Rethinking Fat-Tree Topology Design for Cloud Data Centers

Jarallah Alqahtani and Bechir Hamdaoui
Oregon State University, Corvallis, Oregon
{alqahtaj,hamdaoui}@eecs.oregonstate.edu

***Abstract*— Data center network (DCN) topologies have recently been the focus of many researchers due to their vital role in achieving high DCN performances in terms of scalability, power consumption, throughput, and traffic load balancing. This paper presents a comprehensive comparison between two most commonly used DCN topologies, Fat-Tree and BCube, with a focus on structure, addressing and routing, and proposes a new DCN topology that is better suited for nowadays data center networks. We show that our proposed topology, termed `Circulant Fat-Tree`, alleviates traffic congestion at the core switches, improves network latency, and increases robustness against switch and server failures when compared to traditional Fat-Tree DCN topologies.**

I. Introduction

Data centers nowadays consist of tens of thousands of servers and switches, all connected with high-speed communication links. The network topology design choice of these data centers is vital to ensuring scalability [1] and to improving data throughput [2] and power consumption [3]. As shown in Fig. 1, traditionally, data center network (DCN) topologies are built hierarchically, and are composed of core, aggregation and access (aka edge) layers [4]. Recent years have witnessed an exponential growth in DCN traffic, with a global data traffic projected to reach 20.6 ZB by 2021, about a three-fold increase from 2016 [5]. In addition, measurement-based studies [6], [7] show that the average traffic loads on core-layer switches are much higher than those on aggregation- and access-layer switches. With the booming of social media, DCNs have been growing even more rapidly, both in size and in numbers, making them more susceptible to device and link failures, which in turn results in frequent data routing failures and flow disruption.

In this paper, we study existing DCN topologies, and propose `Circulant Fat-Tree` topology, an improvement over the traditional Fat-Tree topology to better suit nowadays data center networks. Our proposed `Circulant Fat-Tree` outperforms traditional Fat-Tree by:

1) Alleviating traffic congestion at the core switches by balancing traffic loads across the different network switches.
2) Improving network latency by reducing the average path lengths between communicating servers.
3) Increasing robustness against network failures by providing more possible paths between servers.

The reminder of this paper is organized as follows. In Section II, we provide a taxonomy of various typologies that have been proposed in the literature, and detail the two commonly used DCN topologies, Fat-Tree and BCube. We introduce and present our proposed `Circulant Fat-Tree` topology in Section III and evaluate its performance in Section IV. We conclude the paper in Section V.

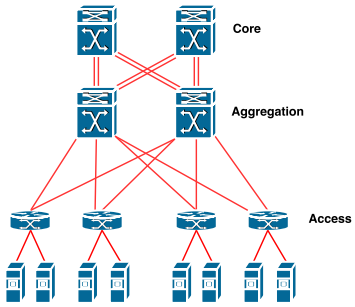

Fig. 1. Hierarchical Structure of DCNs.

II. DCN Topologies

In the past few years, there has been a tremendous research focus on developing new DCN topology designs with the aim of improving the DCN performance in terms of cost, power consumption, and traffic and resource management [1], [8]–[14]. Generally speaking, as shown in Fig. 2, these proposed topologies can be categorized based on device function type, device technique type, or architecture type. In the device function taxonomy, DCN topologies can be viewed as either switch-centric (e.g. Fat-Tree, Portland [10], and VL2 [11]), or server-centric (e.g. BCube [8] and DCell [9]). In switch-centric topologies, the forwarding function is enabled and implemented by switches only, whereas, in the server-centric topologies, servers are also enabled with such a forwarding capability and do play a role in data routing decisions. DCN topologies can also be categorized according to devices technique; that is, optical (e.g. OSA (Optical Switching Architecture) [15]), hybrid (e.g. C-Through [12] and Helios [13]), or electrical (e.g. Fat-Tree, BCube, DCell, and Jellyfish [14]). DCN topologies can also be categorized based their architecture and can be viewed as tree-based architecture (e.g. Fat-Tree, Portland [10], and VL2 [11]), recursive-based architecture (e.g. BCube [8] and DCell [9]), or random-based architecture (e.g. Jellyfish).

