DPillar: Scalable Dual-Port Server Interconnection for Data Center Networks

Yong Liao
ECE Department
University of Massachusetts
Amherst, MA 01003, USA

Dong Yin
Automation Department
Northwestern Polytech University
Xi'an, ShanXi 710072, China

Lixin Gao
ECE Department
University of Massachusetts
Amherst, MA 01003, USA

***Abstract*—Data centers are becoming increasingly important infrastructures for many essential applications such as data intensive computing and large-scale network services. A typical future data center can consist of up to hundreds of thousands of servers. Yet, the conventional data center networks are not able to keep up with the network bandwidth requirement for efficiently connecting that huge number of servers in a cost-efficient manner. In this paper, we present DPillar, a highly scalable data center interconnection architecture, which uses only low-end off-the-shelf commodity PC servers and switches. DPillar has minimal requirements for the equipment. The switches are low-cost plug-and-play layer-2 devices and servers are dual-port commodity PCs. The salient feature of DPillar is that it expands to any number of servers without requiring to physically upgrade the existing servers. We present simple yet efficient routing schemes for DPillar and evaluate the routing performance via simulations.**

I. Introduction

Data centers with a cluster of commodity servers become common places for data storage, data analysis, and running large-scale network services [1,2]. Such a data center infrastructure is driven by the demand of petabyte of data storage and high computation power required for processing the data. It is commonly projected that the demand for data storage and processing will grow rapidly as more data are available for applications such as web searching, medical image processing, social network mining, and scientific computing. To meet the demand of the growth, one of the essential requirements for data center infrastructure is that it must scale to hundreds of thousands or millions of servers.

While inexpensive commodity PCs make it possible to expand a data center to millions of servers, interconnecting these servers in a scalable and cost-efficient fashion can be challenging. With a data center of increasing server number and storage size, the communication bandwidth has to scale more than linearly (or squarely) to meet the bandwidth demand of frequent data accessing and shuffling in distributed data processing and storage. In order to keep the interconnection cost low, one natural choice for interconnecting these servers is to leverage commodity hardware such as inexpensive Ethernet switches and the existing network cards in commodity servers. So far, there are two approaches for interconnecting servers with commodity switches. The first approach is *switch centric* where the switch functionality is extended to accommodate the need of the interconnection, while requiring no modification to the servers [3]–[5]. The second approach is *server centric* where each server acts as both data processing/storage and data relay node while requiring no change to the switches [6]–[8].

In this paper, we take a server centric approach and propose a server interconnection structure called DPillar. Each server in a DPillar network is a computation workstation as well as an intermediate node relaying data between other servers. A server centric design offers several attractive advantages as compared to a switch centric design. First, shifting the networking functionalities from a separate switching fabric to the servers provides much higher degree of programming capability, which facilitates the design of intelligent routing schemes. Second, a server centric design is much more cost-efficient because it uses only low-end layer-2 dummy switches.

The network structure of DPillar resembles existing multi-stage interconnection schemes [9,10]. However, DPillar offers several practical advantages in building large size and scalable data centers. DPillar aims to leverage plug-and-play commodity Ethernet switches with only layer-2 switching capability. Ethernet switches with moderate number of ports (e.g., 24 or 48 ports) and with the ability to switch at line speed are widely available and relatively inexpensive. Layer-2 Ethernet switches also have the advantage of requiring minimal configuration effort as they are basically plug-and-play devices. Further, DPillar requires only two network interfaces for each server. As most off-the-shelf PC servers and servers in existing data centers already have two high-speed Gbit Ethernet ports, one primary port and one backup port, there is no need to physically upgrade the servers when using new commodity servers or reusing servers in existing data centers to build a DPillar data center. When expanding an existing DPillar network with additional servers, it is not required to upgrade existing servers in the data center either (note that although upgrading servers like installing additional NICs is cheap in terms of the equipment cost, but the time and human power needed to upgrade tens or hundreds of thousands servers are very expensive). Therefore, DPillar can scale to any number of servers with minimal deployment overhead.

Despite the fact that each server has only two network interfaces, DPillar offers rich connections between servers and the aggregate bandwidth can facilitate data-intensive applications. A DPillar network structure is totally symmetric, so that it removes any network bottleneck at the architecture level. We have designed a simple yet highly efficient routing scheme for DPillar network. One salient feature of the proposed routing scheme is that it eliminates the need of doing table lookup in packet forwarding. Our prototyping implementation using commodity PC shows that the PC servers can perform data forwarding in line speed without consuming significant resources at the servers [11]. Therefore, such an interconnection structure is feasible for the server centric approach. Furthermore, we propose routing schemes that can efficiently handle a wide range of failures in DPillar network.

