, L_s) and (C_d, L_d), where L_s and L_d are the k-symbol labels of server s and d, respectively. Let the labels of those two servers be $L_s=(v_s^{k-1}\dots v_s^0)$, $L_d=(v_d^{k-1}\dots v_d^0)$, and $L_s\neq L_d$.

(1) *Phase one – helix phase:* From server s, which is in column H_{C_s}, the packet is sent to a server s_1 in column $H_{(C_s+1)\%k}$. The label of s_1 is the same as the label of s except the C_sth symbol in s_1's label can be any number from 0 to $(\frac{n}{2}-1)$. If server s_1 sends the packet to server s_2 in column $H_{(C_s+2)\%k}$, s_2's label is the same as s_1's except for that the $((C_s+1)\%k)$th symbol of s_2's label, which can be any number from 0 to $(\frac{n}{2}-1)$. We see that when a packet is forwarded for one hop, we can "change" one symbol in the label of the server which receives that packet. When a packet is always forwarded from one server column to its clockwise neighboring server column within k hops, the packet can reach a server with any given label. For example, in an (n,k) DPillar network, the trace of a packet forwarded from $(0,0\dots0)$ to $(k-1,1\dots1)$ can be $(0,0\dots0)\to(1,0\dots1)\to(2,0\dots11)\to(k-1,1\dots1)$. As this path resembles a helix among the cylindrical surface of the virtual pillar, we call the first phase of the packet forwarding process the *helix phase*.

Note that in the helix phase, we can always send the packet to either a server in the clockwise neighboring column or a server in the counter-clockwise neighboring column. However, the direction of forwarding a packet should not be changed back and forth in order to avoid loops. Hence, one field in the packet header is used to record the forwarding direction, and the source server decides the forwarding direction as clockwise or counter-clockwise.

(2) *Phase two – ring phase:* After the packet is forwarded to an intermediate server d^* whose label is the same as the label of destination server d, one can forward the packet to d by always sending it to the server in the clockwise neighboring column whose label is L_d too, or sending along the counter-clockwise direction. We choose the same direction as the helix phase in our forwarding process. We see that in this phase of packet forwarding, the packet forwarding path is like a segment in a ring among the cylindrical surface of the virtual pillar. We call the second phase the *ring phase*.

Algorithm 1. *DPillarSP*$((C_c,L_c),(C_d,L_d),D)$

```
/* L_c = (v_c^{k-1} ... v_c^0), L_d = (v_d^{k-1} ... v_d^0).*/
C_P ← (C_c + D + k)%k;
if L_c==L_d then        /*The ring phase.*/
    L_P ← L_d;
else        /* The helix phase. */
    if D==1 then
        L_P ← (v_c^{k-1} ... v_d^{C_c} ... v_c^0);
    else
        L_P ← (v_c^{k-1} ... v_d^{(C_c-1+k)%k} ... v_c^0);
return (C_P,L_P);
```

The pseudo-code of this single-path routing algorithm, denoted as DPillarSP, is shown in Algorithm 1. This algorithm takes the address of the current server (C_c,L_c), the destination server's address (C_d,L_d), and the forwarding direction D as input parameters. $D=1$ means the direction is clockwise; $D=-1$ indicates the counter-clockwise direction. The output is the address of the nexthop server (C_P,L_P). DPillarSP is loop free, and there is no need to maintain any routing table in the servers. Each server can determine the nexthop of a packet in constant time, independent of how many servers there are in a DPillar network. In the rest of this paper, we call a path generated by DPillarSP the DPillarSP path.

3.2. Length of the path computed by DPillarSP

A path derived from DPillarSP is obviously not the shortest one. However, DPillarSP always yields paths with

bounded length. We consider the worst case scenario where the labels of two servers have no common symbols, in which case a packet needs to be forwarded k times in order to reach a server d^* whose label is the same as server d's label. After that, from server d^*, the packet still needs to be forwarded $(k-1)$ hops in order to reach server d in the worst case. Therefore, the length of the longest path computed by DPillarSP in an (n,k) DPillar network is $(2k-1)$.

Proposition 3.1. *Using DPillarSP, any two servers in an (n,k) DPillar network can reach each other in at most $2k-1$ hops.*

4. Multi-path routing

The migration of mission critical applications into data centers imposes stricter requirements to a data center network in terms of reliability, robustness, and performance. Multi-path routing has been shown to be an efficient mechanism to tolerate failures and balance traffic load in a network [18,12,6]. The rich connections inside a DPillar network and the flexibility of DPillar's server-centric architecture facilitate the design of efficient multi-path routing scheme. This section presents *DPillarMP*, a multi-path routing scheme for DPillar. In an (n,k) DPillar network, DPillarMP computes $\frac{n}{2}$ disjoint paths between a pair of source–destination servers. We will formally prove the correctness of our mechanism in computing disjoint paths, and provide a design to utilize the disjoint paths in bypassing failures and balancing traffic load in a DPillar network.

4.1. The basic idea

In DPillar, we define two paths from a source server s to a destination server d to be *node-disjoint* if they do not have any common servers or switches except the switch connected to s and the switch connected to d. As already shown in Section 3, a source server s can always send packets to a server at its clockwise neighboring column. In an (n,k) DPillar network, because server s has $\frac{n}{2}$ directly connected servers (via the same switch) at its clockwise neighboring column, s has $\frac{n}{2}$ possible next hops to forward packets. For the destination server d, there always exists some server at d's counter-clockwise neighboring column forwarding the packets to d, and d directly connects to $\frac{n}{2}$ servers at its counter clockwise neighboring column as well. We show an example in Fig. 3. The source s has $\frac{n}{2}$ next hops to forward the packet; similarly, there are $\frac{n}{2}$ "previous hops" which can deliver the packets to destination d.

Let s'_i be the $\frac{n}{2}$ next hops of source s where $0 \leqslant i \leqslant \frac{n}{2}-1$, and d'_j, $0 \leqslant j \leqslant \frac{n}{2}-1$ be the $\frac{n}{2}$ previous hops of destination d. If there is an algorithm to yield $\frac{n}{2}$ pairs of $\left(s'_i, d'_j\right)$, where $0 \leqslant i,j \leqslant \frac{n}{2}-1$ so that the paths between any two pairs do not have common intermediate servers or switches, we can have $\frac{n}{2}$ disjoint paths between source s and destination d. Next we study the property of disjoint paths in DPillar, and then present how to pair s'_i and d'_j to construct the node-disjoint paths.

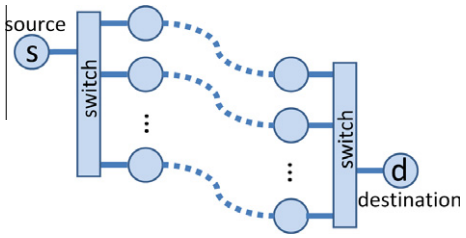

Fig. 3. In a (n,k) DPillar network, by pairing $\frac{n}{2}$ clockwise neighboring servers of the source s and the $\frac{n}{2}$ counter-clockwise neighboring servers of the destination d, we can yield $\frac{n}{2}$ disjoint paths between s and d.

