Towards the Tradeoffs in Designing Data Center Network Architectures

Dawei Li, Jie Wu, *Fellow, IEEE*, Zhiyong Liu, and Fa Zhang

Abstract—Existing Data Center Network (DCN) architectures are classified into two categories: switch-centric and server-centric architectures. In switch-centric DCNs, routing intelligence is placed on switches; each server usually uses only one port of the Network Interface Card (NIC) to connect to the network. In server-centric DCNs, switches are only used as cross-bars, and routing intelligence is placed on servers, where multiple NIC ports may be used. In this paper, we formally introduce a new category of DCN architectures: the *dual-centric* DCN architectures, where routing intelligence can be placed on both switches and servers. The dual-centric philosophy can achieve various tradeoffs in designing DCN architectures. We propose three novel dual-centric DCN architectures: FCell, FRectangle, and FSquare, all of which are based on the folded Clos topology. FCell is a power-efficient DCN architecture, with a larger diameter and lower bisection bandwidth than FSquare and FRectangle. FSquare is a high performance DCN architecture, in which the diameter is small and the bisection bandwidth is large; however, the DCN power consumption per server in FSquare is high. FRectangle significantly reduces the DCN power consumption per server, compared to FSquare, at the sacrifice of some networking performances. By investigating FCell, FRectangle and FSquare, and by comparing them with existing architectures, we demonstrate that, the three novel dual-centric architectures enjoy the advantages of both switch-centric designs and server-centric designs, have various nice properties for practical data centers, and provide flexible tradeoff choices in designing DCN architectures.

Index Terms—Data center network (DCN), power consumption, end-to-end delay, bisection bandwidth, dual-centric design

1 Introduction

Data centers have become important infrastructures for supporting various cloud computing services. These vary from web search, email, video streaming, and social networking [1], [2], [3], to distributed file systems such as GFS [4], and distributed data processing engines, such as MapReduce and Dryad [5], [6]. The Data Center Network (DCN), which defines how the servers and various other components are interconnected, has significant influences on the quality of the services that the data center can provide to the applications that it hosts.

Performance versus Power. Two important performance metrics for a DCN architecture are end-to-end delays in the DCN and the bisection bandwidth. End-to-end delays translate directly to applications' response times in various situations. Bisection bandwidth provides key information on the potential throughput that the network can provide and the fault-tolerance capabilities. As servers are becoming more and more power efficient, the DCN tends to consume 50 percent of the total IT power [7]; thus, the DCN power consumption has become an important issue. To provide low end-to-end delays and high bisection bandwidth, large numbers of networking devices are usually used in DCNs. For example, in Fat-Trees [8], three levels of switches are used, resulting in high DCN power consumption. BCube [9] needs three or more levels of switches to scale the network to a considerable size; besides, BCube needs to use several Network Interface Card (NIC) ports on each server, which also contribute to the DCN power consumption. To achieve a low DCN power consumption, other designs use significantly fewer networking devices. For example, in DPillar [10], SWCube, SWKautz [11], DCell [12], BCN [13], and FiConn [14], the number of switches used is largely reduced, though a small number of extra NIC ports (typically less than 4) are required on servers. The DCN power consumption of these architectures is generally less than that of Fat-Trees and BCubes; however, these architectures rely heavily on servers for packet forwarding. Since servers usually have much greater processing delays than switches, especially when servers' packet forwarding schemes are software-based, the end-to-end delays in these architectures are much greater; besides, these architectures also have a lower bisection bandwidth. Can we achieve high performances and low power consumption at the same time?

Switch-Centric versus Server-Centric. Existing DCN architectures have been classified into two categories: switch-centric and server-centric architectures [15]. In switch-centric designs [7], [8], [16], routing intelligence is placed on switches; servers are equipped with one NIC port, and are not involved in forwarding packets for other servers. In server-centric designs [9], [10], [11], [12], [13], [14], switches are only used as cross-bars, and routing intelligence is placed on servers; servers are usually equipped with multiple NIC ports, and act as both computing and packet forwarding nodes. Switch-centric architectures enjoy the fast switching capability of switches,

- *D. Li and J. Wu are with the Department of Computer and Information Sciences, Temple University, Philadelphia, PA 19122. E-mail: {dawei.li, jiewu}@temple.edu.*
- *Z. Liu is with State Key Laboratory for Computer Architecture, Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100190, China. E-mail: zyliu@ict.ac.cn.*
- *F. Zhang is with Key Laboratory of Intelligent Information Processing, Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100190, China. E-mail: zhangfa@ict.ac.cn.*

Manuscript received 15 Mar. 2016; revised 11 Aug. 2016; accepted 5 Sept. 2016; Date of current version 14 Dec. 2016.
Recommended for acceptance by X. Wang.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TPDS.2016.2610970

but switches are less programmable than servers. Although Software Defined Networking (SDN) technologies, such as OpenFlow and Network Function Virtualization (NFV), increase the programmability on high-end switches, we are considering commodity-of-the-shelf low-end switches as most architectures use them to reduce the cost of constructing large scale DCNs. Server-centric architectures enjoy the high programmability of servers, but servers usually have larger processing delays than do switches. Can we combine the advantages of both categories?

Scalability versus Flexibility. In DCNs, scalability requires that the networking devices, typically the switches, rely on a small amount of information, which does not increase significantly with the network size, to make efficient routing decisions. Flexibility means that expanding the network in a fine-grained fashion should not destroy the current architecture or replace the networking devices. Since modern data centers usually have large network sizes, scalability is an important requirement. Also, data centers require flexible growth of network size after initial deployment, due to the rapidly increasing needs. Regular architectures are generally highly scalable, but do not support flexible growth of the network size due to their rigid topologies. Some regular architectures are able to increase the network size, but have certain limitations. For example, FiConn supports coarse-grained growth; because adding one level to the architecture will make the network size increase by tens, or even hundreds of times, which does not reflect practical needs; expanding DCell and BCube requires adding more NIC ports on all of the existing servers. Recent works have proposed random networks, such as Jellyfish [17], Scafida [18], and Small-World Data Center [19], to provide arbitrary-grained flexibility; however, due to their irregularity, networking devices need to use large routing tables for efficient routing, making them unable to scale to a large network size. Can we design both scalable and flexible DCN architectures?

In this paper, we consider the tradeoffs (in all of the above-mentioned three aspects) in designing DCN architectures. Our main contributions are as follows.

- First, we propose a unified path length definition, and consequently, a unified diameter definition, to characterize the end-to-end delays in a general DCN. Also, a DCN power consumption model is presented to characterize the power efficiency of general DCNs.
- Second, we introduce a new category of DCN architectures, i.e., the *dual-centric* DCN architectures, to complement the current classifications. To the best of our knowledge, we are the first to formally introduce this dual-centric design philosophy. We propose three novel typical dual-centric architectures: FCell, FRectangle, and FSquare.
- Third, based on our unified path length, diameter definitions and DCN power consumption model for general DCNs, we conduct quantitative comparisons between FCell, FRectangle, and FSquare and several typical existing DCN architectures. Results show that FCell, FRectangle, and FSquare reflect various tradeoff choices between network performances and DCN power consumption.
- We show that dual-centric architectures can have appealing properties for practical DCN designs. Routing simulations are conducted for the three proposed architectures to justify their performances under various traffic patterns and loads.

The rest of the paper is organized as follows. Section 2 presents the unified path length, DCN diameter definitions, and DCN power consumption model. We describe our novel DCN architectures, FCell, FRectangle, and FSquare in Sections 3, 4, and 5, respectively. We review related existing works in Section 6. Quantitative comparisons among several architectures are provided in Section 7. Supporting simulations are conducted in Section 8. Conclusions and future directions are described in Section 9.

2 Preliminaries

To characterize the end-to-end delays between two servers in a DCN, the concept of diameter is usually used, which is defined as the maximum length of the shortest path between any pair of two servers. However, for switch-centric and server-centric architectures, path lengths are calculated differently in existing works. For switch-centric architectures, the length of a path is calculated as the number of links in the path [20], [21]; for server-centric architectures, the length is calculated as the number of servers in the path (source and destination excluded) between the two servers, plus 1 [9], [10], [11], [12], [13], [14]. A diameter of six in Fat-Tree means something totally different from a diameter of six in BCube. However, a lot of works still compare these two different kinds of diameters [12], [15], [20], [21]. This somewhat confuses the understanding of end-to-end delays in general DCNs.