Next, we compare the two commonly used network topologies, Fat-Tree and BCube, by highlighting their pros and cons vis-a-vis of their structure, data addressing, and routing capability.

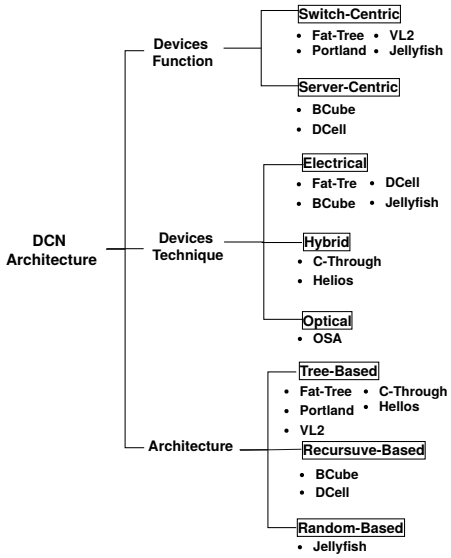

Fig. 2. Taxonomy of existing DCN topologies.

A. Fat-Tree

Fat-Tree as a DCN topology was first introduced by Al-Fares et. al [1] as an improvement to the tree topology. The main characteristic of Fat-Tree topology is that the number of links connecting a switch to its lower-layer switches is equal to the number of links connecting the switch to its parent switch. This feature helps in alleviating traffic congestion at the root level by considering multiple core switches as opposed to only one.

1) Structure: As shown in Fig. 3, there are three types of Fat-Tree switches, and depending on which layer the switch is placed in, a switch is either called a core, aggregation, or edge/access switch. A Fat-tree topology has k pods, with each pod having $k/2$ edge and $k/2$ aggregation switches. Also, each pod has a link to each core switch via aggregation switches. Each switch in the edge layer is connected to $k/2$ servers. For illustration, we consider $k = 4$ pods in Fig. 3 with 4 switches in each pod, two of which are aggregation switches and two are edge switches. Likewise, each aggregation switch has two ports connecting it to the edge switches and to the core switches. In general, a Fat-Tree with k pod has $k^3/4$ servers.

2) Addressing: Fat-Tree topology has special IP addressing. Switches in the edge and aggregation layers are assigned IP addresses of the form $10.pod.switch.1$, where $pod = 0, 1, \ldots, k-1$ denotes the pod number, and $switch = 0, 1, \ldots, k-1$ denotes the position of the switch in the pod, starting from left to right and bottom to top. Core switches are assigned addresses of the form $10.k.j.i$, where $j, i = 1, 2, ..., k/2$ denote the switch's coordinates, starting from top-left. Finally, servers are assigned addresses of the form $10.pod.switch.ID$, where $ID = 2, 3, ..., k/2 + 1$ denotes the position of the server in the subnet, starting from left to right. Fig. 3 shows how addresses are assigned to the Fat-Tree when $k = 4$.

3) Routing: Routing is performed into two steps, first based on primary prefix and then based on secondary suffix lookup. For each incoming packet, destination address prefix is checked in primary table first, and if longest prefix is matched, which means packets are being routed down to the servers, then the packet is forwarded to the specified port. Otherwise, the secondary level table is checked and the port entry with the longest suffix match is used to forward the packet, which means packets are being routed up towards core switches. At the core switches, packets are simply directed to the pod in which the destination is located.

B. BCube

BCube is a server-centric topology introduced mainly to support *modular data centers* [8]. One of its other main purposes is also to support all the traffic patterns (one-to-one, one-to-several, one-to-all, and all-to-all).