The rest of this paper is organized as follows. Section II presents the topological structure of the DPillar network. Section III and section IV are devoted to the discussion of routing in DPillar network. Section V presents the performance evaluation of DPillar. Section VI concludes this paper.

II. Interconnection of DPillar

In this section, we first present the interconnection structure of DPillar. Then we discuss the topological properties of DPillar and the cost of building such a network. We also discuss the differences between DPillar and a closely related multi-stage interconnection network.

A. DPillar Network Structure

A DPillar network is built of two kinds of devices, dual-port servers and n-port switches. The servers are arranged into k columns and so are the switches. Hence, a DPillar network is uniquely defined by two parameters, n and k. We call such a DPillar network an (n, k) DPillar network.

We use $H_0 \sim H_{k-1}$ to represent the k server columns and $S_0 \sim S_{k-1}$ to represent the k switch columns. The k server columns and k switch columns are alternately placed along a cycle, as shown in Fig. 1. Visually, it looks like the $2k$ columns of servers and switches are attached to the cylindrical surface of a pillar. Using its two ports, a server in each server column is connected to two switches in its two neighboring switch columns. In other words, for a server in column H_i, one of its ports is connected to a switch in column S_i and the other port is connected to a switch in column $S_{(i+k-1)\%k}$. For a switch in column S_i, half of its n ports are connected to $n/2$ servers in H_i and the other half are connected to $n/2$ servers in $H_{(i+1)\%k}$. For easy description, in the rest 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.

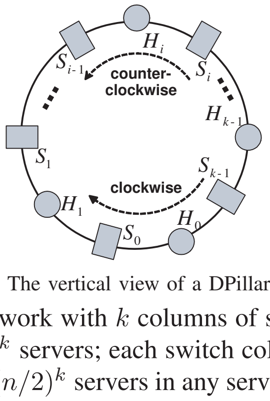

Fig. 1. The vertical view of a DPillar network.

In a DPillar network with k columns of servers, each server column has $(n/2)^k$ servers; each switch column has $(n/2)^{k-1}$ switches. For the $(n/2)^k$ servers in any server column H_i, each of them is assigned with a unique k-symbol label $(\nu^{k-1}...\nu^0)$, where $\nu^i \in [0, n/2-1]$ $(0 \le i \le k-1)$. Under this naming scheme, one server in DPillar can be uniquely identified as $(C, \nu^{k-1}...\nu^0)$, which means a server with label $(\nu^{k-1}...\nu^0)$ in server column H_C. *We call* $(C, \nu^{k-1}...\nu^0)$ *the ID or the address of the server.*

Given the IDs of the servers in a DPillar network, the interconnection between the servers and the switches is as follows. For all the $2(n/2)^k$ servers in any server column H_C and its clockwise neighboring server column $H_{(C+1)\%k}$, they can be divided into $(n/2)^{k-1}$ groups, with each group having n servers. The labels of the n servers in the same group have the following property. That is, their labels are the same if the Cth symbol (i.e., symbol ν^C) is removed. It is easy to see that among the n servers within the same group, half of them are from H_C and the other half are from $H_{(C+1)\%k}$. The n servers in the same group are connected to the same switch in switch column S_C. In other words, given any label $(\nu^{k-1}...\nu^C...\nu^0)$, there are $n/2$ servers in H_C whose labels are $(\nu^{k-1}...\nu^C_*...\nu^0)$ where $0 \le \nu^C_* \le n/2-1$; there are $n/2$ such servers in $H_{(C+1)\%k}$ too. Those n servers are connected to the same n-port switch in S_C.

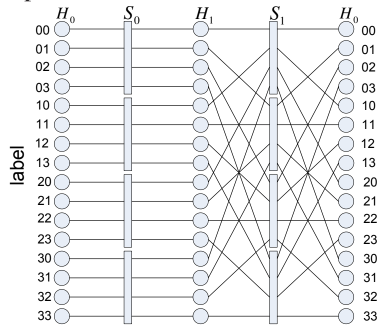

Fig. 2. Two-dimension view of a DPillar network.