In a DCN, the end-to-end delay of a packet from a source server to a destination server consists of the delays on all the devices that the packet traverses. The devices include switches, servers and links. Referring to the classic delay models in [22], we investigate various sources of end-to-end delays in the DCN. In this paper, we assume that all the switches and servers are homogeneous. Packets on switches and servers experience three important delays: *processing delay*, *transmission delay*, and *queuing delay*; we denote them as $d_{w,p}$, $d_{w,t}$, $d_{w,q}$ and $d_{v,p}$, $d_{v,t}$, $d_{v,q}$ for switches and servers, respectively. The processing delay is the time required to examine the packet's head and determine where to direct the packet. Queuing delays largely depend on network traffic conditions and routing protocols. Currently, our focus is on the architectures of DCNs; thus, we do not consider the queuing delay explicitly in the modeling, and assume that $d_{w,q}=d_{v,q}=0$.

Switches can operate in two modes: store-and-forward and cut-through. In store-and-forward mode, a switch needs to receive all the flits of the packet before it forwards the packet to the next device. The total delay on the switch is $d_w=d_{w,p}+d_{w,t}$. The typical value of $d_{w,p}$ is around $2\ \mu s$ [23]. $d_{w,t}=S_{packet}/r_{bit}$, where S_{packet} is the size of the packet and r_{bit} is the data transmission rate. S_{packet} varies between 64 and 1,514 bytes. Given data transmission rate $r_{bit}=1$ Gbps, $d_{w,t}$ varies from about $0.5\ \mu$s to about $10\ \mu$s. In cut-through mode, a switch starts forwarding the packet when it receives the first flit of the packet. Thus, the transmission delay is negligible, and the total delay is around $d_w=d_{w,p}=2\ \mu$s.

The packet forwarding scheme on a server can be implemented in either software or hardware. In software-based forwarding, the processing delay on a server, $d_{v,p}$ is much higher than that on a switch, with a typical value of about 10 μs [23]. Depending on CPU load and NIC configuration, this value varies significantly. In hardware-based forwarding, $d_{v,p}$ can be close to the processing delay on a switch [24]. The overall delay on a server is d_v=$d_{v,p}$+$d_{v,t}$, where $d_{v,t}$ can be calculated in the same way as $d_{w,t}$. Based on the typical values, d_v is generally one to several times of d_w.

Network links have *propagation delay*, d_l=$L_{link}/(\eta c)$, where L_{link} is the length of the link, η is a constant around 0.7, and c is the speed of light in vacuum. Since the length of links in a data center is usually less than 10 meters, the propagation delay on a link is usually less than $10/(0.7\times3\times10^8)$ s= 0.048 μs. Compared with the typical delays on switches and servers, the propagation delay is negligible.

Unified Path Length and Diameter Definitions. In general DCNs, both switches and servers may be used for packet forwarding. Denote the numbers of switches and servers in a path, P from a source server to a destination server by $n_{P,w}$, and $n_{P,v}$ (excluding the source and the destination), respectively. We define the *path length* of P as follows:

$$d_P = n_{P,w}d_w + (n_{P,v}+1)d_v, \tag{1}$$

where 1 is added to $n_{P,v}$, because the delay on the source server should be included as part of the end-to-end delay. The above path length definition applies to all general DCNs. If we assume that d_w=d_v=1, the above path length definition is consistent with the path lengths in a switch-centric architecture. If we assume that d_v=1 and that d_w is negligible, the above path length definition is consistent with the path lengths in a server-centric architecture. Under this unified path length definition, we define the *diameter of a general DCN* as the maximum path length (based on (1)) of the shortest paths between all pairs of servers in the DCN

$$d = \max_{P\in\mathbf{P}} d_P, \tag{2}$$

where **P** is the set of shortest paths between all pairs of servers in the DCN.

Again, queueing delays are not considered explicitly in the modeling. However, queueing delays may have different influences on the average packet delivery time in an architecture. Generally, when an architecture has a low bisection bandwidth, more queueing delays will be involved, and thus, the average packet delivery time for that architecture will be increased.

DCN Power Consumption Model. We consider the power consumption of all DCN devices. A switch's power consumption, p_w is part of the DCN power consumption. For a server in a switch-centric architecture, only the NIC's power consumption, p_{nic} belongs to the DCN power consumption. In a DCN where the server can be used for packet forwarding for other servers, the power consumption of the server's packet forwarding engine should also be included as the DCN power consumption. We denote p_{fwd} as the power consumption of the server's packet forwarding engine (either the CPU core's power consumption for software-based forwarding [25] or the additional hardware's power consumption for hardware-based forwarding [24]), and denote the extent to which a server is involved in packet forwarding by α. The overall *DCN power consumption* can be calculated as follows: p_{dcn}=$N_w p_w$+$n_{nic}N_v p_{nic}$+$\alpha N_v p_{fwd}$, where N_w and N_v are the numbers of switches and servers in the DCN, respectively, and n_{nic} is the average number of NIC ports used on a server. Since different DCNs can hold different numbers of servers, we define the *DCN power consumption per server* as the power efficiency metric of a general DCN

$$p_V = p_{dcn}/N_v = p_w N_w/N_v + n_{nic}p_{nic} + \alpha p_{fwd}. \tag{3}$$

For switch-centric architectures, α=0. For DCNs where servers are involved in packet forwarding for other servers, α depends on various factors; for simple and fair comparisons, we choose α=1. A practical value of p_w for a switch with 48 1 Gbps ports is about 150 Watts [26]; a practical value of p_{nic} for 1 Gbps NIC port is two Watts [27]. As reported in [25], when software-based forwarding is used, the CPU cores can be in reserved or shared models, which correspond to different p_{fwd} values, varying around five Watts if NIC ports are 10 Gbps. The value for p_{fwd} will be lower if NIC ports are 1 Gbps. In hardware-based forwarding, p_{fwd} may also have quite different values [24].

Notice that practical power consumptions of devices also depend on the current traffic density; we use the simplified static power consumption model to provide a unified comparison metric for all architectures. Existing works apply the same strategy to come up with meaningful comparisons among different architectures [21]. In a later section (Section 7), we will investigate how different power consumption values influence the comparisons among different architectures.

3 FCell

One motivation of our work is to design high performance architectures with low DCN power consumption. An intuitive remedy for switch-centric architectures, such as Fat-Tree, is to reduce the levels of switches. However, this makes the DCN unable to scale to a practically large size. Thus, we consider using interconnections among servers to scale the network. In this and the following two sections, we present three novel DCN architectures that belong to the dual-centric category: FCell, FRectangle, and FSquare. The three architectures are all based on the folded Clos topology [16], and use the same basic building block. Each server in these architectures uses two NIC ports. The reason why we consider servers with two NIC ports is that practical servers usually come with two NIC ports, one for primary use and one for backup. This is a common fault-tolerant design. Thus, the second port is seldom used. In our paper, we design novel architectures that can better utilize two-port servers, and do not sacrifice the fault tolerance. We will first describe the basic building block, and then introduce them one by one. We demonstrate the tradeoffs (between server-centric and switch-centric, and between scalability and flexibility) of our three proposed architectures by using FCell as an example in this section. Discussions on the dual-centricness and tradeoff between scalability and flexibility of FRectangle and FSquare are similar, and are omitted due to the page limit.

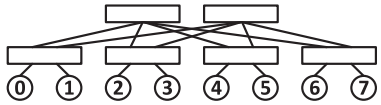


Fig. 1. The basic building block: Cluster. We consistently use rectangles to represent switches and circles to respresent servers, if not otherwise specified.

3.1 Basic Building Block

The basic building block is called a *cluster*. In each cluster, there are two levels of switches: n level 1 switches and $n/2$ level 2 switches. Every level 1 switch is connected to every level 2 switch. Then, there are $n/2$ ports remaining on every level 1 switch; we use these ports to connect $n/2$ servers. Thus, the switches and servers in one cluster form a simple instance of the folded Clos topology. The numbers of switches and servers in each cluster are $3n/2$ and $n^2/2$, respectively. The interconnections of a cluster with n=4 is shown in Fig. 1. In all of our discussions, we assume $n \geq 4$.