1) Structure: BCube is a recursive structure, with $BCube_0$ is constructed by n servers that are connected to an n-port switch, $BCube_1$ is constructed from n $BCube_0$ that are connected to n switches, and so on. In general, $BCube_k$ is constructed from n $BCube_{k-1}$ that are connected to n^k n-port switches. Each server in $BCube_k$ has $k+1$ ports, numbered from level-0 to level-k. Thus, $BCube_k$ has n^{k+1} servers and n^k switches. Fig. 4 shows the BCube structure with $n = 4$ and $k = 1$.

2) Addressing: Servers in $BCube_k$ are assigned addresses $a_k a_{k-1} a_0$ with $a_i \in \{0, 1, ..., n-1\}$. Switches, on the other hand, are assigned addresses in the form $< l, s_{k-1} s_{k-2} s_0 >$ where $l = \{0, 1, ..., k\}$ is the level of the switch. The i^{th} server in the j^{th} $BCube_0$ connects to the j^{th} port of the i^{th} level-1 switch. For example, server 12 is connected to switch $< 0, 1 >$ in level 0 and to switch $< 1, 2 >$ in level 1. Fig. 4 shows how addresses are assigned to BCube when $n = 4$ and $k = 1$.

3) Routing: BCube uses the Hamming distance technique, $h(A, B)$, to denote the distance between two servers, A and B, which corresponds to the number of different digits within their address arrays. When a packet is forwarded from source to destination, one digit is changed in each step. Thus, the path length is at most $k + 1$. There are $k + 1$ edge-disjoint paths between any two servers in a $BCube_k$ topology. These paths can be utilized for achieving high bandwidth in order to improve one-to-one, one-to-many, and all-to-all traffic performances. Fig. 4 shows an example of routing in $BCube_1$ when n=4.

Table I presents a summary of the Fat-Tree and BCube features that we presented.

III. The Proposed Topology Design: `Circulant Fat-Tree`

To overcome these aforementioned challenges, we propose an improved Fat-Tree topology, termed `Circulant Fat-Tree` in this paper. `Circulant Fat-Tree` improves DCN performances by reducing data traffic traversing congested links, thereby reducing network latency. In addition, `Circulant Fat-Tree` improves the robustness

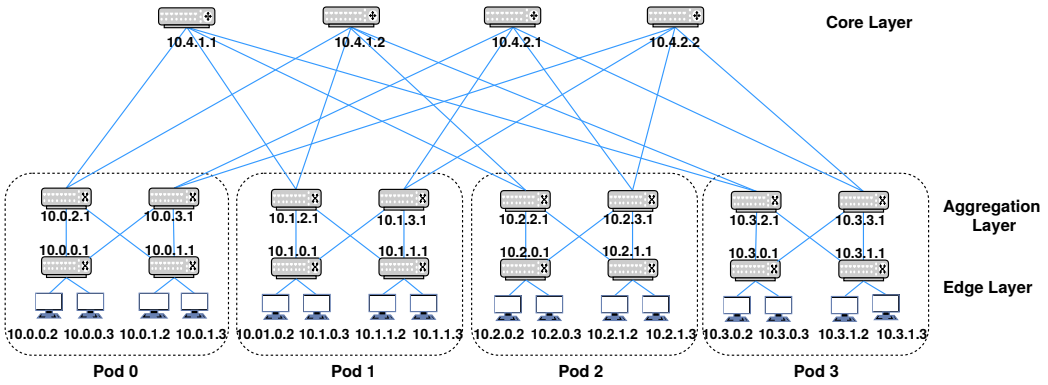

Fig. 3. Fat-Tree topology with $k = 4$.