4.2. Disjoint paths in DPillar

We use an example in Fig. 4 to demonstrate the disjointness of two paths. Here the source server is (0,00), and the destination server is (0,33), denoted as s and d in Fig. 4. We need to pick two next hop servers (denoted by x_1 and x_2) for source s and two previous hop servers (denoted by y_1 and y_2) for destination d so that the path between x_1 and y_1 does not share common intermediate nodes with the path between x_2 and y_2. In this example, we select server (1,00) and (1,01) as the two next hops of source s; and select (1,13) and (1,23) as the two previous hops of destination d. The DPillarSP path from x_1 to y_1 is $(1,00) \to (0,10) \to (1,13)$; the DPillarSP path from x_2 to y_2 is $(1,01) \to (0,21) \to (1,23)$. One can see that these two paths do not have any common intermediate servers. Let us have a closer look at the addresses of x_1, x_2, y_1, and y_2. Because x_1 and x_2 have different symbol v^0, their next hops according to Algorithm 1 should not be the same since the v^0 symbols of those two servers' labels are different. In general, the addresses of x_1, x_2, y_1, and y_2 should be chosen in a way that when the paths are computed by Algorithm 1 and two intermediate servers are at the same column, their labels are different.

Based on the intuition described above, we have identified a sufficient condition for two DPillarSP paths between two next hops of the source and two previous hops of the destination to be disjoint. Without loss of generality we consider a scenario where source server s is at column H_s and destination server is at column H_d. Let x_1 and x_2 be two clockwise neighboring servers of s at column H_{s+1}; y_1 and y_2 be two counter-clockwise neighboring servers of d at column H_{d-1} (the modulo operations are omitted here for simplicity of expression). Since x_1 and x_2 are connected to the same switch at switch column S_s, their labels are the same except for the sth symbol. Their addresses can be represented as $\left(s+1, v_x^{k-1} \ldots v_{x_1}^s \ldots v_x^0\right)$, and $(s+1, v_x^{k-1} \ldots v_{x_2}^s \ldots v_x^0)$, respectively, where $v_{x_1}^s \neq v_{x_2}^s$. Similarly, y_1 and y_2 are connected to the same switch at switch column S_{d-1}, so their labels are the same except for the $(d-1)$th symbol. We use $(d-1, v_y^{k-1} \ldots v_{d_1}^{d-1} \ldots v_d^0)$ and $(d-1, v_y^{k-1} \ldots v_{d_2}^{d-1} \ldots v_d^0)$ to represent their addresses, where $v_{d_1}^{d-1} \neq v_{d_2}^{d-1}$. The following Corollary 4.1 describes a sufficient condition for the DPillarSP paths between those two pairs of servers to share no common servers or

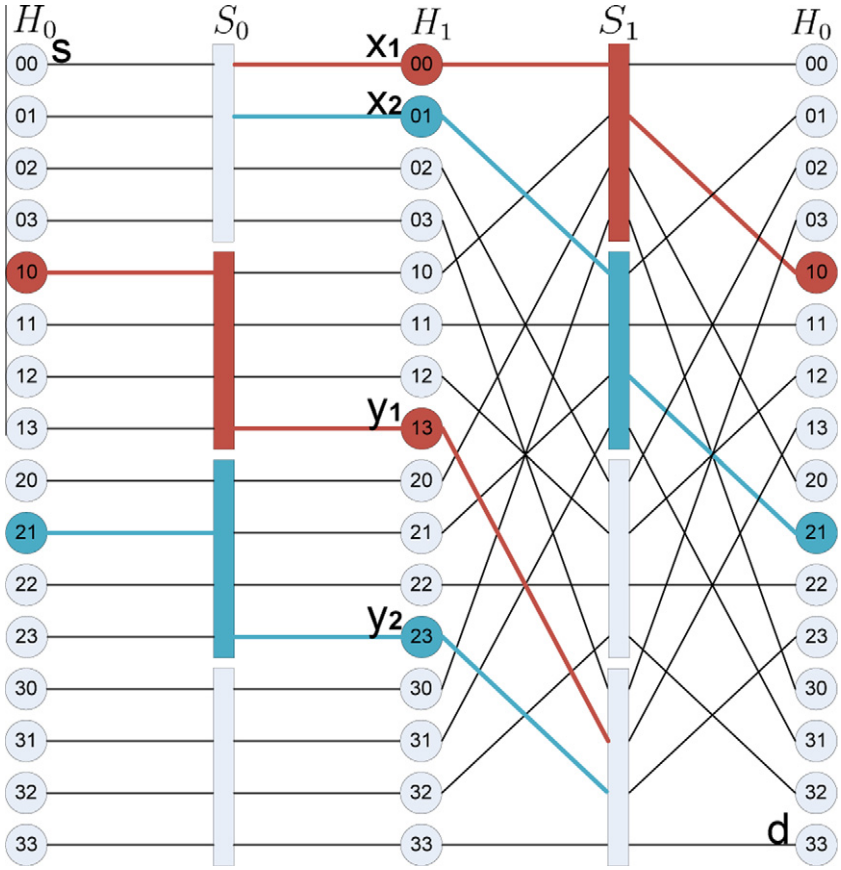

Fig. 4. Two paths with no common servers or switches in an (8,2) DPillar network.

switches, assuming that the DPillarSP paths follow the clockwise direction.

Corollary 4.1. *Let P_1 be the DPillarSP path from server $\left(s+1, v_x^{k-1}\dots v_{x_1}^s\dots v_x^{d-1}\dots v_x^0\right)$ to server $\left(d-1, v_y^{k-1}\dots v_y^s\dots v_{y_1}^{d-1}\dots v_y^0\right)$; and P_2 be the DPillarSP path from server $\left(s+1, v_x^{k-1}\dots v_{x_2}^s\dots v_x^{d-1}\dots v_x^0\right)$ to server $\left(d-1, v_y^{k-1}\dots v_y^s\dots v_{y_2}^{d-1}\dots v_y^0\right)$. P_1 and P_2 do not share any common servers or switches, if $v_{x_1}^s \neq v_y^s$ or $v_x^{d-1} \neq v_{y_2}^{d-1}$, and $v_{x_2}^s \neq v_y^s$ or $v_x^{d-1} \neq v_{y_1}^{d-1}$.*

Proof. For each path, we divide it into two segments, Seg_h and Seg_r. Seg_h includes the nodes (servers and switches) on the path from column H_{s+1} to column H_s, and Seg_r includes remaining nodes on the path (from column H_{s+1} to column H_{d-1})[2].

When $d \leqslant s$: In the first segment of P_1, denoted by $Seg_h(P_1)$, the labels of all servers have the sth symbol to be $v_{x_1}^s$. Similarly, the labels of all servers in $Seg_h(P_2)$ have sth symbol to be $v_{x_2}^s$. Because $v_{x_1}^s \neq v_{x_2}^s$, these two segments have no common servers. Moreover, as the switch columns in those two segments are $S_{s+1}, S_{s+2}, \dots, S_{s-1}$, according to Proposition 2.1, we know servers in these two segments always connect to different switches when they are in the same server column. So these two segments have no common switches either.

In $Seg_r(P_1)$, all servers have the $(d-1)$th symbol to be $v_{y_1}^{d-1}$; all servers in $Seg_r(P_2)$ have the $(d-1)$th symbol to be $v_{y_2}^{d-1}$. Similarly, because $v_{y_1}^{d-1} \neq v_{y_2}^{d-1}$, these two segments also have no common servers or switches.

For $Seg_h(P_1)$ and $Seg_r(P_2)$, $v_{x_1}^s \neq v_y^s$ or $v_x^{d-1} \neq v_{y_2}^{d-1}$ ensures that they have no common servers or switches. Similarly, $v_{x_2}^s \neq v_y^s$ or $v_x^{d-1} \neq v_{y_1}^{d-1}$ ensures that $Seg_h(P_2)$ and $Seg_r(P_1)$ have no common servers or switches. Hence, the first segment of one path shares no common servers or switches with the second segment of another path.