A level 1 switch is also called a *Top of Rack (ToR)* switch. When the servers align in a row, as in FRectangle or FSquare, we call it a *row ToR* switch; when the servers align in a column, we call it a *column ToR* switch.

3.2 FCell Construction

An FCell built from servers with two NIC ports and switches with n ports is denoted by FCell(n). An FCell(n) consists of $n^2/2+1$ clusters. Servers in all of the $n^2/2+1$ clusters are interconnected in a similar way to that of DCell [12]. Simply put, each of the $n^2/2$ servers in a cluster is directly connected to another server in each of the other $n^2/2$ clusters. Thus, if we regard each cluster as a single node, the $n^2/2+1$ clusters will form a complete graph. We denote a server by $a_{i,j}$, which represents the jth server in the ith cluster, $\forall 0 \leq i \leq n^2/2, 0 \leq j \leq n^2/2-1$. The interconnection rules we use are: server $a_{j+1,i}$ is connected with server $a_{i,j}$, $\forall i \leq j \leq n^2/2-1, \forall 0 \leq i \leq n^2/2-1$. Fig. 2 shows the interconnections of an FCell(4).

3.3 Routing in FCell

We present two basic routing schemes: shortest path routing (SRouting) and detour routing (DRouting), to show that FCell reflects a tradeoff between switch-centric and server-centric designs. Notice that, the two basic routing schemes are also useful for practical routing protocol design.

3.3.1 Shortest Path Routing

We denote the source and destination servers by $a_{i,j}$ and $a_{k,l}$ ($0 \leq i,k \leq n^2/2$ and $0 \leq j,l \leq n^2/2-1$), respectively. Then, the ith and the kth clusters are called source and destination clusters, respectively.

If $a_{i,j}$ and $a_{k,l}$ are in the same cluster, i.e., i=k, $a_{i,j}$ sends the packet to its level 1 (ToR) switch, which checks whether the destination is in the local rack. If the destination is in the local rack, the level 1 switch forwards the packet to the destination. Otherwise, it forwards the packet to a randomly chosen level 2 switch; the level 2 switch checks which rack the destination is in, and forwards the packet to the corresponding level 1 switch, i.e., the $\lfloor l/(n/2) \rfloor$th level 1 switch, which forwards the packet to the destination directly.

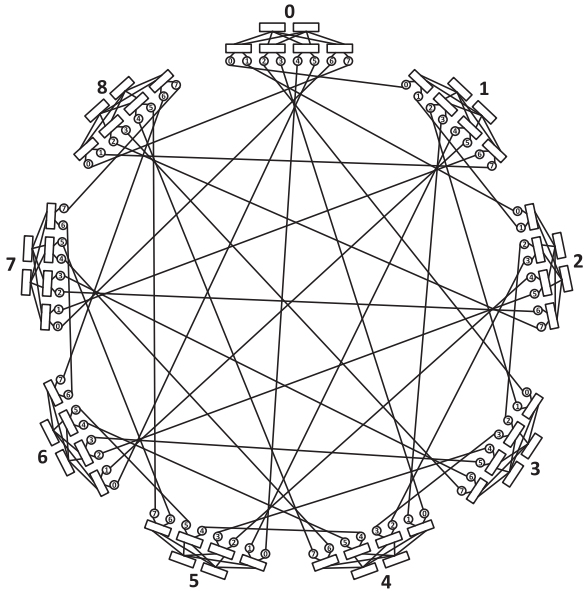


Fig. 2. FCell(4).

If $a_{i,j}$ and $a_{k,l}$ are not in the same cluster, based on servers' interconnection rules in FCell, the source server can determine the two servers (one in the source cluster, denoted by a_{i,r_1} and one in the destination cluster, denoted by a_{k,r_2}) that connect the source and destination servers. Then, $a_{i,j}$ forwards the packet to a_{i,r_1} within the source cluster. After that, a_{i,r_1} sends the packet to a_{k,r_2} directly, since they are directly connected. Finally, a_{k,r_2} forwards the packet to $a_{k,l}$ within the destination cluster.

We can see that, in all cases, the source server can determine all the server(s) in the path (including the destination) before sending the packet. Since the servers have high programmability and the decision logic is quite simple, we place the task of determining all the servers in the path on the source server. The source server initializes a *server stack (srv_stk)* that pushes the servers from the last one to the first one in the path, and indicates whether they are the true destination of the packet. For example, for the case where $a_{i,j}$ and $a_{k,l}$ are not in the same cluster, and $i<k$, $j \neq k-1$, $i \neq l$, all the servers in the shortest path are $a_{i,k-1}$, $a_{k,i}$, and $a_{k,l}$ (including the destination). The source server labels $a_{k,l}$ as the *true* destination and labels $a_{i,k-1}$ and $a_{k,i}$ as *fake* destinations. Then, it pushes $a_{k,l}$, $a_{k,i}$ and $a_{i,k-1}$ into *srv_stk* one by one. When sending the packet to a ToR switch, the source server uses the next server of the packet as the *temporary* destination, based on which, switches make decisions within the local cluster.

When another server in the DCN receives the packet, it pops the *srv_stk* of the packet. If the popped value is a true destination, the server consumes the packet. If the popped value is a fake destination, it checks whether the next server in the path of the packet is in the local cluster. If yes, it sends the packet to its ToR switch, using the next server as the temporary destination. Otherwise, it sends the packet to the next server directly.

When a switch receives the packet, only the destination (either fake or true) set by the previous sending server is visible to the switch. We call this destination a *temporary* destination. The switch makes forwarding decisions based on

this temporary destination. The behaviors of switches are similar to the case when the real source and destination are within the same local cluster.

Notice that, instead of randomly choosing, a level 1 switch can wisely choose a level 2 switch, if related information is available, and if it has the intelligence to do so. Thus, levels 1 and 2 switches can help with load-balancing, traffic-aware, fault-tolerant or even multi-path routing within the local cluster.

3.3.2 Detour Routing

The problem with the shortest path routing is that, if servers in two clusters have intensive communications, the link that directly connects the two clusters will become congested. To solve this problem, a detour routing scheme can be applied. Instead of determining the shortest path from the source to the destination directly, the source server can choose to detour the packet to a randomly chosen intermediate cluster before the packet arrives at the destination cluster; we call the intermediate cluster the *relay* cluster.

After choosing the relay cluster, also based on servers' interconnection rules in FCell, the source server can determine the *first relay server* (in the relay cluster), which has a direct connection with a server in the source cluster, and the *second relay server* (in the relay cluster), which has a direct connection with a server in the destination cluster. Then, the *detour path* consists of the shortest path from the source server to the first relay server, the shortest path from the first relay server to the second relay server, and the shortest path from the second relay server to the destination server. The source server initializes the *srv_stk* by pushing the servers from the last one (the destination) to the first one in the detour path, and then sends the packet into the network. All the routing logics on other servers, level 1 switches and level 2 switches need not to be changed, compared with the shortest path routing scheme.

Notice that, instead of randomly choosing the relay cluster, if related information, such as the traffic conditions in other clusters, is available at the source server, the source server can make wiser decisions for choosing the relay cluster. Thus, detour routing provides the basic mechanism for load balancing, traffic-aware, fault-tolerant, and even multi-path routing among clusters in the network.

FCell serves as a good example of dual-centric architectures, where both switches and servers can have some degree of routing intelligence. As indicated, switches and servers in FCell can help with load-balancing, traffic-aware, and fault-tolerant, and even multi-path routing within the local cluster, and among clusters, respectively. Besides, the number of servers in the shortest paths (excluding the source and the destination) is upper bounded by 2; even in a basic detour path, the number of servers is upper bounded by 4. Thus, FCell enjoys both the fast switching capability of switches and the high programmability of servers, without significantly increasing the end-to-end delays.