TABLE I

FEATURES OF FAT-TREE AND BCUBE TOPOLOGIES: N=NUMBER OF SERVERS, n=NUMBER OF n-PORT SWITCHES, k= NUMBER OF BCUBE LEVELS

	Structure						Routing			
	Nbr of Servers	**Nbr of Switches**	**Nbr of Wires**	**Connecting Rule**	**Server Degree**	**Diameter**	**Decider**	**Type**	**Protocol**	**Node-Disjoint Paths**
Fat-Tree	$n^3/4$	$5N/n$	$3N$	A switch can connect to other switches. A server has only one direct link to another switch	1	6	Switches only	Two-level lookup table	Equal-cost multi-path routing (ECMP)	1
BCube	n^{k+1}	$(Nlog_nN)/n$	$N(k+1)$	Switch-to-switch connections are not allowed. A server has k+1 direct links to k+1 other switches.	$k+1$	$2(k+1)$	Servers and Switches	Hamming distance	BCube Source Routing (BSR)	$k+1$

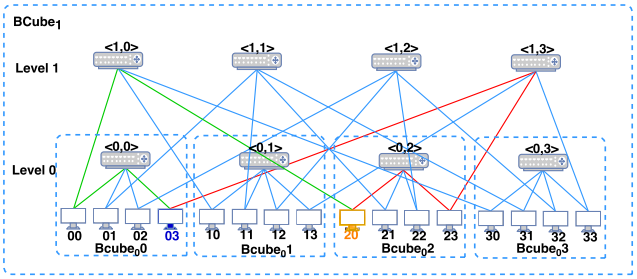

Fig. 4. Construction of $BCube_1$ from $BCube_0$ with $n = 4$. Source is (03) and destination is (20). Using hamming distance yields two parallel paths: (03) → (00) → (20) or (03) → (23) → (20).

of DCNs against node and link failures by increasing the number of possible paths among server pairs.

A. The Physical Structure

Similarly to Fat-Tree, the switches in `Circulant Fat-Tree` are also categorized into core, aggregation and edge switches. Edge and aggregation switches are arranged into k pods, where every edge switch connects $k/2$ servers. The total number of servers and switches are $k^3/4$ and $5 * k^2/4$ respectively. In `Circulant Fat-Tree`, every pod is connected to the adjacent pod through its aggregation and edge switches situated at the pod's border. Formally, if pod switches are addressed in the form $10.pod.switch.1$, then the connecting rule between adjacent pods are as follows. For every pod, $10.pod.k/2-1.1$ switch is connected to $10.pod + 1.k/2.1$ and $10.pod.k - 1.1$ is connected to $10.pod+1.0.1$. Hence, the number of wires in `Circulant Fat-Tree` is increased by $2(k-1)$ when compared to Fat-Tree. However, the increase in number of wires is negligible especially when the number of pods is large. For example, when $k = 24$ this increase is only 0.4%. Figure 5 shows an example of `Circulant Fat-Tree` where $k = 6$.

B. Key Features of `Circulant Fat-Tree`

`Circulant Fat-Tree` outperforms traditional Fat-Tree in each of the following performance metrics.

1) Alleviating Traffic Congestion at Core Switches: In Fat-Tree DCNs, the average link loads at the core switches are higher than those at aggregation and edge switches. For instance, any two servers belonging to two different pods can only communicate through core switches. However, unlike Fat Tree, in `Circulant Fat-Tree`, any two servers belonging to any two consecutive pods can communicate directly through aggregation switches without needing to go through core switches. For fair comparison, throughout, we only consider paths whose lengths (number of hops) do not