When $d > s$: The proof is similar. □

Next we will show how to pair the $\frac{n}{2}$ clockwise neighboring servers of source s with the $\frac{n}{2}$ counter-clockwise neighboring servers of destination d, so as to yield $\frac{n}{2}$ *node-disjoint* paths between s and d.

[2] Note that we reuse the first letter of "helix" and "ring" here as the subscripts because Seg_h and Seg_r resemble the helix phase and the ring phase, respectively.

4.3. Path construction

Let source server s be at column H_s, destination server d be at column H_d, and their addresses be $(C_s, v_x^{k-1} \dots v_x^s \dots v_x^0)$, $\left(C_d, v_y^{k-1} \dots v_y^d \dots v_y^0\right)$, respectively. The addresses of the $\frac{n}{2}$ clockwise neighbors of the source are $(s+1, v_x^{k-1} \dots v_{x_i}^s \dots v_x^{d-1} \dots v_x^0)$, where $i \in [0, \frac{n}{2}-1]$ and $v_{x_i}^s = i$. The destination server has $\frac{n}{2}$ counter-clockwise neighbors at column H_{d-1}, whose addresses are $(d-1, v_y^{k-1} \dots v_y^s \dots v_{y_j}^{d-1} \dots v_y^0)$, where $j \in [0, \frac{n}{2}-1]$ and $v_{y_j}^{d-1} = j$.

For those $\frac{n}{2}$ servers at H_{s+1} and $\frac{n}{2}$ servers at H_{d-1}, we first pair server $(s+1, v_x^{k-1} \dots v_{x_*}^s \dots v_x^{d-1} \dots v_x^0)$ with server $(d-1, v_y^{k-1} \dots v_y^s \dots v_{y_*}^{d-1} \dots v_y^0)$, where $v_{x_*}^s = v_y^s$ and $v_{y_*}^{d-1} = v_x^{d-1}$, then arbitrarily pair the rest servers at H_{s+1} with the other servers at H_{d-1}. This pairing scheme yields $\frac{n}{2}$ clockwise DPillarSP paths between the $\frac{n}{2}$ servers at H_{s+1} and $\frac{n}{2}$ servers at H_{d-1}.

Now we show that those $\frac{n}{2}$ paths between servers in H_{s+1} and $\frac{n}{2}$ in H_{d-1} do not share any common servers or switches. Let P be the path between server $(s+1, v_x^{k-1} \dots v_{x_*}^s \dots v_x^{d-1} \dots v_x^0)$ and server $(d-1, v_y^{k-1} \dots v_y^s \dots v_{y_*}^{d-1} \dots v_y^0)$, where $v_{x_*}^s = v_y^s$ and $v_{y_*}^{d-1} = v_x^{d-1}$. Let P' be another path between server $(s+1, v_x^{k-1} \dots v_{x_i}^s \dots v_x^{d-1} \dots v_x^0)$ and server $(d-1, v_y^{k-1} \dots v_y^s \dots v_{y_j}^{d-1} \dots v_y^0)$. Since $v_{x_*}^s \neq v_{x_i}^s$ and $v_{y_*}^{d-1} \neq v_{y_j}^{d-1}$, we have $v_y^s \neq v_{x_i}^s$ and $v_x^{d-1} \neq v_{y_j}^{d-1}$. According to Corollary 4.1, P and P' cannot have any common servers or switches. Similarly, we can derive that P' must be disjoint to any other path P'' too, where $P'' \neq P$.

Algorithm 2. *ConstructPairSet*((C_s,L_s), (C_d,L_d))

/* $(C_s, L_s) = (s, v_x^{k-1} \dots v_x^0)$ is the source,
$(C_d, L_d) = \left(d, v_y^{k-1} \dots v_y^0\right)$ is the destination. */
1 $SET_s = \left(s+1, v_x^{k-1} \dots v_{x_i}^s \dots v_x^0\right)$, $v_{x_i}^s = i \in [0, \frac{n}{2}-1]$;
2 $SET_d = \left(d-1, v_y^{k-1} \dots v_{y_i}^{d-1} \dots v_y^0\right)$, $v_{y_i}^{d-1} = i \in [0, \frac{n}{2}-1]$;
3 remove $s' = \left(s+1, v_x^{k-1} \dots v_y^s \dots v_x^{d-1} \dots v_x^0\right)$ from SET_s;
4 remove $d' = \left(d-1, v_y^{k-1} \dots v_y^s \dots v_x^{d-1} \dots v_y^0\right)$ from SET_d;
5 pair s' and d';
6 sort SET_s according to symbol $v_{x_i}^s$;
7 sort SET_d according to symbol $v_{y_i}^{d-1}$;
8 **while** SET_s and SET_d are not empty **do**
9 remove first elements in SET_s and SET_d, s' and d';
10 pair s' and d';

The idea described above leads to a straightforward algorithm in computing node-disjoint paths in DPillar. We show the pseudo code in Algorithm 2. This algorithm first pairs a server in SET_s with a server in SET_d, where their sth and $(d-1)$th symbols are the same. Then it sorts the remaining $(\frac{n}{2}-1)$ servers in SET_s and $(\frac{n}{2}-1)$ servers in SET_d according to one of the symbols and always pairs the first element in each set together, so as to yield deterministic pairing of all servers.[3] Algorithm 2 generates $\frac{n}{2}$ server pairs. All the clockwise DPillarSP paths between two servers of these server pairs do not share any common servers or switches. As we already discussed in Section 4.1, using those $\frac{n}{2}$ server pairs, we can have $\frac{n}{2}$ node-disjoint paths between source s and destination d. The correctness of Algorithm 2 in producing node-disjoint paths is formally stated as Theorem 4.2 and its detailed proof is provided in Appendix B.

Theorem 4.2. *For any two servers s and d in an (n,k) DPillar network, we pair the clockwise neighbors of s with the counter-clockwise neighbors of d by following Algorithm 2 to obtain $\frac{n}{2}$ clockwise DPillarSP paths between these neighbors. Those $\frac{n}{2}$ paths between s and d are node-disjoint.*

Note that according to Proposition 3.1, no DPillarSP path in an (n,k) DPillar network is longer than $2k-1$, so the length of these $\frac{n}{2}$ node-disjoint paths between the source server and the destination server is bounded by $2k+1$.

We have shown how to construct $\frac{n}{2}$ node-disjoint paths between a source and a destination in the clockwise direction. There are also $\frac{n}{2}$ counter-clockwise node-disjoint paths between the source and destination servers. That is, we pair the $\frac{n}{2}$ counter-clockwise neighbors of source with the destination's $\frac{n}{2}$ clockwise neighbors, and build $\frac{n}{2}$ node-disjoint counter-clockwise paths. The detailed scheme is omitted here as it is quite similar to the technique in building clockwise node-disjoint paths.

4.4. Packet forwarding

After constructing the multiple node-disjoint paths, the next question is how to enforce a packet to follow one of those paths. For a packet to follow a path from server s to server d, the packet traverses a path $s \to s' \dots d' \to d$, where s' and d' are two intermediate servers paired by the scheme described in Section 4.3. We use *tunneling* to embed the information of s' and d' into a packet header. The source selects a path among the set of node-disjoint paths and adds another header in front of the original packet, where the source and destination fields in the outer header are set to s' and d', respectively. We call the source address in the outer header a *proxy source*, and the destination address in the outer header a *proxy destination*. As the source server is connected directly with the *proxy source* server, the source simply forwards the packet to the *proxy source* server. Then the packet is forwarded according to the scheme specified in Algorithm 1 to the *proxy destination* server, which will decapsulate the packet and send it to the destination. Algorithm 3 shows the pseudo-code of how a server computes the nexthop in forwarding a packet based on the inner and outer packet headers.