3.4 FCell Basic Properties

Property 1. *In an FCell(n), the number of switches is $N_w = 3n(n^2+2)/4$, and the number of servers is $N_v = n^2(n^2+2)/4$.*

Proof. There are $n^2/2+1$ clusters, each with $3n/2$ switches and $n^2/2$ servers. □

Property 2. *The diameter of an FCell(n) is $d = 6d_w + 3d_v$.*

Proof. The diameter is defined as the maximum length of the shortest path between two servers. Obviously, the longest shortest path in an FCell is between two servers that are not in the same cluster. We consider two servers, $a_{i,j}$ and $a_{k,l}$, which are not in the same cluster, i.e., $i \neq k$. Without loss of generality, we assume that $0 \leq i < k \leq n^2/2$. According to the interconnection rules of FCell, the server $a_{i,k-1}$ in the ith cluster, and the server $a_{k,i}$ in the kth cluster are directly connected. The shortest path from server $a_{i,j}$ to server $a_{k,l}$ consists of at most three segments: 1) the shortest path from server $a_{i,j}$ to server $a_{i,k-1}$ in the ith cluster, 2) the path from server $a_{i,k-1}$ to server $a_{k,i}$, and 3) the shortest path from server $a_{k,i}$ to server $a_{k,l}$ in the kth cluster. Notice that, the shortest path from $a_{i,j}$ to $a_{i,k-1}$ includes at most three switches; also, the shortest path from $a_{k,i}$ to $a_{k,l}$ includes at most three switches. Thus, the shortest path from $a_{i,j}$ and $a_{k,l}$ includes at most six switches, and at most two servers (excluding the source and destination). According to (1) and (2), the diameter of an FCell(n) is $d = 6d_w + 3d_v$. □

We assume that all the links in a DCN have a unit bandwidth, 1. Then, the bisection bandwidth of a DCN is the minimal number of links to be removed to partition the DCN into two parts of "equal" sizes that differ by at most 1. We conjecture that FCell has the following property.

Property 3. *The bisection bandwidth of an FCell(n) is $B \approx N_v/4$.*

Proof. The bisection bandwidth of a complete graph with N nodes, when N is even, is $N/2 \times N/2 = N^2/4$. The reason is quite straightforward. The cut partitions the N nodes into two equal sets, each consisting of $N/2$ nodes. Since each node in one set has a link to every node in the other set, the total number of links in the cut is $N/2 \times N/2 = N^2/4$. When N is odd, the bisection bandwidth is $(N+1)/2 \times (N-1)/2 = (N^2-1)/4$. As has been mentioned, for the FCell architecture, we can regard each of the $(n^2/2+1)$ clusters as a single node; then, these nodes form a complete graph. Besides, within each cluster, the architecture has a much greater bisection bandwidth, just like Fat-Tree and folded-Clos. For a cut to have a minimum number of links, it should try to avoid cutting through the clusters. Thus, the cut should cut between clusters. As a result, the bisection bandwidth of FCell is approximately equal to the bisection bandwidth of a complete graph with $(n^2/2+1)$ nodes: $B \approx 1/4(n^2/2+1)^2 \approx N_v/4$. □

Property 4. *The DCN power consumption per server of an FCell(n) is $p_V = 3p_w/n + 2p_{nic} + p_{fwd}$.*

Proof. The switch-number to server-number ratio in an FCell(n) is $N_w/N_v = 3/n$; in FCells, all servers are equipped with two NIC ports, and servers may be involved in forwarding packets for other servers. □

We further investigate the number of parallel paths between two servers in our proposed DCN architectures. We consider two types of parallel paths: intra-cluster switch-disjoint parallel paths, and inter-cluster server-disjoint parallel paths.

Definition 1. *The number of intra-cluster switch-disjoint parallel paths between two servers is the number of distinct paths that do not share the same switches, excluding the source and destination switches, within the cluster.*

Definition 2. *The number of inter-cluster server-disjoint parallel paths between two servers is the number of distinct paths that do not share the same servers, excluding the source and destination servers, across the clusters.*

It is easy to notice that, the number of switch-disjoint parallel paths in every cluster (the basic building block) is $n/2$, because there are $n/2$ level 2 switches. We are more interested in the number of inter-cluster server-disjoint parallel paths. Since in practice, the delay on servers is generally much larger than that of switches, we focus on the number of servers in the server-disjoint parallel paths.

Property 5. *The number of inter-cluster server-disjoint parallel paths between two servers (belonging to two different clusters), with length $n_w d_w+(n_v+1)d_v$, where $n_v \leq 2$, is at most 2.*

Illustration. Actually, we have simplified the shortest path routing in our previous discussion. In some rare cases, we can find other shortest paths using other approaches, and even the shortest path generated by the previous algorithm may have greater length than other approaches. We give an example here. We take $a_{0,2}$ and $a_{2,2}$ in an FCell(4) as the source and destination. Using the previous shortest path algorithm, we generate the path as: $a_{0,2} \rightarrow a_{0,1} \rightarrow a_{2,0} \rightarrow a_{2,2}$; its length is $6d_w+3d_v$. However, we can have another path: $a_{0,2} \rightarrow a_{3,0} \rightarrow a_{3,2} \rightarrow a_{2,2}$; its length is $3d_w+3d_v$. The condition for these rare cases it that, the source and destination both connect to the same other cluster.

Property 6. *The number of inter-cluster server-disjoint parallel paths between two servers (belonging to two different clusters), with length $n_w d_w+(n_v+1)d_v$, where $n_v \leq 4$, is $n^2/2$.*

Illustration. Here, we are actually counting the number of paths that are detoured at most once. For a detoured path generated by the detour routing procedure, the number of clusters involved is 3. In each of the three clusters, at most two servers will be included in the path (including the source and destination). Thus, there will be at most a total of six servers in a detour path. We can see that each detour path will be a path length $n_w d_w+(n_v+1)d_v$, where $n_v \leq 4$. Excluding the source cluster and the destination cluster, we have $n^2/2-1$ choices to select a relay cluster. Thus, the number of detour paths will be at $n^2/2-1$. Plus the shortest path (from the source to the destination) whose length is at most $6d_w+3d_v$, we will have $n^2/2$ parallel paths between two servers (belonging to two different clusters), with length $n_w d_w+(n_v+1)d_v$, where $n_v \leq 4$.

3.5 FCell Scalability and Flexibility

FCell has good scalability due to its regularity. As can be seen in the basic routing schemes, switches in FCell only need local information for packet forwarding; thus, the routing table size on each switch can be kept small. Servers only need basic configuration parameters of FCell for packet forwarding. In other words, they both need a small amount of information to make efficient routing decisions. Thus, FCell is highly scalable.

Unlike various rigid regular architectures, FCell supports flexibility quite well, i.e., it allows fine-grained incremental growth of its network size. We call the FCell(n) constructed previously in this paper a *complete* FCell(n). FCell supports two fundamental ways to expand the network.

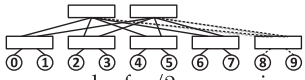
The first way is to expand a complete FCell. In this case, we require that the level 2 switches have a number of ports reserved for future expansion. Using one reserved port on each of the level 2 switches, we are able to add one level 1 switch with $n/2$ servers to each cluster, by connecting the added level 1 switch with each of the $n/2$ level 2 switches in the cluster. We call the cluster with $n/2$ added servers an *expanded cluster*. After this, each of the $n^2/2+1$ expanded clusters in the FCell(n) will have $n/2(n/2+1)$ servers, among which, $n/2$ added servers are not directly connected to other servers. Thus, we are allowed to add $n/2$ expanded clusters into the current architecture. Adding the first expanded cluster, the $(n^2/2)$th server of the ith expanded cluster is connected to the ith server in the newly added expanded cluster, which becomes the $(n^2/2+1)$th expanded cluster, $\forall 0 \leq i \leq n^2/2$. Adding the second expanded cluster, the $(n^2/2+1)$th server of the ith expanded cluster is connected to the ith server in the newly added $(n^2/2+2)$th expanded cluster, $\forall 0 \leq i \leq n^2/2+1$. Continuing this process until adding the $(n/2)$th expanded cluster, the $(n^2/2+n/2-1)$th server of the ith cluster is connected to the i server in the newly added $(n^2/2+n/2)$th expanded cluster, $\forall 0 \leq i \leq n^2/2+n/2-1$.

Fig. 3a shows adding one level 1 switch and 2 servers to each cluster of the complete FCell(4). Fig. 3b shows adding one expanded cluster to the existing architecture. Fig. 3c shows adding the second expanded cluster. Dashed lines represent added links compared with the existing architecture.