Fig. 2 shows an $(8, 2)$ DPillar network (note that we duplicate server column H_0). Each server column has $(\frac{8}{2})^2 = 16$ servers. The label of each server has two symbols. The first row of servers in Fig. 2 have label (00) and the last row of servers have label (33). If we select a label $(\nu^1\nu^0)$=(00), there are four servers in H_1 whose labels are (ν^1_*0) with $0 \le \nu^1_* \le 3$, i.e., (00), (10), (20), and (30). There are four servers in H_0 whose labels are (00), (10), (20), and (30) too. Those eight servers are connected to the same switch in switch column S_1.

B. DPillar Topological Properties

After presenting the interconnection of DPillar, we proceed to study the basic topological properties of DPillar.

1) Number of servers: Since each server column has $(n/2)^k$ servers, there are $k(n/2)^k$ servers in an (n, k) DPillar network. Let N represent the total number of servers in an (n, k) DPillar network, we have $N = k(n/2)^k$.

Considering that 48-port Gbit Ethernet switches are widely available now and relatively inexpensive, a $(48, 3)$ DPillar network has $41,472$ servers. The number of servers will be about 1.3 million for a $(48, 4)$ DPillar network. If we build a $(48, 5)$ DPillar network, it has about 40 million servers.

2) Number of switches: As we have mentioned, each switch column has $(n/2)^{k-1}$ switches in an (n, k) DPillar network. As there are k switch columns, the total number of switches is $k(n/2)^{k-1}$. Actually, there is an explanation why there are $(n/2)^{k-1}$ switches in each switch column. If we change all other symbols in label $(\nu^{k-1}...\nu^{C}...\nu^{0})$ except symbol ν^{C}, there are $(n/2)^{k-1}$ different combinations. Each of those $(n/2)^{k-1}$ combinations requires one switch to connect n servers whose labels are $(\nu^{k-1}...\nu_{*}^{C}...\nu^{0})$ where $0 \leq \nu_{*}^{C} \leq n/2-1$. Therefore, the number of switches in each switch column is $(n/2)^{k-1}$ and the total number of switches in an (n,k) DPillar network is $k(n/2)^{k-1}$.

3) Bisection width: Bisection width is an important criterion to quantify the performance of an inter-connection network. 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 the network can sustain more communications between nodes, which is important for running communication intensive applications like MapReduce [12]. The bisection width of an (n,k) DPillar network is $(n/2)^{k}$, as stated in Proposition II.1. The proof is presented in [11].

Proposition II.1 *The bisection width of an (n,k) DPillar network is close to $(n/2)^{k}$.*

C. Building Cost

DPillar network is cost-efficient as it uses inexpensive commodity hardware. Here we provide some "example budgets" of building DPillar networks. We ignore the cost of servers and focus on the networking devices (switches and Ethernet cables). As most off-the-shelf servers already integrate dual-port interfaces, there is no need to invest on NICs.

TABLE I shows the total cost and the per-server cost of building DPillar networks with four servers columns, when different types of switches are used. The prices for the four types of switches are gotten from an online retailing store (www.newegg.com) and we assume each cable costs \$1. We expect their wholesale price would be even lower. In general, letting U_s be the unit price of a n-port switch and U_c be the unit price of an Ethernet cable, the average cost of connecting one server in the DPillar network is $2(U_s/n+U_c)$.

TABLE I
THE COST OF THE NETWORKING DEVICES WHEN USING FOUR TYPES OF SWITCHES TO BUILD DPILLAR NETWORKS WITH FOUR SERVER COLUMNS.

switch type	8-port	16-port	24-port	48-port
switch unit price	\$50	\$150	\$180	\$600
# of servers	1,024	16,384	82,944	1,327,104
networking cost	\$14,848	\$339,968	\$1,410,048	\$35,831,808
per-server cost	\$14.5	\$20.75	\$17	\$27

D. Contrasting DPillar and Existing Network Structure

Our DPillar network is closely related to *wrapped butterfly network* [9], which is one of the multi-stage interconnection networks extensively studied before. However, the servers in a wrapped butterfly network 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 investing 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.

The bisection width or an (n,k) DPillar network is equal to the number of servers in each server column. A wrapped butterfly network also has this property [9]. However, a wrapped butterfly network uses degree 4 nodes, but DPillar achieves the same bisection width using degree 2 nodes.

III. ROUTING IN DPILLAR NETWORK

Because of its symmetric structure, routing in DPillar network can be simple and efficient. In this section, we first describe the packet forwarding process in DPillar, based on which we design an efficient DPillar routing algorithm. We also briefly discuss the performance of the routing algorithm.

A. Two-Phase Packet Forwarding in DPillar

The packet forwarding process 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's label. In the second phase, the packet is sent from that intermediate server to the destination.