[3] Note that this is purely for generating deterministic pairing of the servers. One can arbitrarily the remaining $(\frac{n}{2}-1)$ servers in SET_s with the $(\frac{n}{2}-1)$ servers in SET_d, and it does not affect the disjointness of those $\frac{n}{2}$ paths.

Algorithm 3. *Nexthop*(*c*,*pkt*)

```
/* c is the current server, and pkt is the packet
   to be routed.    */
/* pkt.src and pkt.dst are src/dst in inner
   header; pkt.psrc and pkt.pdst are the proxy
   src/dst in outer header; pkt.d is the
   forwarding direction. */
1 if pkt.dst==c then          /* Reach dst. */
2 |   return null;
3 else if pkt.src==c then     /* At src. */
4 |   return pkt.psrc;        /* To proxy src. */
5 else if pkt.pdst==c         /* At proxy dst. */
6 |   return pkt.dst;         /* To dst. */
7 else return DPillarSP (pkt.psrc, pkt.pdst, pkt.d);
```

4.5. Applications of multiple node-disjoint paths in DPillar

The multiple node-disjoint paths in DPillar can be utilized to improve the performance of a DPillar network in various aspects. Here we provide the sample designs on how to tolerate failures and balance traffic by using the multiple disjoint paths in DPillar.

4.5.1. Fault-tolerant multi-path routing

The multiple disjoint paths between two servers in DPillar can be utilized to route traffic in a fault-tolerant manner. When an application running in a source server requires fault-tolerant routing from DPillar, the source server builds multiple node-disjoint paths between itself and the destination server before the application starts to send traffic. After constructing the paths, the source server can pick one of them to send packets to the destination server. In order to tolerate failures, the source server should also proactively monitor the status of those node-disjoint paths by periodically sending probing messages on each path. After the destination server receives a probing message, it replies a probing response message to the source. Upon receiving the probing response message, the source knows that the path is working. If an intermediate server receives a probing message and its nexthop is unreachable, it returns a path failure message to the source. Upon receiving the path failure message, the source can switch the traffic to an alternative node-disjoint path.

We should note that monitoring the status of multiple paths incurs certain overhead to the network because of using probing messages. Probing paths also takes time as a message needs to do a round trip between source and destination servers. Therefore, this proactive probing scheme is more suitable for applications generating lots of bulk flows or long-haul flows. For small and short-lived flows, we can use a more light-weight scheme in utilizing the multiple node-disjoint paths. Each time an application in a source server generates traffic to a destination, it tries a few times in establishing a TCP connection to the destination.[4] When the source server gets the request from an application to connect with the destination using TCP, it randomly picks one of the node-disjoint paths to establish the connection. If the first try of TCP connection fails due to failure or congestion in the network and the application issues a re-try request, the source server randomly picks a path again. As there are multiple node-disjoint paths, it is very likely that the source server can find a path to bypass the failure or congestion point after a few tries.

4.5.2. Traffic-aware multi-path routing

The existing of multiple disjoint paths between two servers also opens the design dimension of balancing traffic load in DPillar. A source generating a bulk or long-haul flow can periodically send messages on the multiple node-disjoint paths to the destination, to probe the available bandwidth on each path, and schedule its flow to the path with the largest available bandwidth. Because the multiple paths are node-disjoint, we can easily avoid any hot-spot in the network.

5. Evaluation

We evaluate DPillar from two different aspects. First, we implement the DPillar packet forwarding mechanism in Click software router [2] and study the microscopic behavior of DPillar by measuring the packet forwarding overhead of one DPillar server. Our measurements show that packet forwarding does not cause too much CPU usage overhead to servers. Second, we study the macroscopic behavior of DPillar by simulating the packet routing and forwarding in DPillar networks, using a simulation tool we developed. Our simulation results show that our routing scheme performs well even in the presence of a large number of server and switch failures.

5.1. Implementation and testbed

We have implemented the DPillar routing algorithms presented in Sections 3 and 4 as an element in kernel mode Click software router [2]. Our implementation uses an IP address to encode the column and label information of one server, so as to provide backward compatibility to upper layer applications. The most significant 8 bits of the 32-bit IPv4 address are reserved to represent the column number of a server. As each host has two NICs, the least significant bit of the 32-bit IPv4 address is used to represent the NICs. The k-symbol label consumes $k\lceil \log_2(n) - 1\rceil$ bits. The 32-bit IPv4 address is allocated as shown in Fig. 5.

For example, when using 48-port switches to build a DPillar network with 4 columns of hosts (the total number of hosts is about 1.3 million), we need to use 2 bits in the most significant 8 bits of the IP header to represent the columns. In the remaining 24 bits, we use $4\lceil \log_2(48) - 1\rceil + 1 = 21$ bits to represent the label of servers and the interfaces.

Fig. 6 shows the testbed used in our experiments to evaluate DPillar routing and forwarding element implemented in kernel mode Click. Server P in Fig. 6 is a commodity PC with a 2.4 GHz dual-core CPU and 1 GB

[4] Existing measurement work shows that most of the traffic in data centers is TCP [7].

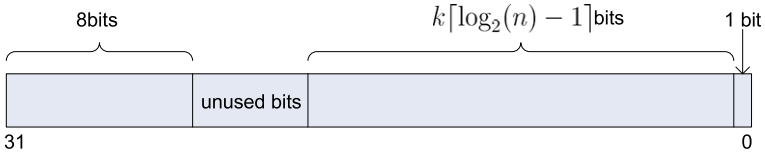

Fig. 5. Reusing bits of an IP address to represent the addresses of servers in DPillar.

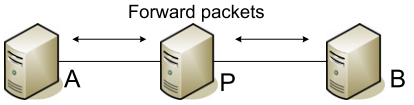

Fig. 6. The testbed network. Server *P* runs DPillar packet routing and forwarding Click element and forwards packets between server *A* and server *B*.

memory. Server *A* and *B* are similar machines pumping traffic into server *P*.

5.2. Overhead of forwarding packets

We measure its CPU usage when a DPillar server forwards traffic for other servers. In this experiment, server *A* and *B* in Fig. 6 use iperf to send out UDP traffic to each other at various speed, and server *P* forwards the traffic between *A* and *B*. We record the CPU usage of the Click kernel thread and plot the results in Fig. 7.

The results in Fig. 7 show that even when the DPillar server forwards traffic at full load, i.e., 1 Gbps each direction, 2 Gbps in total, the CPU usage of the DPillar server is less than 50%. Our server is a dual-core machine and the less than 50% CPU usage is for one CPU core; the other core is almost 100% idle. As commodity PCs with multi-core CPUs are becoming common, we expect the traffic forwarding overhead can be amortized so that only a small portion of the total processing power of a multi-core server is used in traffic forwarding.

It is well known that the amount of CPU cycles used to forward a packet does not depend on the packet length. Hence, in a server-centric data center network, one can reduce the CPU load and still maintain the throughput by using larger packets [16]. As *jumbo frame packet transfer* is commonly supported in commodity Ethernet switches,[5] we can take advantage of this feature to reduce the CPU usage of DPillar servers in traffic forwarding.