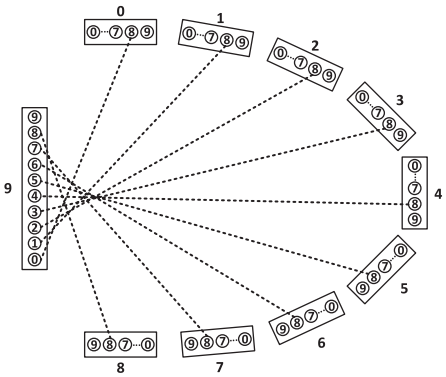
Notice that, the original interconnections among switches and servers are never modified. The original architecture consists of $N_v^{original}=n^2(n^2+2)/4$ servers. After expanding, the architecture consists of $N_v^{expanded}=(n^2/2+n/2)(n^2/2+n/2+1)$ servers. The increase of the number of servers for $n=24$ is from 83,232 to 90,300, i.e., an increase of 8.49 percent. The increase for $n=48$ is from 1,328,256 to 1,384,152, i.e., an increase of 4.21 percent. In this way, FCell supports a fine-grained incremental growth of network size, without modifying its original interconnections.

We have used one reserved port on each of the level 2 switches. If there are k ports reserved for future expansion on each level 2 switch, the expanded architecture can reach $(n^2/2+nk/2)(n^2/2+nk/2+1)$ servers. For $k=n$, it indicates a size of approximately 4 times of the original size; we argue that this meets typical requirements of network size growth. We have to admit that having reserved ports on level 2 switches is a drawback. However, only one third of the switches need to have reserved ports; this cuts down the extra initial investment for future expansion.

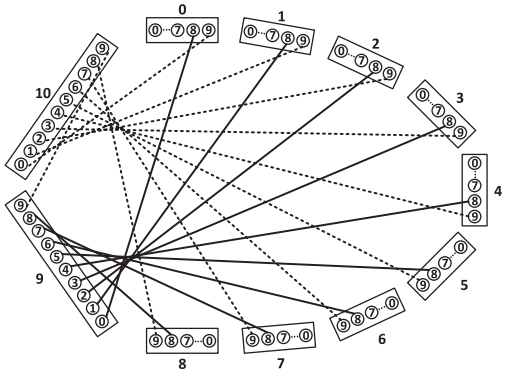
FCell supports another way of expanding its network size. Instead of using $n^2/2+1$ clusters, we can use



(a) Adding one rack of $n/2$ servers in each cluster.



(b) Adding the first expanded cluster.



(c) Adding the second expanded cluster.

Fig. 3. Illustration of flexible expansion. In (b) and (c), each cluster is simplified by a rectangle enclosing circles.

$m+1 < n^2/2+1$ clusters to build an *incomplete* FCell, by connecting the first m $(< n^2/2)$ servers in all of the $m+1$ clusters. The method for adding clusters to an incomplete FCell is similar to that of expanding a complete FCell. This provides the possibility of flexibly adding servers, without reserving ports on the level 2 switches. Of course, the two ways to expand the network can be combined. When expanding an incomplete FCell makes the FCell complete, we can further expand the complete FCell.

Since the original interconnections among switches and servers are never modified, after expanding the network, very limited information needs to be updated for switches and servers to make efficient routing decisions. Therefore, scalability of FCell is well maintained.

4 FRectangle

4.1 FRectangle Construction

FRectangle is constructed with two dimensions. Each column of the architecture is our basic building block, the cluster. In each row, n switches are used to interconnect n^2 servers. Each row of the FRectangle architecture chooses one type of interconnections from the following:

- Type A interconnections: For servers in the ith row, $a_{i,j}, 0 \le j \le n^2-1$, if $kn \le j \le kn+n-1, (0 \le k \le n-1)$, then $a_{i,j}$ is connected to the kth switch in this row.

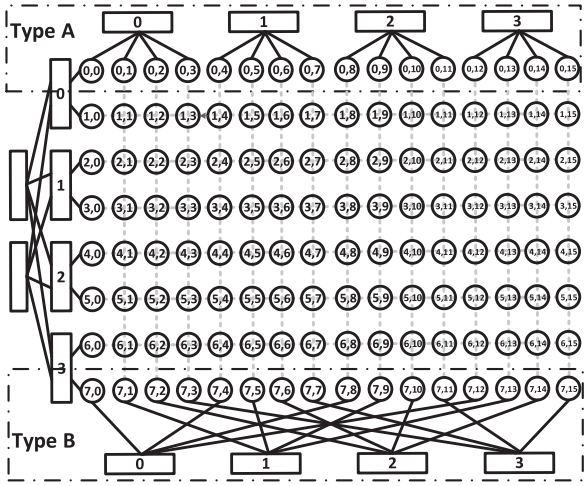


Fig. 4. FRectangle(4).

- Type B interconnections: For servers in the ith row, $a_{i,j}, 0 \le j \le n^2-1$, if $j\%n=k$, $(0 \le k \le n-1)$, then $a_{i,j}$ is connected to the kth switch in this row.

We let FRectangle choose from the two types of interconnections in an interleaved fashion: if $i\%2$=0, the ith row chooses the type A interconnections; if $i\%2$=1, the ith row chooses the type B interconnections. An FRectangle constructed by switches with n ports is denoted FRectangle(n). Fig. 4 shows an FRectangle(4). Notice that we only draw the zeroth column, the zeroth and the last rows; other columns and rows are represented by grey dashed lines.

4.2 Routing in FRectangle

Before presenting the routing scheme, we introduce some characteristics in FRectangle.

Characteristic 1. *For two servers that belong to the same type of rows, the communication between them may or may not need a row of a different type to relay.*

Illustration Given two servers in the same row, $a_{i,j}$ and $a_{i,l}$, we first consider the case when $i\%2 = 0$, i.e., the row is a type A row. If $\lfloor j/n \rfloor = \lfloor l/n \rfloor$, then $a_{i,j}$ and $a_{i,l}$ are connected to the same row ToR switch, i.e., the $\lfloor j/n \rfloor$th ToR switch in the row; otherwise, they are not connected in this row. For the case when $i\%2 = 1$, the discussions are similar. Moreover, for two servers, $a_{i,j}$ and $a_{k,l}$ that belong to the same type of rows. The communication between them may need a row of a different type to relay. Taking $i\%2 = k\%2 = 0$ as an example, if $\lfloor j/n \rfloor \neq \lfloor l/n \rfloor$, without using a type B row, $a_{i,j}$ and $a_{k,l}$ will never be connected; thus, the communication between $a_{i,j}$ and $a_{k,l}$ must need server(s) in a type B row to relay.

Characteristic 2. *For a rack of $n/2$ servers that are connected to the same column ToR switch, there exists at least one server belonging to a type A row, and at least one server belonging to a type B row.*

Illustration. There are $n/2 \ge 2$ servers connecting to the kth column ToR switch in the jth column, i.e., servers $a_{kn/2,j}$, $a_{kn/2+1,j}$, ..., $a_{kn/2+n/2-1,j}$. Since rows choose type A and type B interconnections in an interleaved fashion, this characteristic follows directly.

Now, we are ready to consider the detailed shortest path routing in FRectangle. We denote the source and destination servers by $a_{i,j}$ and $a_{k,l}$ ($0 \le i,k \le n^2/2-1$ and $0 \le j,l \le n^2-1$), respectively. If $a_{i,j}$ and $a_{k,l}$ are in the same column, i.e., j=l, the shortest path will be within the column. This case is essentially the same as the basic case in FCell, and requires no further explanation.

In the following, we consider the general cases where $a_{i,j}$ and $a_{k,l}$ are not in the same column. Based on the observed characteristics, the types of rows that the source and destination belong to make the most important difference. Thus, we classify the cases according to the row types of the source and destination servers.