We consider a source server s sends a packet to 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 s and d, i.e., L_s=$(\nu_{s}^{k-1}...\nu_{s}^{0})$, L_d=$(\nu_{d}^{k-1}...\nu_{d}^{0})$, and $L_s \neq L_d$.

1) Phase one – helix phase: From server s in column H_{C_s}, the packet can be sent to a server s_1 in column $H_{(C_s+1)\%k}$. The labels of s and s_1 are the same except the C_sth symbol of s_1's label can be any number in $[0, n/2-1]$. If s_1 sends the packet to s_2 in column $H_{(C_s+2)\%k}$, the labels of s_1 and s_2's are the same except that the $((C_s+1)\%k)$th symbol of s_2's label can be any number in $[0, n/2-1]$. We see that forwarding a packet one hop can "change" one symbol in the label of the server receiving that packet. When a packet is always forwarded from one server column to its clockwise neighboring server column, the packet can reach a server with any given label in k hops. For example, in an (n,k) DPillar network, the trace of a packet forwarded from $(0,0...0)$ to $(k{-}1,1...1)$ is $(0,0...0) \rightarrow (1,0...1) \rightarrow (2,0...11) \rightarrow (k{-}1,1...1)$. As this path resembles a helix, we call this packet forwarding phase the *helix phase*.

Note that in the helix phase, we can 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. Some field in the packet header can be used to record the forwarding direction information of this packet. In DPillar routing, the *default direction* of forwarding a packet in the helix phase is the clockwise direction.

2) Phase two – ring phase: After the packet is forwarded to server d^{*}, whose label is the same as the label of destination 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 select the shorter one among those two paths in our DPillar routing. In other words, suppose server d^{*} is in column $H_{C_{d^{*}}}$,

d^* sends the packet to a server in column $H_{(C_{d^*}+1)\%k}$ if $(C_d + k - C_{d^*})\%k \le \lfloor \frac{k}{2} \rfloor$; otherwise server d^* sends the packet to a server in column $H_{(C_{d^*}+k-1)\%k}$. In either case, the label of the nexthop server should be L_d. As the trace of the packet forwarding in this phase is like a segment in a ring, we call it the *ring phase*.

B. Routing Algorithm

Algorithm 1 shows the pseudocode of the routing algorithm. This algorithm takes the address of the server running this algorithm (C_s, L_s), 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 default value of D should be 1. The output is the address of the nexthop server (C_u, L_u).

Algorithm 1: $SRoute(C_s, L_s, C_d, L_d, D)$

```
input  : $(C_s, L_s)$ is the address of the server running this
         algorithm. $(C_d, L_d)$ is the destination. $D$ is the
         forwarding direction recorded in the packet header
         (either 1 or $-1$). $L_s$=$(\nu_s^{k-1}...\nu_s^0)$, $L_d$=$(\nu_d^{k-1}...\nu_d^0)$.
output : Address of the nexthop server $(C_u, L_u)$.
/* $L_s$-$\nu_s^{C_s}$ means removing the $C_s$th symbol
   from label $L_s$ */
1  if $\{(C_d+k-C_s)\%k \le 1$ and $L_s$-$\nu_s^{C_s}$== $L_d$-$\nu_d^{C_s}\}$ or
   $\{(C_s+k-C_d)\%k \le 1$ and $L_d$-$\nu_d^{C_d}$==$L_s$-$\nu_s^{C_d}\}$ then
2      $C_u \leftarrow C_d$; $L_u \leftarrow L_d$;
3  else                          /* $s$ cannot directly reach $d$ */
4      if $L_s == L_d$ then                 /* the ring phase */
5          $L_u \leftarrow L_d$;
6          if $(C_d + k - C_s)\%k \le \lfloor \frac{k}{2} \rfloor$ then $C_u \leftarrow (C_s + 1)\%k$;
7          else $C_u \leftarrow (C_s + k - 1)\%k$;
8      else                                 /* the helix phase */
9          $L_u \leftarrow (\nu_s^{k-1}...\nu_d^s...\nu_s^0)$; $C_u \leftarrow (C_s + D + k)\%k$;
10 return $(C_u, L_u)$;
```

The DPillar routing presented in Algorithm 1 does not compute the shortest path between two servers. However, the paths computed by Algorithm 1 have bounded length. Here we study what is the length of the longest path in a DPillar network. In counting the path length, we treat the distance between two servers connecting to the same switch as *one* hop, because the switches are dummy layer-2 connection media.