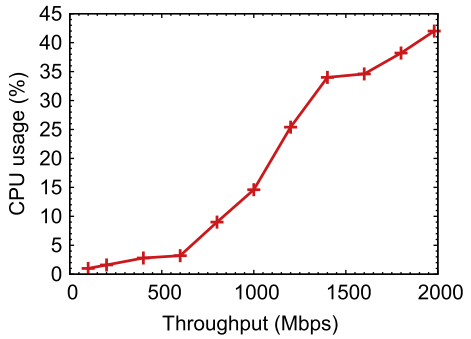

Fig. 7. The server CPU usage under various UDP traffic load. The packet size is 1024 bytes.

5.3. Path length

We have developed an event-driven simulation tool to study packet forwarding in DPillar networks. Given the number of server columns k and the switch port number n, our simulation tool builds an (n,k) DPillar network topology and simulates how each server routes packets.

5.3.1. Without failure

We first study the length of paths computed by DPillarSP and DPillarMP. We simulate the scenarios where the DPillar network is built from 16-port switches, i.e., $n = 16$, and the number of server columns k varies. For each network, we randomly select 10,000 source–destination pairs and simulate the paths yielded by DPillarSP and DPillarMP. Fig. 8 plots the results of the average length of the paths between those 10,000 source–destination pairs.

The results show that the path length is proportional to the number of server columns in a DPillar network, and it is always about 20% shorter than the maximum path length $(2k - 1)$ for DPillarSP and about 25% shorter than the maximum path length $(2k + 1)$ for DPillarMP. In addition, compared with DPillarSP, DPillarMP does not inflate the path length too much. The inflation is always within 2 hops.

5.3.2. With failure

We also study the average length of paths computed by DPillarMP scheme when there are server failures. Our simulation includes four DPillar networks of different sizes, i.e., (16,3) network, (24,3) network, (16,4) network and (24,4) network. In each simulation instance, we randomly fail a certain percentage of the total servers. We vary the ratio of failed servers from 0 to 0.2 (e.g., failure ratio of 0.1 means 10% of the servers have failed). Then we randomly select 10,000 source–destination pairs from those servers without failures and compute the average length of the randomly picked working paths (if any) for each pair yielded by DPillarMP.

Fig. 9 plots the average path length vs. the server failure ratio. We observe that although there are failures, the path length is still proportional to the number of server columns and it slightly decreases as there are more failed servers. This is because the lengths of the node-disjoint paths for a source–destination pair can be different. As we randomly

[5] The 16-port Gbit switch mentioned in Section 2.5 supports up to 12.2 k bytes jumbo frame.

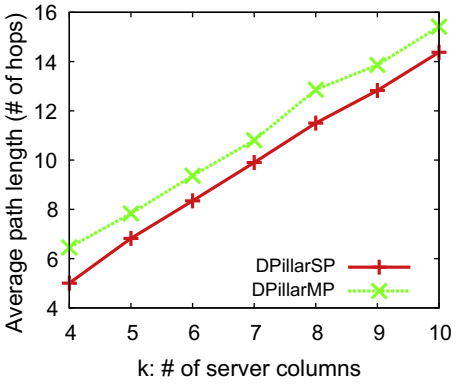

Fig. 8. Average path length for DPillarSP and DPillarMP.

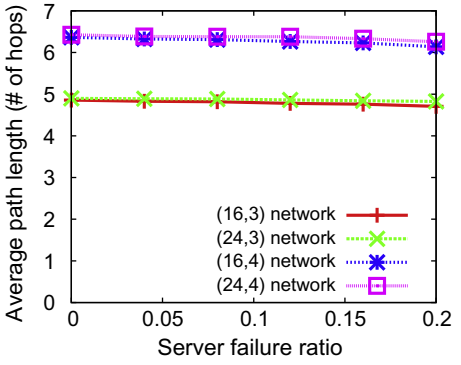

Fig. 9. Average path length vs. server failure ratio.

generate failed servers, a shorter path is less likely to fail as compared with a longer path, because the short path has fewer servers.

5.4. Fault tolerance

We also study the performance of DPillarMP in tolerating failures in the network. In each simulation instance, a certain number of servers or switches in a (16,3) DPillar network are randomly chosen as failed ones. We then randomly select 10,000 source–destination pairs from those servers without failures. For each source–destination pair, we compute how many paths among the $\frac{n}{2}$ clockwise disjoint paths yielded by DPillarMP are still available. The average number of available paths among all 10,000 source–destination pairs is plotted in Fig. 10. From Fig. 10, we see that as there are more server failures or switch failures, the average number of the available paths decreases. Switch failures have much larger impact than server failures, since a switch connects much more links than a server (n vs. 2).

We further study how the server and switch failures can impact reachability in DPillar. For a source–destination pair, if DPillarSP (or DPillarMP) cannot yield a working path, we say DPillarSP (or DPillarMP) has a *routing failure*. In each simulation instance, we randomly select

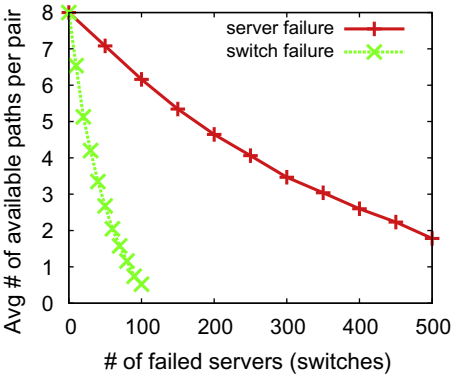

Fig. 10. Average number of available paths for one pair.

$M = 10{,}000$ source–destination pairs. If M^* of them have routing failures, we say the routing failure ratio of this scheme is $\frac{M^*}{M}$.

Figs. 11 and 12 plot the routing failure ratio of DPillarSP and DPillarMP, vs. the number of server failures and the switch failures, respectively. For both DPillarSP and DPillarMP, the routing failure ratio increases as there are more server failures or switch failures. However, the routing failure ratio of DPillarMP is much lower than that of DPillarSP in all cases. DPillarMP does not have any routing failures even when the number of server failures reaches 300 (20% of total servers), while DPillarSP has 45% routing failures at this point. When the number of server failures is as high as 500 (32% of total servers), DPillarMP has only about 0.1% routing failures. Even when the number of switch failures reaches 20 (10% of total switches), the routing failure ratio is about 1% only for DPillarMP.

5.5. Server forwarding load under typical traffic patterns

We study the forwarding load of servers under two typical traffic patterns, i.e., one-to-one communication pattern and all-to-all communication pattern. In the former pattern, we divide all N servers in a DPillar into $\frac{N}{2}$ sources and $\frac{N}{2}$ destinations. Each source server sends a flow to a randomly selected destination server. For the latter pattern, we randomly select 1% servers from a DPillar network

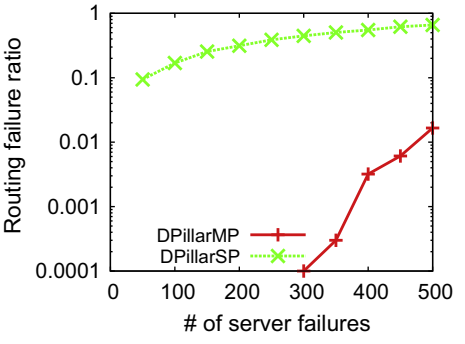

Fig. 11. Routing failure ratio vs. number of server failures.