Case 1. The source, $a_{i,j}$ belongs to a type A row, and the destination, $a_{k,l}$ belongs to a type B row, i.e., $i\%2 = 0$ and $k\%2 = 1$. A packet from $a_{i,j}$ to $a_{k,l}$ does not need to traverse servers in rows other than the ith row and the kth row. Notice that, $a_{i,j}$ is connected to the $\lfloor j/n \rfloor$th row ToR switch in the ith row; besides, servers, $a_{i,\lfloor j/n \rfloor n}, a_{i,\lfloor j/n \rfloor n+1}, a_{i,\lfloor j/n \rfloor n+2}, \ldots, a_{i,\lfloor j/n \rfloor n+n-1}$ are also connected to the $\lfloor j/n \rfloor$th row ToR switch in the ith row. Notice also that, $a_{k,l}$ is connected to the $(l\%n)$th row ToR switch in the kth row; besides, servers, $a_{k,(l\%n)}, a_{k,(l\%n)+n}, a_{k,(l\%n)+2n}, \ldots, a_{k,(l\%n)+(n-1)n}$ are also connected to the $(l\%n)$th row ToR switch in the kth row. Thus, we can find the column number $c^* = \lfloor j/n \rfloor n + (l\%n)$, such that a_{i,c^*} is connected to the same row ToR switch as $a_{i,j}$, and that a_{k,c^*} is connected to the same row ToR switch as $a_{k,l}$. We use a_{i,c^*} and a_{k,c^*} as the *first relay server* and the *second relay server*, to help forward packets from $a_{i,j}$ to $a_{k,l}$. Notice that the shortest path from a_{i,c^*} to a_{k,c^*} is in the same column and requires no further explanation. Thus, the shortest path from $a_{i,j}$ to $a_{k,l}$ consists of three segments: the path from $a_{i,j}$ to a_{i,c^*}, which includes the $\lfloor j/n \rfloor$th row ToR switch in the ith row, the shortest path from a_{i,c^*} to a_{k,c^*}, and the path from a_{k,c^*} to $a_{k,l}$, which includes the $(l\%n)$th row ToR switch in the kth row. Cases where $a_{i,j}$ is identical to a_{i,c^*}, and/or a_{k,c^*} is identical to $a_{k,l}$, are just special cases which require no further explanation.

Case 2. Source $a_{i,j}$ belongs to a type B row, and destination $a_{k,l}$ belongs to a type A row, i.e., $i\%2 = 1$ and $k\%2 = 0$. The situation is very similar to the previous one. The shortest path can be constructed by reversing the source and destinations; thus, we omit further discussion here.

Case 3. Source $a_{i,j}$ and destination $a_{k,l}$ both belong to type A rows, i.e., $i\%2$=$k\%2$=0. We need to consider which columns that the source and destination are in. Notice that in this case, whether i is or is not equal to k makes little difference. If $\lfloor j/n \rfloor$=$\lfloor l/n \rfloor$, then $a_{i,j}$ is connected to the $\lfloor j/n \rfloor$th row ToR switch in the ith row, and $a_{k,l}$ is also connected to the $\lfloor j/n \rfloor$th ($\lfloor l/n \rfloor$th) row ToR switch in the kth row. Thus, we can choose $a_{i,l}$ as the *relay server* for forwarding packets from $a_{i,j}$ to $a_{k,l}$. The shortest path consists of two segments: the path from $a_{i,j}$ to $a_{i,l}$, and the shortest path from $a_{i,l}$ to $a_{k,l}$ in the lth column. Notice that, we can also choose $a_{k,j}$ as the relay server. If $\lfloor j/n \rfloor \neq \lfloor l/n \rfloor$, according to *Characteristic 1*, we need servers in a type B row to relay packets from $a_{i,j}$ and $a_{k,l}$. We choose a server that connects to the same column ToR switch as of $a_{i,j}$'s, and that belongs to a type B row as the *first relay server*; we denote the server as $a_{r^*,j}$. Notice that, we can always succeed in choosing $a_{r^*,j}$ according to

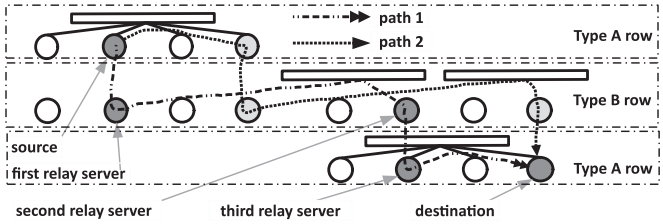


Fig. 5. Shortest path for the case when the source and destination both belong to type A rows.

Characteristic 2. The *second relay server* is chosen as $a_{r^*,\lfloor l/n \rfloor n+j\%n}$, which connects to the same $(j\%n)$th row ToR switch in the r^*th row, as $a_{r^*,j}$ does. The *third relay server* is chosen as $a_{k,\lfloor l/n \rfloor n+j\%n}$, which connects to the same $\lfloor l/n \rfloor$th row ToR switch in the kth row, as $a_{k,l}$ does. The shortest path from $a_{r^*,\lfloor l/n \rfloor n+j\%n}$ to $a_{k,\lfloor l/n \rfloor n+j\%n}$ is within the $(\lfloor l/n \rfloor n+j\%n)$th column, and requires no further explanation. Then, the shortest path from $a_{i,j}$ to $a_{k,l}$ consists of at most four segments: the path from $a_{i,j}$ to the first relay server, which includes one switch; the path from the first relay server to the second relay server, which includes one switch; the shortest path from the second relay server to the third relay server, which includes at most three switches; and the path from the third relay server to $a_{k,l}$, which includes one switch. Path 1 in Fig. 5 is the shortest path if we choose the first relay server in the same column as of $a_{i,j}$'s. We can actually choose the first relay server in the same row as of $a_{i,j}$'s; a simpler way to see this is that we can swap the roles of the source and destination, and still choose the first relay server in the same column as of the new source's. Path 2 in Fig. 5 is an alternative shortest path from the source to the destination.

Case 4. The source $a_{i,j}$ and destination $a_{k,l}$ both belong to type B rows. The shortest path construction is similar to the case when they both belong to type A rows. We omit further discussions here.

4.3 FRectangle Basic Properties

Property 7. *In an FRectangle(n), the number of servers is $N_v = n^4/2$, and the number of switches is $N_w = 2n^3$.*

Proof. In an FRectangle(n), there are $n^2/2$ rows and n^2 columns of servers. Thus, N_v=$n^4/2$. In each column, there are $3n/2$ switches; in each row, there are n switches. Thus, N_w=$3n/2\times n^2+n\times n^2/2$=$2n^3$. □

Property 8. *FRectangle(n) has a diameter of d=$6d_w$+$4d_v$.*

Proof. According to the shortest path routing scheme in FRectangle, the maximum length of the shortest path is achieved in the case when the source and destination belong to rows of the same type. In this case, the shortest path consists of at most four segments: the paths from the source to the first relay server, from the first relay server to the second relay server, from the second relay server to the third relay server, and from the third relay server to the destination. The first, second, and the fourth segment each consist of only one switch. The third segment is a shortest path between two servers in the same column, and thus consists of at most three switches. According to Eqs. (1) and (2), the diameter of FRectangle is d=$3d_w$+$3d_w$+$(3+1)d_v$=$6d_w$+$4d_v$. □

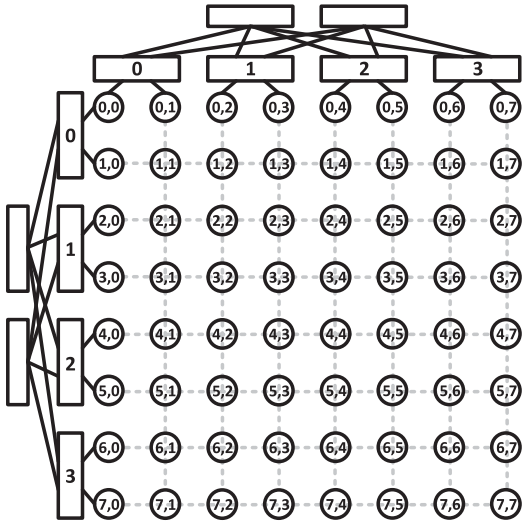


Fig. 6. FSquare(4). We consistently use rectangles to represent switches and circles to represent servers, respectively.