From source server s, a packet needs to be forwarded at most k times to reach a server d^*, whose label is the same as destination server d's label. After that, from server d^*, the packet still needs to be forwarded at most $\lfloor k/2 \rfloor$ hops to reach server d. Therefore, in an (n, k) DPillar network, the longest path computed by Algorithm 1 is $k + \lfloor k/2 \rfloor$.

IV. Handling Failures in DPillar

The rich connections in DPillar facilitate the design of simple yet efficient fault-tolerant routing scheme. In this section, we present the design of a fault-tolerant routing algorithm which can bypass a wide range of failures in DPillar.

A. Discovering Failures

To bypass failures in DPillar network, the first step would be detecting the failures. Each server in DPillar runs a lightweight *Hello* protocol to report the reachability to other servers connected to the same switches. If a server a does not hear *Hello* message from server b for a certain period of time, a assumes b is not directly reachable. Servers should not forward any *Hello* messages.

After discovering the failures, the next question would be how to bypass those failures. In the following, we discuss how to bypass failures when the packet forwarding is in the *helix phase* and the *ring phase*, respectively.

B. Bypassing Failures in Helix Phase

In bypassing failures in the helix phase, our goal is to detour a packet so that it can still reach a server whose label is the same as the destination of that packet. After that, the packet is forwarded in the ring phase and section IV-C discusses how to bypass failures in the ring phase.

Our fault-tolerant routing scheme works in the following three steps in the helix phase: (1) if a nexthop server in the clockwise neighboring column is not reachable, we first try to send the packet to another directly reachable server in the clockwise neighboring column; (2) if none of the servers in the clockwise neighboring column is directly reachable, the packet takes a "u-turn" and be forwarded to a server in the counter-clockwise neighboring column; (3) after the u-turn, the packet should be forwarded along the counter-clockwise direction throughout the helix phase.

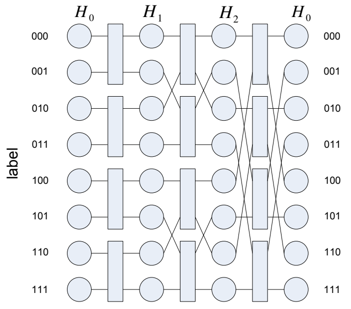

Fig. 3. A $(4, 3)$ DPillar network example.

1) Bypassing a failed clockwise server: Suppose server s sends a packet to destination d. According to Algorithm 1, s should send the packet to nexthop u. If s finds that u is not directly reachable any longer, s should try to bypass the failed server u. However, simply forwarding the packet to an alternative clockwise nexthop can cause loops in some scenarios. For example, in Fig. 3, when server (0,000) sends a packet to (2,001), Algorithm 1 tells us the ID of the nexthop should be (1,001). If server (1,001) has failed and server (0,000) tries to bypass that failure by sending the packet to server (1,000), the path taken by the packet will be (0,000) $\rightarrow$ (1,000) $\rightarrow$ (2,000) $\rightarrow$ (0,000), which is a loop.

The reason for the loop to occur is that, server s tries to bypass u and sends the packet to server v, but s is in the path (computed by Algorithm 1) from v to a server whose label is the same as destination d. To avoid this loop, we can let v send the packet to a clockwise neighbor w, so that neither s nor u is in the path from w to a server whose label is the same as destination d. For example, if (1,000) forwards the packet to (2,010), the path taken by the packet will be (0,000) $\rightarrow$ (1,000) $\rightarrow$ (2,010) $\rightarrow$ (0,010) $\rightarrow$ (1,011) $\rightarrow$ (2,001).

We use *tunneling* to ensure the packet is forwarded from s to w (they are two hops away) without using the failed server u. In the above example, server (0,000) encapsulates the packet with another header, whose destination is set to (2,010). Tunneling the packet between (0,000) and (2,010) makes sure the path of the encapsulated packet is (0,000) $\rightarrow$ (1,000) $\rightarrow$ (2,010). Therefore, the failed server (1,001) is bypassed.

We conclude the discussion into Proposition IV.1. Note that we omit the $\%k$ in presenting the indexes for clarity.