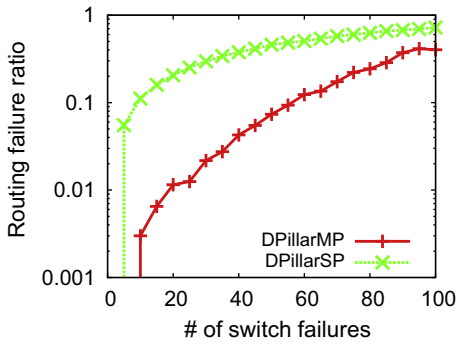

Fig. 12. Routing failure ratio vs. number of switch failures.

and let each of those $\frac{N}{100}$ servers send $(\frac{N}{100}-1)$ flows to the other $(\frac{N}{100}-1)$ servers.

In our experiments, we use the number of flows forwarded by a server as the measure for its traffic forwarding load. The distribution of all servers' traffic forwarding load is plotted in Fig. 13. We can see in both networks we tested (a (16,3) network and a (24,3) network), most servers forward a small number of flows. For example, in one-to-one communication pattern around 95% of servers in both (16,3) network and (24,3) network forward no more than

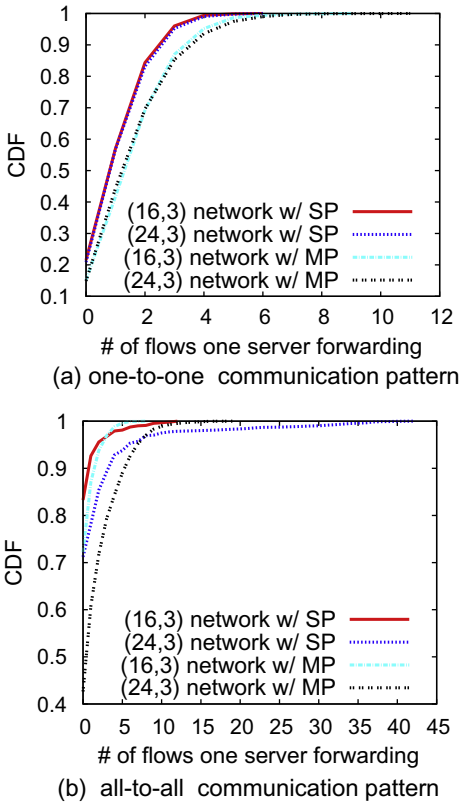

Fig. 13. Server traffic forwarding load in typical traffic patterns. Here "w/SP" means with DPillarSP routing scheme; "w/MP" means with DPillarMP routing scheme. In DPillarMP routing scheme, the source server randomly selects one route among all routes.

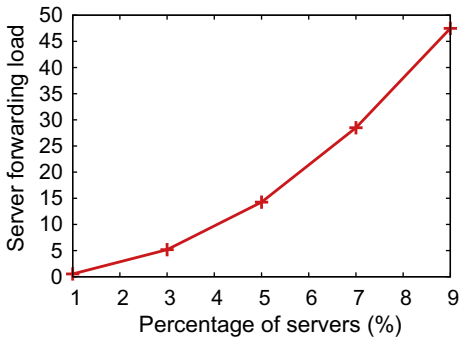

Fig. 14. Average server forwarding load in all-to-all communication pattern when the percentage of participating servers varies. The network is (16,3) DPillar network. A source server randomly selects a path among all paths to a destination.

4 flows using DPillarMP. The most loaded servers forward no more than 11 flows. Although the total number of flows in (24,3) network is much more than the total number of flows in (16,3) network, the server forwarding load does not have noticeable increase in one-to-one communication pattern. In all-to-all communication pattern, servers in (24,3) network need to forward more flows as compared to servers in (16,3) network, because there are more flows in (24,3) network as we select 1% of the servers to send out traffic.

To study how the server forwarding load changes when more servers participate in all-to-all communication, we simulate DPillarMP in a (16,3) DPillar where the percentage of servers participating in all-to-all communication ranges from 1% to 9%. We compute the average number of flows forwarded by servers in the DPillar network and plot the results in Fig. 14. As expected, when more servers participate in all-to-all communication, each server needs to forward more flows. The forwarding load of each server does not increase faster than the number of flows in the network. For example, when 3% of the servers participate in all-to-all communication, each server forwards 5 flows on average; when 9% of the servers participate in all-to-all communication, the total number of flows increases roughly $(\frac{0.09}{0.03})^2 = 9$ times, and the average number of flows forwarded by each server is 45, which increases 9 times as well.

6. Background and related work

6.1. Interconnection networks

Server interconnection network has long been an active research topic. Two categories of interconnection networks are broadly studied. The first one has a clear boundary between network and end hosts, where multiple levels of switches are connected into a switching fabric, and servers are attached as "leaves" of the switching fabric [4]. Servers are pure end-hosts, which perform computation and storage tasks only. One network interface is enough for each server to be connected with other servers.

In the second category of interconnection network, servers are not only computation/storage workstations

but also intermediate nodes relaying traffic for other servers. Classic interconnection topologies include full mesh, hypercube, butterfly, de Bruijn, etc. [19,24,10]. Compared to connecting servers by a switching fabric, using servers as relay nodes is often believed to be more flexible in building the interconnection network, because servers are much easier to program than switching devices.

6.2. Related work in data center networks

A thread of recent research activities on data center networks have proposed several interconnection architectures. The Monsoon network presented in [14] uses a hierarchical tree-structure switching fabric with two levels of switches, the top-of-rack switches and the core switches. The fat-tree network presented in [5] also uses a switching fabric to connect servers. The switching fabric of fat-tree network is built from identical switches, so there is no need to use expensive high-speed core switches. The switches used in fat-tree should have layer-3 switching capability and need to be slightly upgraded in order to make full use of its underlying topology. A second fat-tree based data center network structure, the PortLand network, is proposed in [22]. By using hierarchical pseudo MAC addresses, switches in a PortLand network can forward traffic as layer-2 packets. Removing layer-3 switching capability from switches can significantly reduce the cost of building large scale data center networks.

DCell [17] is a server-centric network where a higher level DCell network is recursively constructed from lower level DCell networks, and the number of accommodated servers grows double exponentially as the level increases. As the number of levels in a DCell network increases, servers need to install more interfaces. The links in DCell network are not evenly loaded. Those links connecting lower level DCells are usually more loaded than the links connecting higher level DCells. The FiConn network proposed in [20] uses similar recursive construction technique as DCell, but requires only two network interfaces in each server. FiConn also has the unevenly loaded link issue. BCube [16] is another server-centric network, whose topology is closely related to the Hypercube network [19]. BCube has rich connections to support bandwidth demanding applications running in data centers. But servers in a BCube network need to install more network interfaces in order to scale the network size to accommodate more servers. Table 2 summarizes the key features of different data center interconnection networks.

7. Conclusion

This paper presents DPillar, a data center network architecture built from commodity hardware. DPillar is a server-centric architecture and the networking intelligence is placed in servers. Switches in DPillar are layer-2 plug-and-play devices, which are cost efficient and widely available. DPillar can easily scale to a huge number of servers without imposing any additional requirements to servers, such as installing more network interfaces. The topology of DPillar is symmetric and a DPillar network provides rich connections between servers. We have designed efficient routing schemes for DPillar. Prototyping implementation and simulation studies show that our routing schemes are lightweight, high-performance, and efficient in bypassing failures in the network.

Acknowledgments

The authors are grateful to the editor, Dr. Luigi Iannone, and the anonymous reviewers for many insightful comments and constructive suggestions. This work is partially supported by NSF Grants CNS-0917078 and CNS-0831940.