Property 9. *The bisection bandwidth of an FRectangle(n) is $B{=}N_v/4$.*

Illustration. Cutting a column of FRectangle into two halves requires removing $n^2/4$ links, while there are n^2 columns. Thus, cutting through columns requires removing $n^4/4$ links. However, cutting rows is different. Actually, if we consider a single row each time, no links need to be removed, since servers in each row are not fully connected, and they are already partitioned into two equal halves. Take a look at any type A row, and it is not difficult to find out. If we consider a type A row and a type B row together, we can see these two rows are connected through columns. An intuitive way to cut rows is to cut through the middle. For a *pair* of two rows consisting of one type A row and one type B row, the links to be removed consist of only links removed in the type B row, which is $n^2/2$, since each server in the first half of a type B row has an exclusive path to a server in the second half of the row. Notice that, we have $n^2/4$ type A rows and $n^2/4$ type B rows; thus, we have $n^2/4$ such two-row pairs. Thus, cutting FRectangle into two halves through the rows requires removing $n^2/2{\times}n^2/4{=}n^4/8$ links. The bisection bandwidth is $B{=}\min\{n^4/4, n^4/8\}{=}N_v/4$.

Property 10. *The DCN power consumption per server of an FRectangle(n) is $p_V{=}4p_w/n + 2p_{nic} + p_{fwd}$.*

Proof. The switch-number to server-number ratio in an FRectangle(n) is $N_w/N_v = 2n^3/(n^4/2) = 4/n$. In an FRectangle, each server uses two NIC ports, and servers may be involved in relaying packets. □

Property 11. *The number of server-disjoint parallel paths between two servers that belong to different row types, with path length $n_w d_w + (n_v+1)d_v$, where $d_v \le 2$, is 1.*

Illustration. When the source and destination belong to different row types, i.e., as in *Case 1* in Section 4.2, the column number $c^* = \lfloor j/n \rfloor n + (l\%n)$ is uniquely determined. Also, we can easily tell that, the number of servers (excluding the source and destination) in the path is 2, i.e, a_{i,c^*} and a_{k,c^*}.

Property 12. *The number of server-disjoint parallel paths between two servers that belong to the same row type, with path length $n_w d_w{+}(n_v{+}1)d_v$, where $d_v \le 3$, is at most $n^2/2$.*

Illustration. When the source and destination belong to the same row types, as in *Case 3* of Section 4.2, we need two other servers in a different row type to relay the packet. Since we have a total of $n^2/2$ rows, where half of the rows are type A rows, and half of the rows are type B rows, we have $n^2/4$ choices from which to choose a row with a different row type. Given the relay row, we still have two choices to construct such a path, as shown in Fig. 5. Thus, we have $n^2/2$ such paths. For each of these paths, the number of servers in the path (excluding the source and destination) is at most 3.

5 FSQUARE

5.1 FSquare Construction

We now introduce our FSquare architecture. Each column and each row of FSquare forms a basic building block, the cluster. We denote the server located at the ith row and the jth column by $a_{i,j}$ $(0{\le}i,j{\le}n^2/2{-}1)$. We number the ToR switches sequentially, such that $a_{i,j}$'s row ToR switch is the $\lfloor j/(n/2) \rfloor$th ToR switch in the ith row, and that $a_{i,j}$'s column ToR switch is the $\lfloor i/(n/2) \rfloor$th ToR switch in the jth column. An FSquare(4) is shown in Fig. 6, where we only draw the zeroth row and the zeroth column; other rows and columns are represented by grey dashed lines.

5.2 Routing in FSquare

We consider shortest path routing in FSquare(n). Denote the source and destination servers as $a_{i,j}$ and $a_{k,l}$ $(0{\le}i,j,k,l{\le}n^2/2{-}1)$, respectively. If the source and destination servers are in the same row, or in the same column, the shortest path is within the local row cluster or column cluster, and requires no further explanation. If $i{\neq}k$ and $j{\neq}l$, we can choose one from two *relay servers*: $a_{i,l}$ and $a_{k,j}$. The shortest path from source to destination consists of the shortest path from the source to the relay server, and the shortest path from the relay server to the destination. The two choices mean that we can traverse along the row first or along the column first.

5.3 FSquare Basic Properties

Property 13. *In an FSquare(n), the number of servers is $N_v = n^4/4$, and the number of switches is $N_w = 3n^3/2$.*

Proof. The number of servers in each row and in each column is $n^2/2$; and the number of switches in each row and in each column is $n{+}n/2{=}3n/2$. The architecture has $n^2/2$ rows and $n^2/2$ columns. Thus, $N_v{=}n^2/2{\times}n^2/2{=}n^4/4$; and $N_w{=}3n/2{\times}n^2/2{\times}2{=}3n^3/2$. □

Property 14. *FSquare(n) has a diameter of $d{=}6d_w{+}2d_v$.*

Proof. Obviously, the longest shortest path in an FSquare is between two servers that are neither in the same row nor in the same column. We consider two servers, $a_{i,j}$ and $a_{k,l}$, where $i \neq k$ and $j \neq l$. A shortest path from $a_{i,j}$ to $a_{k,l}$ can be the shortest path from $a_{i,j}$ to $a_{i,l}$ plus the shortest path from $a_{i,l}$ to $a_{k,l}$. Though $a_{i,j}$ and $a_{k,l}$ may connect to the

same switch, in the worst case, the shortest path from $a_{i,j}$ to $a_{i,l}$ consists of three switches. Similarly, in the worst case, the shortest path from $a_{i,l}$ to $a_{k,l}$ also consists of three switches. According to Eqs. (1) and (2), the diameter of an FSquare(n) is $d=6d_w+2d_v$. □

Property 15. *The bisection bandwidth of an FSquare(n) is $B=N_v/2$.*

Illustration. Since FSquare(n) is highly symmetric, we can cut the architecture into two equal halves through either all the rows or all the columns. Without loss of generality, we choose to cut through all the rows. We first consider cutting one row. Recall that there are $n^2/2$ servers in each row. The first half $(n^2/4)$ of servers can have an exclusive path to another server in the second half. Thus, cutting one row of servers into two equal halves requires removing $n^2/4$ links. Notice that there are $n^2/2$ rows in total; to cut the whole architecture into two equal halves, we need to remove $n^2/4 \times n^2/2 = n^4/8$ links. Thus, the bisection bandwidth of an FSquare(n) is $B = n^4/8 = N_v/2$.

Property 16. *The DCN power consumption per server of an FSquare(n) is $p_V=6p_w/n + 2p_{nic} + p_{fwd}$.*

Proof. The switch-number to server-number ratio in an FSquare(n) is $N_w/N_v=(3n^3/3)/(n^4/4)=6/n$; in an FSquare, each server uses two NIC ports, and servers may be involved in forwarding packets for other servers. □

Property 17. *The number of server-disjoint parallel paths between two servers that are not in the same row and not in the same column, with path length $n_w d_w + (n_v + 1)d_v$, where $d_v \leq 1$, is 2.*

Illustration. The shortest path from the source to the destination can be row first or column first.

Property 18. *The number of server-disjoint parallel paths between two servers that are not in the same row and not in the same column, with path length $n_w d_w + (n_v + 1)d_v$, where $d_v \leq 2$, is $n^2 - 2$.*

Illustration. We can construct a path with at most two relay servers as follows: randomly choose the first relay server from the same row, then from the first relay server, construct the remaining path by column-first shortest path routing; or, randomly choose the first relay server from the same row, then from the first relay server, construct the remaining path by row-first shortest path routing. It is easy to see that such paths are server-disjoint. Since we have $n^2/2 - 1$ choices in choosing the first relay server in the same row, and $n^2/2 - 1$ choices in choosing the first relay server in the same column, we have $n^2 - 2$ such paths.

6 Related Existing Works

DCN architecture design is an active research area [28], [29], [30]. In this section, we survey important existing DCN architectures that are classified as switch-centric architectures or server-centric architectures.

Typical switch-centric architectures include folded-Clos [16], Fat-Tree [8], Flattened Butterfly [7], and HyperX [16]. We denote a folded-Clos DCN architecture with l levels of n-port switches with by FDCL(n, l). The switch-number to server-number ratio in an FDCL(n, l) is $N_w/N_v=(2l-1)/n$. Fat-Tree is actually a folded-Clos with three levels, i.e., FDCL$(n, 3)$. In a Flattened Butterfly (FBFLY), switches form a generalized hypercube [31]. Then, each switch is connected to a set of c servers. An FBFLY with k dimensions and r switches along each dimension is denoted by FBFLY(r, k, c). The switch-number to server-number ratio is $N_w/N_v=1/c$. If the numbers of switches in each dimension are different in an FBFLY, it becomes the HyperX architecture.