Proposition IV.1 *When sending a packet to destination* $(C_d, \nu_d^{k-1}...\nu_d^0)$*, server* $(C_s, \nu_s^{k-1}...\nu_s^0)$ *can bypass server* $(C_s+1, \nu_s^{k-1}...\nu_d^{C_s}...\nu_s^0)$ *by tunneling the packet to* $(C_s+2, \nu_s^{k-1}...\nu_w^{C_s+1}\nu_w^{C_s}...\nu_s^0)$*, if* $\nu_w^{C_s}{\neq}\nu_d^{C_s}$ *and* $\nu_w^{C_s+1}{\neq}\nu_s^{C_s+1}$*.*

Proof: Due to the symmetry of DPillar, without losing generality, we consider server s $(0, \nu_s^{k-1}...\nu_s^0)$, bypasses server $(1, \nu_s^{k-1}...\nu_s^1\nu_d^0)$ in reaching destination $(C_d, \nu_d^{k-1}...\nu_d^0)$. Server s tunnels the packet to server w, whose address is $(2, \nu_s^{k-1}...\nu_w^1\nu_w^0)$, where $\nu_w^0 \neq \nu_d^0$ and $\nu_w^1 \neq \nu_s^1$.

To tunnel the packet from s to w, the nexthop is $(1, \nu_s^{k-1}...\nu_s^1\nu_w^0)$. Because $\nu_w^0 \neq \nu_d^0$, that nexthop is not the failed server $(1, \nu_s^{k-1}...\nu_s^1\nu_d^0)$. After the packet is forwarded from w to server s_0 in column H_0, the address of s_0 is $(0, \nu_d^{k-1}...\nu_w^1\nu_w^0)$. As $\nu_w^1{\neq}\nu_s^1$, s_0 is not s. Server s_0 forwards the packet to server s_1 in column H_1 whose address is $(0, \nu_d^{k-1}...\nu_w^1\nu_d^0)$. Because $\nu_w^1{\neq}\nu_s^1$, s_1 is not the failed server $(1, \nu_s^{k-1}...\nu_s^1\nu_d^0)$. After s_1 forwards the packet, it reaches a server in H_2 whose label is the same as the destination. ■

2) Making a "packet u-turn": Proposition IV.1 assumes a server can always forward a packet to another server in its clockwise neighboring column in the helix phase. However, it is possible that server s cannot send the packet to any servers in its clockwise neighboring column, e.g., the server (0,000) in Fig. 3 has a link failure so it is disconnected from the top switch between H_0 and H_1, or the top switch between H_0 and H_1 has failed. In that case, server s needs to change the packet forwarding direction (make a "u-turn") to bypass the failure, i.e., if a packet cannot be forwarded to a server in the clockwise direction, it should be forwarded to a server in the counter-clockwise neighboring column. The forwarding direction information should be recorded in the packet header, so that other servers will forward this packet along the new direction. Similar to the rationale behind Proposition IV.1, we can prove the following Proposition IV.2. Again, the $\%k$ is omitted in presenting the indexes for clarity.

Proposition IV.2 *When sending a packet to destination* $(C_d, \nu_d^{k-1}...\nu_d^0)$*, if server* $(C_s, \nu_s^{k-1}...\nu_s^0)$ *cannot reach any server in* H_{C_s+1}*, it can bypass the failure by changing the packet forwarding direction and sending the packet to server* $(C_s{-}1,\ \nu_s^{k-1}...\nu_w^{C_s-1}...\nu_s^0)$*, where* $\nu_w^{C_s-1} \neq \nu_s^{C_s-1}$*.*

Proof: We consider that server s, whose ID is $(0, \nu_s^{k-1}...\nu_s^0)$, sends a packet to destination d $(C_d, \nu_d^{k-1}...\nu_d^0)$ but s cannot reach any server in H_1. To bypass the failure, server s changes the forwarding direction of the packet and sends it to nexthop w, whose ID is $(k-1, \nu_w^{k-1}...\nu_s^0)$, where $\nu_w^{k-1}{\neq}\nu_s^{k-1}$.

After the packet is forwarded along the counter-clockwise direction from w to some server s_1 in H_1, the address of s_1 is $(1, \nu_w^{k-1}...\nu_d^2\nu_d^1\nu_s^0)$. Because $\nu_w^{k-1}{\neq}\nu_s^{k-1}$, s_1 is not connected to server s by the same switch. After s_1 forwards the packet to some server s_0 in H_0, s_0's address is $(0, \nu_w^{k-1}...\nu_d^0)$. Because $\nu_w^{k-1}{\neq}\nu_s^{k-1}$, s_0 is not s so no loop occurs. After s_0 forwards the packet to server s_{k-1} in column H_{k-1}, the packet reaches a server whose label is the same as destination server d. ■

3) Bypassing a failed counter-clockwise server: Once the forwarding direction of a packet is changed, it is recorded in the packet header and all servers should forward the packet along the new direction. To avoid potential forwarding loops, if the forwarding direction of a packet was changed before, we should not change its forwarding direction again. Suppose server s selects a counter-clockwise neighbor u as the nexthop according to Algorithm 1, if u is unreachable and there are other reachable counter-clockwise neighbors, s can bypass the failure according to the following Proposition IV.3. For a packet whose direction was changed from clockwise to counter-clockwise and it cannot be forwarded along the new direction (no server in the counter-clockwise neighboring column is directly reachable), the packet should be *dropped*.