Appendix A. Proof of Proposition 2.3

Our proof is inspired by previous work [8] in studying the bisection width of butterfly networks. Clearly, if we cut a (n,k) DPillar network horizontally, i.e., each server column is cut into halves, we can always cut a DPillar network into a top half and a bottom half by cutting the connections among H_{k-1}, S_{k-1}, and H_0. For example, we can divide the DPillar network shown in Fig. 2 into top and bottom halves by cutting the links cross a virtual line between row (13) and row (20). Because each switch in S_{k-1} has $\frac{n}{2}$ links crossing the virtual cutting line, the total number of links crossing the virtual cutting line is $\left(\frac{n}{2}\right)^{k-1} \times \left(\frac{n}{2}\right) = \left(\frac{n}{2}\right)^{k}$. Hence, we have an upper bound, $\left(\frac{n}{2}\right)^{k}$, for the bisection width of a (n,k) DPillar network.

Table 2
Comparison between different data center interconnection networks. Parameter n is the number of ports in each switch; N is the total number of servers; U_s is the unit price of a n-port switch. For DCell, FiConn, and BCube, l is depth of recursion in building the network. For DPillar, k is the number of server columns.

	DCell	FiConn	BCube	FatTree	DPillar
Server degree	$l+1$	2	$l+1$	1	2
Bisection width	$\frac{N}{4\log_n^N}$	$\frac{N}{4\times 2^l}$	$\frac{N}{2}$	$\frac{N}{2}$	$\frac{N}{k}$
Number of servers	$(n+1)^{2^l}$	$2^{l+2}\left(\frac{n}{4}\right)^{2^l}$	n^{l+1}	$\frac{n^3}{4}$	$k\left(\frac{n}{2}\right)^k$
Number of switches	$\frac{N}{n}$	$\frac{N}{n}$	$(l+1)\frac{N}{n}$	$6\frac{N}{n}$	$k\frac{N}{n}$
Cost of connecting One server[†]	$\frac{U_s}{n}$	$\frac{U_s}{n}$	$(l+1)\frac{U_s}{n}$	$5\frac{U_s}{n}$	$2\frac{U_s}{n}$
Switch upgrade	No	No	No	Yes	No
Traffic balance	No	No	Yes	Yes	Yes
Disjoint paths	$l+1$	1	$l+1$	1	$\frac{n}{2}$

[†] Not including the cost of NICs and cables.

Next we find the lower bound of the bisection width. Let G be the number of servers in each column of a (n,k) DPillar network. We consider bisecting the $2G$ servers in server columns S_0 and S_{k-1} by embedding a complete bipartite graph $K_{G,\ G}$ into a (n,k) DPillar network so that the left side nodes and right side nodes of $K_{G,G}$ are mapped to the servers in S_0 and servers in S_{k-1} of the (n,k) DPillar network, respectively. If each of the G servers in S_0 has a path to every server in S_{k-1}, because DPillar network is symmetry, there are at most $G/2$ paths use the same server-to-switch link. Also because the bisection width of a complete bipartite graph $K_{G,G}$ is $G^2/2$, the size of the cut that bisects the $2G$ servers in S_0 and S_{k-1} should be at least G. Now we consider a minimal cut C that bisects all servers in a (n,k) DPillar network into Set_1 and Set_2. If there exist two neighboring server columns, e.g., S_i and $S_{(i+1)\%k}$, where the $2G$ servers are bisected by cut C, we know that the size of cut C is at least G. Otherwise, we find two neighboring server columns S_j and $S_{(j+1)\%k}$ so that among the $2G$ servers in those two server columns, more servers are in Set_1 than in Set_2. Then we move some servers (among those $2G$ servers in server columns S_j and $S_{(j+1)\%k}$) from Set_1 to Set_2 so that half of those $2G$ servers are in Set_1. Note that moving the servers from Set_1 to Set_2 does not increase the size of cut C. We already know that bisecting the $2G$ servers in S_j and $S_{(j+1)\%k}$ requires cutting at least G links. Hence, the size of cut C has lower bound $G = \left(\frac{n}{2}\right)^k$.

Because both the upper bound and the lower bound is $\left(\frac{n}{2}\right)^k$, the bisection width of a (n,k) DPillar network is $\left(\frac{n}{2}\right)^k$.

Appendix B. Proof of Theorem 4.2

Without loss of generality, the addresses of s's clockwise neighbors can be represented as $(s+1, v_x^{k-1} \dots v_{x_i}^s \dots v_x^{d-1} \dots v_x^0)$, where $i \in [0, \frac{n}{2}-1]$ and $v_{x_i}^s = i$. The addresses of d's counter-clockwise neighbors are denoted as $\left(d-1, v_y^{k-1} \dots v_y^s \dots v_{y_j}^{d-1} \dots v_y^0\right)$, where $j \in [0, \frac{n}{2}-1]$ and $v_{y_j}^{d-1} = j$. Next we divide the theorem into two cases and prove them one by one.

Case 1: $s = d-1$: We have $v_{x_i}^s = v_{x_i}^{d-1} = i$ and $v_{y_i}^s = v_{y_i}^{d-1} = i$, and thus Algorithm 2 will pair server $(s+1, v_x^{k-1} \dots v_{x_i}^s \dots v_x^0)$ with server $\left(d-1, v_y^{k-1} \dots v_{y_i}^s \dots v_y^0\right)$ for $i \in [0, \frac{n}{2}-1]$. Let P be the DPillarSP path between server $\left(s+1, v_x^{k-1} \dots v_{x_i}^s \dots v_x^0\right)$ and server $(d-1, v_y^{k-1} \dots v_{y_i}^s \dots v_y^0)$, and P' be the DPillarSP path between server $\left(s+1, v_x^{k-1} \dots v_{x_j}^s \dots v_x^0\right)$ and server $\left(d-1, v_y^{k-1} \dots v_{y_j}^s \dots v_y^0\right)$, where $i \neq j$. It is easy to show both P and P' have length $k-1$. Furthermore, all servers on P have the sth symbol to be $v_{x_i}^s$ and all servers on P' have sth symbol to be $v_{x_j}^s$. Because $v_{x_i}^s \neq v_{x_j}^s$, these two paths have no common servers. Moreover, for the servers on these two paths, s does not equate to the number of the server column or the number minuses one. By Proposition 2.1, we know servers of these two paths always connect to different switches when they are in the same server column. Therefore, P and P' have no common servers or switches.

Case 2: $s \neq d-1$: In this case, the $(d-1)$th symbol $\left(v_x^{d-1}\right)$ of the labels of s's clockwise neighbors is the same, and the sth symbol $\left(v_y^s\right)$ of the labels of d's counter-clockwise neighbors is the same as well. We let $a = v_x^{d-1}$ and $b = v_y^s$. Then, server $\left(s+1, v_x^{k-1} \dots v_{x_b}^s \dots v_x^{d-1} \dots v_x^0\right)$ and server $\left(d-1, v_y^{k-1} \dots v_y^s \dots v_{y_a}^{d-1} \dots v_y^0\right)$ have the same sth and $(d-1)$th symbols. Let P'' be the DPillarSP path from server $\left(s+1, v_x^{k-1} \dots v_{x_b}^s \dots v_x^{d-1} \dots v_x^0\right)$ to server $\left(d-1, v_y^{k-1} \dots v_y^s \dots v_{y_a}^{d-1} \dots v_y^0\right)$. Assume P be the DPillarSP path between server $(s+1, v_x^{k-1} \dots v_{x_e}^s \dots v_x^{d-1} \dots v_x^0)$ and server $(d-1, v_y^{k-1} \dots v_y^s \dots v_{y_g}^{d-1} \dots v_y^0)$, and P' be the DPillarSP path between server $\left(s+1, v_x^{k-1} \dots v_{x_f}^s \dots v_x^{d-1} \dots v_x^0\right)$ and server $\left(d-1, v_y^{k-1} \dots v_y^s \dots v_{y_h}^{d-1} \dots v_y^0\right)$, where $b \neq e \neq f$ and $a \neq g \neq h$. We have $v_{x_e}^s \neq v_y^s$, and $v_{x_f}^s \neq v_y^s$. According to Corollary 4.1, P and P' have no common servers or switches. We also have $v_x^{d-1} \neq v_{y_g}^{d-1}$, and $v_{x_f}^s \neq v_y^s$. According to Corollary 4.1, P'' and P have no common servers or switches either. Similarly, we can prove P'' and P' have no common servers or switches.