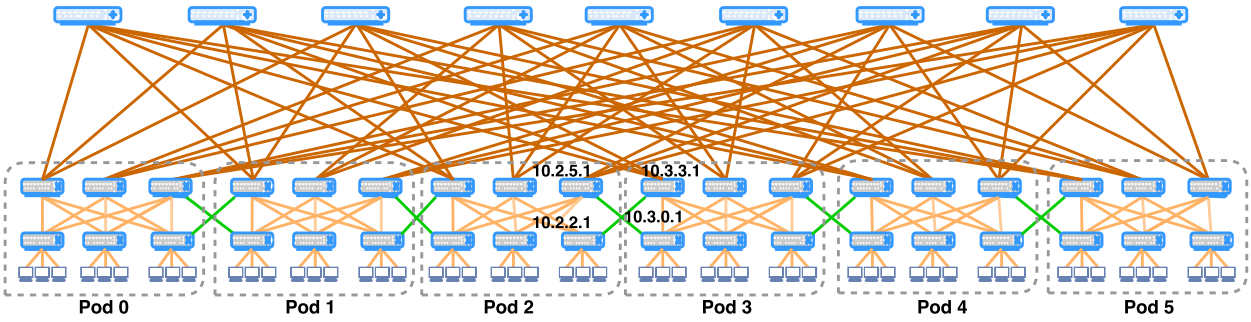

Fig. 5. The proposed `Circulant Fat-Tree` with $k = 6$.

TABLE II
APL COMPARISON

	APL: adjacent pods only		APL: all pods	
No. of Servers	Fat-Tree	`Circulant Fat-Tree`	Fat-Tree	`Circulant Fat-Tree`
128	6	5.125	5.688	5.496
1024	6	5.469	5.859	5.804
3456	6	5.681	5.911	5.884

exceed 6, since this is the maximum path length of Fat-Tree topologies.

2) Reducing Average Path Length (APL): APL is a key performance metric for evaluating DCN topologies, as it captures end-to-end latency of traffic between pairs of servers. The proposed `Circulant Fat-Tree` reduces APL of $(k^3/4 - k/2)$ pairs of servers[1] among a total of $(k^4/16)$ pair of servers from 6 (as in the case of Fat-Tree) to only 4. When $k = 24$, this, for example, corresponds to reducing APL of about 16% of the total pairs from 6 hops to 4 only. More on this will be provided in the next section.

3) Reducing Network Latency: Most DCN routing algorithms use the shortest path metric for making routing decisions. Since `Circulant Fat-Tree` reduces the APL among communicating servers, then the average network latency achieved under `Circulant Fat-Tree` is reduced when compared to Fat-Tree, especially for communicating servers belonging to adjacent pods; this is shown in Table II.

4) Improving Robustness Against Node/Link Failure: `Circulant Fat-Tree` is more tolerant to switch and server failures than Fat-Tree topologies. Robustness to network failures is improved because `Circulant Fat-Tree` increases the number of possible paths between pairs of servers belonging to consecutive pods when compared to Fat-Tree.

IV. PERFORMANCE EVALUATION:

In this section, we evaluate and compare our proposed `Circulant Fat-Tree` topology to Fat-Tree topology in terms of the metrics presented in Section III-B. To ensure a fair comparison, we only consider path lengths of at most 6 hops. Recall that `Circulant Fat-Tree` increases the number of servers that can be reached from another server via pod-level by at least 200% when compared to Fat-Tree. For example, when referring to Fig. 5, a server connected to switch 10.3.0.1 in pod 3 can communicates with 38 other servers without needing to go through core-level switches. On the other hand, this same server can only communicate with 8 servers when Fat-Tree topology is used for the same scenario. Fig. 6 shows the number of servers that can reach each other through pod-level switches instead of core-level switches for both topologies, Fat-Tree and `Circulant Fat-Tree`, with path length of at most 6 for different total numbers of servers.

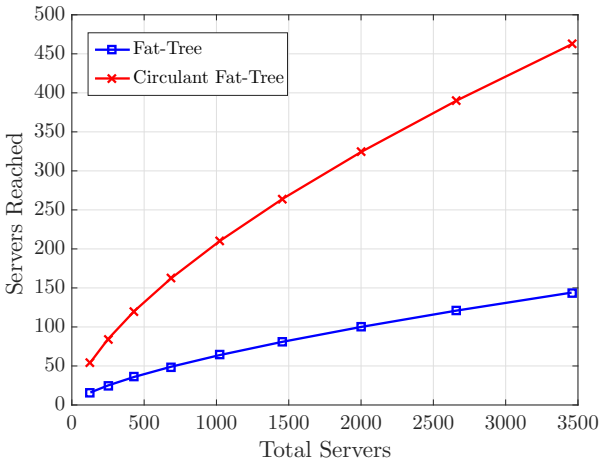

Fig. 6. Number of servers that can reach other servers without going through core-level switches with path length of at most 6.