Proposition IV.3 *When sending a packet to destination* $(C_d, \nu_d^{k-1}...\nu_d^0)$*, server* $(C_s, \nu_s^{k-1}...\nu_s^0)$ *can bypass server* $(C_s{-}1, \nu_s^{k-1}...\nu_d^{C_s-1}...\nu_s^0)$ *by tunneling the packet to* $(C_s{-}2, \nu_s^{k-1}...\nu_w^{C_s-1}\nu_w^{C_s-2}...\nu_s^0)$*, if* $\nu_w^{C_s-1}{\neq}\nu_d^{C_s-1}$ *and* $\nu_w^{C_s-2}{\neq}\nu_s^{C_s-2}$*.*

The proof of Proposition IV.3 is similar to the proof of Proposition IV.1. We omit it here to save space.

C. Bypassing Failures in Ring Phase

Bypassing a failed server in the ring phase is relatively straightforward. As there are two ways to forward a packet to the destination in the ring phase, the clockwise direction or the counter-clockwise direction, if it cannot be forwarded along one direction, a packet should be forwarded along the other direction. To avoid forwarding loops, the forwarding direction of a packet should not be changed more than once. If a packet has changed its forwarding direction in the ring phase and it encounters a failed server again, we *drop* that packet.

D. Fault-tolerant Routing Algorithm

Based on the above discussions, we design the DPillar fault tolerant routing algorithm as shown in Algorithm 2. $FTRoute$ first calls $SRoute$ to compute a nexthop (C_w, L_w). Then $FTRoute$ tests whether (C_w, L_w) is reachable. If it is, $FTRoute$ does nothing; otherwise it tries to find a new nexthop according to the basic ideas specified in Proposition IV.1 $\sim$ IV.3. If the new nexthop is null, the packet should be dropped to prevent forwarding loops.

V. Evaluations

In this section, we study the macroscopic behavior of DPillar by simulating the packet routing and forwarding in a large scale DPillar network, using a simulation tool developed by ourselves. Please refer to [11] for the microscopic experiments in measuring the forwarding performance of DPillar.

Algorithm 2: $FTRoute(C_s, L_s, C_d, L_d, D)$

```
input  : $(C_s, L_s)$ is the address of the server running this
         algorithm. $(C_d, L_d)$ is the address of the destination.
         $D$ is the forwarding direction, either 1 or -1.
output : Address of the nexthop server $(C_w, L_w)$.
1  $(C_w, L_w)$ = SRoute($C_s, L_s, C_d, L_d, D$);
2  if $(C_w,L_w)$ is reachable then return $(C_w,L_w)$;
3  if $L_s$==$L_d$ then                          /* the ring phase */
4      if the packet direction was changed then
5          $C_w \leftarrow null$; $L_w \leftarrow null$;
6      else
7          if $(C_s + k - C_d)\%k \leq \lfloor \frac{k}{2} \rfloor$ then $C_w \leftarrow (C_s + 1)\%k$;
8          else $C_w \leftarrow (C_s + k - 1)\%k$;
9  else                                          /* the helix phase */
10     if the packet direction was changed then
11         apply Proposition IV.3 to get new nexthop;
12         if nexthop exists then set $C_w$ and $L_w$;
13         else $C_w \leftarrow null$; $L_w \leftarrow null$;
14     else
15         apply Proposition IV.1 to get new nexthop;
16         if nexthop exists then set $C_w$ and $L_w$;
17         else
18             change packet direction according to
               Proposition IV.2;
19             if nexthop exists then set $C_w$ and $L_w$;
20             else $C_w \leftarrow null$; $L_w \leftarrow null$;
21 return $(C_w, L_w)$;
```

Average path length: In our simulations, we focus on a scenario where the DPillar network is built from 12-port switches, i.e., $n = 12$, and the number of server columns k varies. For each network, we randomly select $100,000$ source-destination pairs and simulate packet routing and forwarding in the network. Fig. 4 plots the results of the average path lengths. As expected, the average path length is proportional to the number of server columns in a DPillar network. For all server column k, the simulated average path length is always about 20% shorter than the maximum path length $k + \lfloor k/2 \rfloor$.