We have shown that all the DPillarSP paths between two servers of the server pairs constructed by Algorithm 2 have no common servers or switches. Therefore, those $\frac{n}{2}$ paths (each path via one server pair) between s and d are node-disjoint. The proof completes.

References

[1] Amazon elastic compute cloud. <http://aws.amazon.com/ec2/>.
[2] The click modular router project. <http://read.cs.ucla.edu/click/>.
[3] Microsoft windows azure platform. <http://www.microsoft.com/windowsazure>.
[4] Cisco data center infrastructure 2.5 design guide, December 2007. <http://www.cisco.com/application/pdf/en/us/guest/netsol/ns107/c649/ccmigration_09186a008073377d.pdf>.
[5] M. Al-Fares, A. Loukissas, A. Vahdat, A scalable, commodity data center network architecture, in: Proceedings of SIGCOMM' 08, 2008.
[6] R. Banner, A. Orda, Multipath routing algorithms for congestion minimization, IEEE/ACM Trans. Netw. 15 (2007) 413–424.
[7] T. Benson, A. Akella, and D. A. Maltz, Network traffic characteristics of data centers in the wild, in: IMC '10: Proceedings of Internet Measurement Conference, 2010.
[8] C. Bornstein, A. Litman, B. Maggs, R. Sitaraman, T. Yatzkar, On the bisection width and expansion of butterfly networks, in: Parallel Processing Symposium, 1998. IPPS/SPDP 1998. Proceedings of the First Merged International Symposium on Parallel and Distributed Processing, 1998, pp. 144–150.
[9] F. Chang, J. Dean, S. Ghemawat, W.C. Hsieh, D.A. Wallach, M. Burrows, T. Chandra, A. Fikes, R.E. Gruber. Bigtable: a distributed storage system for structured data, in: OSDI '06: Proceedings of the 7th Symposium on Operating Systems Design and Implementation, Berkeley, CA, USA, 2006. USENIX Association, pp. 205–218.
[10] W.J. Dally, B.P. Towles, Principles and Practices of Interconnection Networks, Morgan Kaufman, 2004.
[11] J. Dean, S. Ghemawat, MapReduce: simplified data processing on large clusters, in: OSDI '04: Proceedings of the 5th Symposium on Operating Systems Design and Implementation, 2004, pp. 137–150.
[12] D. Ganesan, R. Govindan, S. Shenker, D. Estrin, Highly-resilient, energy-efficient multipath routing in wireless sensor networks, SIGMOBILE Mob. Comput. Commun. Rev. 5 (2001) 11–25.

[13] S. Ghemawat, H. Gobioff, S.-T. Leung, The google file system, SOSP '03: Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles, ACM Press, New York, NY, USA, 2003, pp. 29–43.
[14] A. Greenberg, P. Lahiri, D.A. Maltz, P. Patel, S. Sengupta, Towards a next generation data center architecture: scalability and commoditization, in: PRESTO '08: Proceedings of the ACM Workshop on Programmable Routers for Extensible Services of Tomorrow, New York, NY, USA, 2008. ACM, pp. 57–62.
[15] A.G. Greenberg, J.R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D.A. Maltz, P. Patel, S. Sengupta, Vl2: a scalable and flexible data center network, in: ACM SIGCOMM Conference, 2009, pp. 51–62.
[16] C. Guo, G. Lu, D. Li, H. Wu, X. Zhang, Y. Shi, T. Chen, Y. Zhang, S. Lu, BCube: a high performance, server-centric network architecture for modular data centers, in: Proceedings of SIGCOMM' 09, 2009.
[17] C. Guo, H. Wu, K. Tan, L. Shi, Y. Zhang, S. Lu, Dcell: a scalable and fault-tolerant network structure for data centers, in: Proceedings of SIGCOMM' 08, 2008.
[18] J. He, J. Rexford, Toward internet-wide multipath routing, Network, IEEE 22 (2) (2008) 16–21.
[19] F.T. Leighton, Introduction to Parallel Algorithms and Architectures: Arrays. Trees. Hypercubes, Morgan Kaufman, 1992.
[20] D. Li, C. Guo, H. Wu, K. Tan, Y. Zhang, S. Lu, FiConn: Using backup port for server interconnection in data centers, in: Proceedings of INFOCOM' 09, 2009.
[21] Y. Liao, D. Yin, L. Gao, Dpillar: Scalable dual-port server interconnection for data center networks, in: Proceedings of ICCCN 2010: the 19th International Conference on Computer Communications and Networks.
[22] R.N. Mysore, A. Pamboris, N. Farrington, N. Huang, P. Miri, S. Radhakrishnan, V. Subramanya, A. Vahdat, Portland: a scalable fault-tolerant layer 2 data center network fabric, in: Proceedings of SIGCOMM' 09, 2009.
[23] L. Rabbe, Powering the Yahoo! network, November 2006. <http://ycorpblog.com/2006/11/27/powering-the-yahoo-network>.
[24] M.R. Samatham, D.K. Pradhan, The de bruijn multiprocessor network: a versatile parallel processing and sorting network for VLSI, IEEE Trans. Comput. 38 (4) (1989) 567–581.

Yong Liao graduated with a BS degree in 2001 from University of Science and Technology of China. In 2004, he received his MS degree from the Graduate School of Chinese Academy of Sciences. Since fall 2004, he has been working as research assistant in University of Massachusetts at Amherst, where now he is a PhD candidate in the Electrical and Computer Engineering department. His current research interests include inter-domain routing, data center network, and network virtualization.

Jiangtao Yin received a BE degree from Beijing Institute of Technology, China, in 2006, and his ME degree from Beijing University of Posts and Telecommunications, China, in 2009. Since fall 2009, he has been working as research assistant in University of Massachusetts at Amherst, where he is currently pursuing the PhD degree in electrical and computer engineering department. His current research interests include data center network and data-intensive computing.

Dong Yin is a PhD student at Northwestern Polytechnical University (NWPU), Xi'an, Shanxi, China, whose major is Control Theory and Engineering. He received the bachelor degree in Information Security and the master degree in Automatic Control and Engineering from NWPU in 2004 and 2007, respectively. From October 2009 to October 2010, he had been a visiting student in department of Electrical and Computer Engineering at University of Massachusetts at Amherst, supported by China state scholarship fund. His research interests include Network virtualization and information security.

Lixin Gao is a professor of Electrical and Computer Engineering at the University of Massachusetts at Amherst. She received her PhD degree in computer science from the University of Massachusetts at Amherst in 1996. Her research interests include multimedia networking, Internet routing and security. Between May 1999 and January 2000, she was a visiting researcher at AT&T Research Labs and DIMACS. She is an Alfred P. Sloan Fellow, an IEEE Fellow and received an NSF CAREER Award in 1999. She has served on number of technical program committees including SIGCOMM2006, SIGCOMM2004, SIGMETRICS2003, and INFOCOM2004, and is on the Editorial Board of IEEE Transactions on Networking.