Without loss of generality, to measure the network traffic, we suppose that each server sends one traffic unit (e.g., one packet) to every other server in the network. This scenario is called *all-to-all* traffic. We also assume that traffic is routed along shortest paths, where here the number of

[1]We only consider pairs whose servers belong to different pods.

hops is used to measure the path length. To ensure a fair comparison, we assume that the maximum shortest distance (in number of hops) among all the server pairs is at most 6 for both topologies. Fig. 7 shows the total amount of traffic that crosses core-level switches when all server pairs (each belonging to a different pod) are communicating. In this experiment, the number of servers is varied from 128 to 3456. Observe that `Circulant Fat-Tree` reduces the core-level traffic when compared to Fat-Tree, especially when the number of servers is low. This reduction varies from 35% to 10% when the number of servers goes from 128 to 3456.

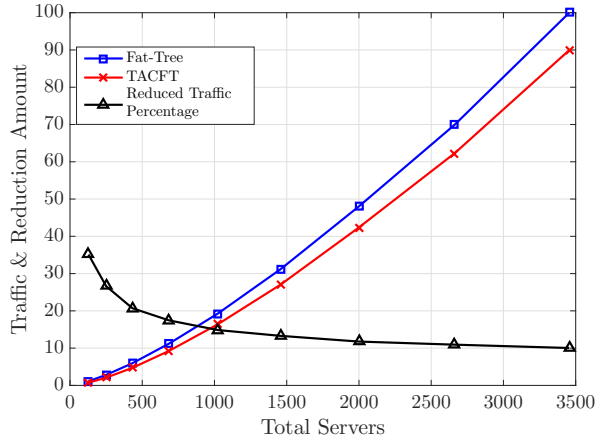

Fig. 7. All-to-All traffic between two random pods.

Fig. 8 shows that `Circulant Fat-Tree` reduces APL between any two adjacent pods when compared to Fat-Tree. APL achieved under `Circulant Fat-Tree` varies from 5.125 to 5.680 as opposed to 6 in the case of Fat-Tree.

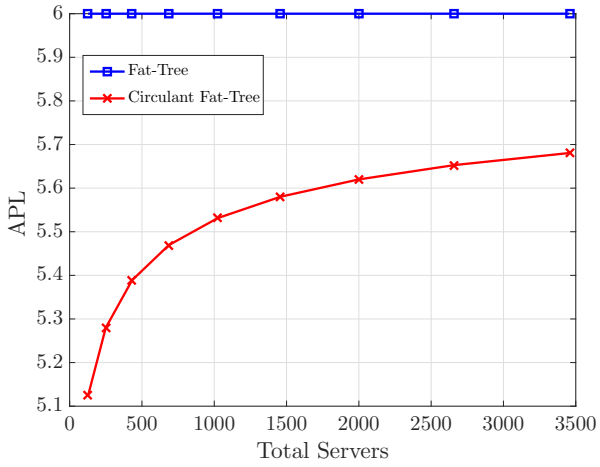

Fig. 8. APL between any two consecutive pods.

Fig. 9 shows the total amount of traffic that crosses core-level switches under all-to-all traffic scenario when considering server pairs in the entire network. The figure shows that lesser traffic goes through core-level switches under `Circulant Fat-Tree` than under Fat-Tree, with a traffic reduction that varies from 34% to 10% when the number of servers varies from 128 to 3456. Therefore, `Circulant Fat-Tree` reduces the congestion at core-level switches by balancing traffic among all switches: core, aggregation and edge. This is one of key features of `Circulant Fat-Tree`.