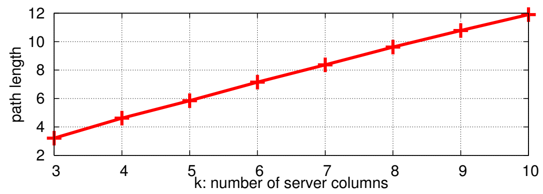

Fig. 4. Average path length in DPillar routing.

Average path length in failure-tolerant routing: We also study the fault-tolerant behavior of DPillar routing. In this simulation, we use a $(12, 4)$ DPillar network to conduct our experiments. In each simulation instance, we first randomly fail a certain number of servers. Then we randomly select 100,000 source-destination pairs and simulate the fault-tolerant routing scheme in Algorithm 2. We range the number of failed servers from 1 to 300 and plot the average path length as a function of the number of failed hosts.

Our simulation results show that the average path length increases as there are more failed servers in the DPillar network. However, no packet drop occurs in our simulations, even when 300 servers in the DPillar network have failed. As a $(12, 4)$ DPillar network has about 5,000 servers, 300 server failures mean 6% of the servers are failed.

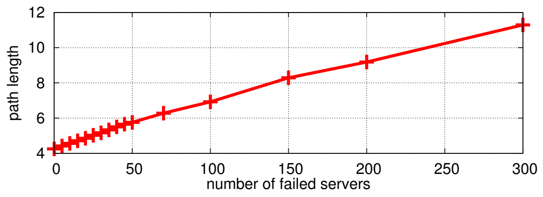

Fig. 5. Path length in fault-tolerant routing.

VI. Conclusion

In this paper, we present DPillar, a scalable data center network architecture which uses only commodity off-the-shelf hardware. DPillar can easily scale to huge number of servers without imposing any additional requirements to the devices, such as installing additional NICs in the servers. The topology of DPillar is totally symmetric and a DPillar network has balanced network capacity. DPillar is server centric and the networking intelligence is placed in the servers. The switches used in DPillar are merely dummy layer-2 devices connecting the servers. We designed simple yet efficient routing schemes for DPillar. Our experiment studies show that the routing schemes are high-performance and efficient in bypassing failures in the network.

Acknowledgment

This work is partially supported by NSF grants CNS-066618 and CNS-0626617. Dong Yin was a visiting student at UMass, supported by China State Scholarship Fund CSC-2008629080, when this work was performed.

References

[1] "Amazon elastic compute cloud," http://aws.amazon.com/ec2/.
[2] L. Rabbe, "Powering the Yahoo! network," Nov. 2006. [Online]. Available: http://ycorpblog.com/2006/11/27/powering-the-yahoo-network
[3] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," in *Proceedings of SIGCOMM' 08*, 2008.
[4] R. N. Mysore and et al., "Portland: A scalable fault-tolerant layer 2 data center network fabric," in *Proceedings of SIGCOMM' 09*, 2009.
[5] "Cisco data center infrastructure 2.5 design guide," Dec. 2007. [Online]. Available: www.cisco.com/application/pdf/en/us/guest/netsol/ns107/c649/ccmigration_09186a008073377d.pdf
[6] 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 *Proceedings of SIGCOMM' 08*, 2008.
[7] D. Li, C. Guo, H. Wu, K. Tan, Y. Zhang, and S. Lu, "FiConn: Using backup port for server interconnection in data centers," in *Proceedings of INFOCOM' 09*, 2009.
[8] C. Guo, G. Lu, D. Li, H. Wu, X. Zhang, Y. Shi, T. Chen, Y. Zhang, and S. Lu, "BCube: A high performance, server-centric network architecture for modular data centers," in *Proceedings of SIGCOMM' 09*, 2009.
[9] F. T. Leighton, *Introduction to Parallel Algorithms and Architectures: Arrays. Trees. Hypercubes.* Morgan Kaufmann, 1992.
[10] M. Jeng and H. Siegel, "A fault-tolerant multistage interconnection network for multiprocessor systems using dynamic redundancy," in *Proceedings of ICDCS' 86*, 1986.
[11] Y. Liao, D. Yin, and L. Gao, "Dpillar: Scalable dual-port server interconnection for data center networks," Tech. Rep., 2009. [Online]. Available: http://rio.ecs.umass.edu/mnilpub/papers/dpillar09tech.pdf
[12] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," in *Proceedings OSDI '04*, 2004.