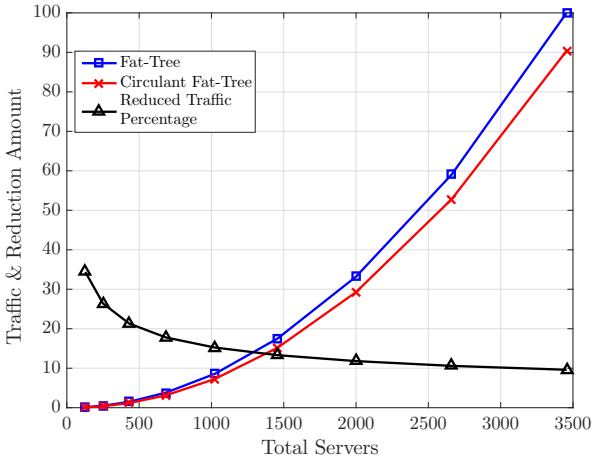

Fig. 9. All-to-All traffic in entire network.

Now when considering all servers communicating to all servers regardless of their pod, APL achieved under `Circulant Fat-Tree` is also smaller than that achieved under Fat-Tree. This is illustrated in Fig. 10.

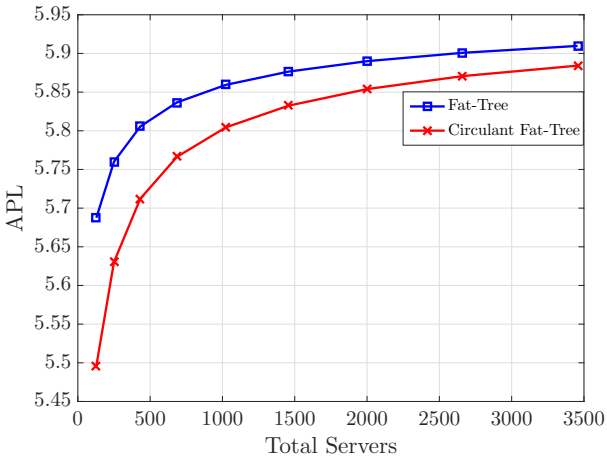

Fig. 10. APL for the entire network.

In `Circulant Fat-Tree`, the multiple redundant paths that are introduced between any two adjacent pods result in alleviating core-level traffic congestion, reducing network latency, and improving robustness to switch and server failures. We only consider paths of length ≤ 6 to ensure fairness with the Fat-Tree. Fig. 11 shows `Circulant Fat-Tree` results in more possible routing paths between any two adjacent pods than Fat-Tree. For example, in

`Circulant Fat-Tree`, when the total number of servers is 1024, any two servers in two consecutive pods have 78 possible routing paths with length $\leq$ 6. Fat-Tree, on the other hand, has only 64 possible routing paths with the same length.

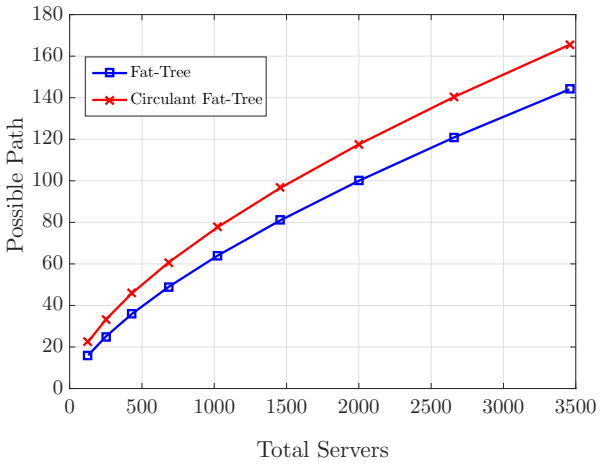

Fig. 11. Average number of possible paths between any two servers in two consecutive pods.

V. Conclusion

We propose the `Circulant Fat-Tree` topology, an improvement over the traditional Fat-Tree topology to better suit nowadays data center networks. The proposed `Circulant Fat-Tree` topology alleviates traffic congestions at core-level switches by providing a better balance of traffic loads across the different network switches, reduces latency of data communicated across servers by reducing the average path lengths among communicating servers, and augments robustness against network failures by increasing the number of possible paths between server pairs.

Acknowledgment

This work was supported in part by the US National Science Foundation under NSF award CNS-1162296.

References

[1] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," in *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 4. ACM, 2008, pp. 63–74.

[2] S. A. Jyothi, A. Singla, P. B. Godfrey, and A. Kolla, "Measuring and understanding throughput of network topologies," in *High Performance Computing, Networking, Storage and Analysis, SC16: International Conference for*. IEEE, 2016, pp. 761–772.

[3] L. Popa, S. Ratnasamy, G. Iannaccone, A. Krishnamurthy, and I. Stoica, "A cost comparison of datacenter network architectures," in *Proceedings of the 6th International COnference*. ACM, 2010, p. 16.

[4] A. Headquarters, "Cisco data center infrastructure 2.5 design guide," *Cisco Validated Design I*, 2013.

[5] C. V. networking Index, "Forecast and methodology, 2016-2021, white paper," *San Jose, CA, USA*, vol. 1, 2016.

[6] T. Benson, A. Akella, and D. A. Maltz, "Network traffic characteristics of data centers in the wild," in *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*. ACM, 2010, pp. 267–280.

[7] T. Benson, A. Anand, A. Akella, and M. Zhang, "Understanding data center traffic characteristics," in *Proceedings of the 1st ACM workshop on Research on enterprise networking*. ACM, 2009, pp. 65–72.

[8] C. Guo, G. Lu, D. Li, H. Wu, X. Zhang, Y. Shi, C. Tian, Y. Zhang, and S. Lu, "Bcube: a high performance, server-centric network architecture for modular data centers," *ACM SIGCOMM Computer Communication Review*, vol. 39, no. 4, pp. 63–74, 2009.

[9] 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 *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 4. ACM, 2008, pp. 75–86.

[10] R. Niranjan Mysore, A. Pamboris, N. Farrington, N. Huang, P. Miri, S. Radhakrishnan, V. Subramanya, and A. Vahdat, "Portland: a scalable fault-tolerant layer 2 data center network fabric," in *ACM SIGCOMM Computer Communication Review*, vol. 39, no. 4. ACM, 2009, pp. 39–50.

[11] A. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. A. Maltz, P. Patel, and S. Sengupta, "Vl2: a scalable and flexible data center network," in *ACM SIGCOMM computer communication review*, vol. 39, no. 4. ACM, 2009, pp. 51–62.

[12] G. Wang, D. G. Andersen, M. Kaminsky, K. Papagiannaki, T. Ng, M. Kozuch, and M. Ryan, "c-through: Part-time optics in data centers," in *ACM SIGCOMM Computer Communication Review*, vol. 40, no. 4. ACM, 2010, pp. 327–338.

[13] N. Farrington, G. Porter, S. Radhakrishnan, H. H. Bazzaz, V. Subramanya, Y. Fainman, G. Papen, and A. Vahdat, "Helios: a hybrid electrical/optical switch architecture for modular data centers," *ACM SIGCOMM Computer Communication Review*, vol. 40, no. 4, pp. 339–350, 2010.

[14] A. Singla, C.-Y. Hong, L. Popa, and P. B. Godfrey, "Jellyfish: Networking data centers, randomly." in *NSDI*, vol. 12, 2012, pp. 17–17.

[15] K. Chen, A. Singla, A. Singh, K. Ramachandran, L. Xu, Y. Zhang, X. Wen, and Y. Chen, "Osa: An optical switching architecture for data center networks with unprecedented flexibility," *IEEE/ACM Transactions on Networking*, vol. 22, no. 2, pp. 498–511, 2014.