Typical server-centric architectures include BCube [9], SWCube [11], DPillar [10], DCell [12], and FiConn [14]. In a BCube(n, k) the switch-number to server-number ratio is $(k+1)/n$ and its diameter is $(k+1)(d_w+d_v)$. It uses $k+1$ NIC ports on all the servers; its DCN power consumption per server is $p_V=(k+1)p_w/n+(k+1)p_{nic}+p_{fwd}$. The diameter of SWCube$(r, k)$ is $d=(k+1)(d_w+d_v)$. The diameter of DPillar(n, k) is $d=(k+\lfloor k/2 \rfloor)(d_w+d_v)$. The switch-number to server-number ratios of SWCube and DPillar are both $2/n$, and they both use two NIC ports on all the servers; thus, their DCN power consumption per server values are both $p_V=2p_w/n+2p_{nic}+p_{fwd}$. For DCell and FiConn, their switch-number to server-number ratios are both $1/n$. DCell(n, k) uses $k+1$ NIC ports on each server; in FiConn(n, k), the average number of NICs used on a server is $2-1/2^k$. The diameters of DCell$(n, 2)$ and FiConn$(n, 2)$ are both $d=4d_w+7d_v$. In DCell$(n, 2)$, $p_V=p_w/n+3p_{nic}+p_{fwd}$. In FiConn$(n, 2)$, $p_V=p_w/n+7p_{nic}/4+3p_{fwd}/4$.

7 Comparisons of Various DCN Architectures

7.1 On Comparison of Diameter and DCN Power Consumption Per Server

We compare various DCN architectures, constructed by the same homogenous servers and switches, with comparable numbers of servers. For architectures using 24 and 48-port switches, basic quantitative comparisons are presented in Table 1. Typical data centers have tens of thousands, or hundreds of thousands of servers, and the world's largest data centers can achieve one or two million. The numbers of servers in the table meet the needs of practical data centers.

Switch-centric architectures usually have a small diameter and a large bisection bandwidth. However their switch-number to server-number ratio is usually large, resulting in a large DCN power consumption. BCube also has a large bisection bandwidth; but it needs to use four levels of switches to reach a comparable DCN, and consequently four NIC ports on all the servers; this results in a large DCN power consumption. Other server-centric architectures, such as SWCube, DPillar, DCell and FiConn, use much fewer switches, though a small number of extra NIC ports are required on servers; their power consumption is lower than switch-centric architectures and BCube. However, they rely heavily on servers for packet forwarding; even the maximum shortest paths contain a considerable number of servers (usually $\geq$5 for them to scale to a comparable network size), which results in large end-to-end delays; besides, their bisection bandwidths are much lower.

We regard the delay on a switch, d_w as 1, and vary the delay on a server, d_v from 1 to 5. Fig. 7a shows the diameters

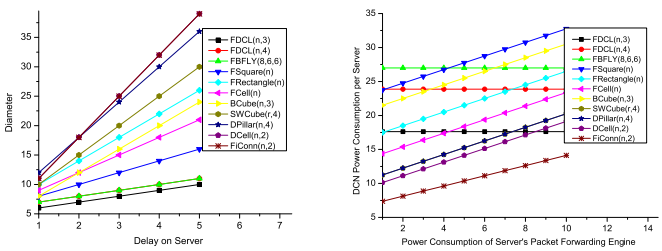
TABLE 1
Comparison of Various DCN Architectures

	$N_v(n{=}24)$	$N_v(n{=}48)$	N_w/N_v	d	B	p_V
FDCL$(n,3)$	3,456	27,648	$5/n$	$5d_w{+}d_v$	$N_v/2$	$5p_w/n+p_{nic}$
FDCL$(n,4)$	41,472	663,552	$7/n$	$7d_w{+}d_v$	$N_v/2$	$7p_w/n+p_{nic}$
FBFLY$(4,7,3)$	49,125	—	$8/24$	$8d_w{+}d_v$	$N_v/3$	$8p_w/24+p_{nic}$
FBFLY$(8,6,6)$	—	1,572,864	$8/48$	$7d_w{+}d_v$	$N_v/3$	$8p_w/48+p_{nic}$
FSquare(n)	82,944	1,327,104	$6/n$	$6d_w{+}2d_v$	$N_v/2$	$6p_w/n+2p_{nic}+p_{fwd}$
FRectangle(n)	165,888	2,654,208	$4/n$	$6d_w{+}4d_v$	$N_v/4$	$4p_w/n+2p_{nic}+p_{fwd}$
FCell(n)	83,232	1,328,256	$3/n$	$6d_w{+}3d_v$	$N_v/4$	$3p_w/n+2p_{nic}+p_{fwd}$
BCube$(n,3)$	331,776	5,308,416	$4/n$	$4d_w{+}4d_v$	$N_v/2$	$4p_w/n+4p_{nic}+p_{fwd}$
SWCube$(r,4)$	28,812	685,464	$2/n$	$5d_w{+}5d_v$	$(N_v/8)\times r/(r-1)$	$2p_w/n+2p_{nic}+p_{fwd}$
DPillar$(n,4)$	82,944	1,327,104	$2/n$	$6d_w{+}6d_v$	$N_v/4$	$2p_w/n+2p_{nic}+p_{fwd}$
DCell$(n,2)$	360,600	5,534,256	$1/n$	$4d_w{+}7d_v$	$> N_v/(4\log_n N_v)$	$p_w/n+3p_{nic}+p_{fwd}$
FiConn$(n,2)$	24,648	361,200	$1/n$	$4d_w{+}7d_v$	$> N_v/16$	$p_w/n+7p_{nic}/4+3p_{fwd}/4$

of various DCN architectures. FCell has a lower diameter than all server-centric architectures when $d_v > 2$, which reflects most practical situations. For switches with $n{=}48$ 1 Gbps ports and 1 Gbps NIC ports, we set $p_w{=}150$ and $p_{nic}{=}2$. We vary p_{fwd} from 1 to 10. Fig. 7b shows the DCN power consumption per server of various architectures. When $p_{fwd} \leq 4$, which also reflects most practical situations, FCell consumes less power than switch-centric architectures and BCube$(n,3)$. Also, FCell has a satisfiable bisection bandwidth of $N_v/4$. We can see that FCell reflects a tradeoff design between network performances and DCN power consumption.

7.2 On Comparison of Diameter and Average Shortest Path Length of FCell, FRectangle, and FSquare

Notice that, from FCell to FRectangle and FSquare, the number of switches per server in the architecture are increasing (from $3/n$, to $4/n$, and to $6/n$). However, the diameters of the three architectures are not strictly reducing. FSquare architecture uses the greatest number of switches among the three and has the lowest diameter, $6d_w + 2d_v$. FRectangle and FCell use fewer switches; thus, their diameters are larger. However, the diameter of FRectangle is greater than FCell, while the number of switches per server is greater than that of FCell. It seems that, they are not following the expected trend. We discover that, though FCell has a lower diameter than FRectangle, a majority of the server pairs have a shortest path whose length is equal to the diameter. Although, FRectangle has a larger diameter, only about half of the server pairs have a shortest path whose length is equal to the diameter, for the remaining pairs of servers, we have much shorter shortest path length than the diameter.



(a) Diameters vs. d_v. (b) p_V vs. p_{fwd}.

Fig. 7. Comparison of various architectures ($n = 48$). Notice that, some of the lines are overlapped.

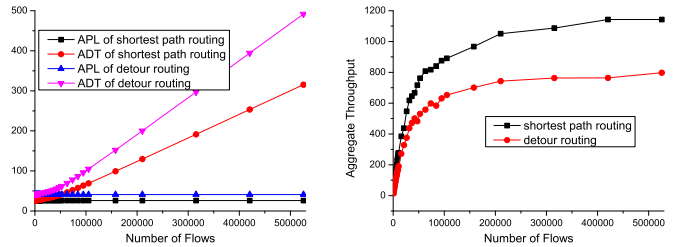
To verify this observation, we calculate the average path length (APL) of the three architectures and compare them. The results are shown in Table 2.

7.3 Additional Discussions on the Three Proposed Architectures

Again, in this paper we are not to provide the best architecture in any sense; instead, our purpose is to present example tradeoff designs for DCN architectures. FCell uses the least number of switches among the three. In FCell, for each pair of two clusters, there is only one pair of servers from the two clusters, respectively, that are directly connected to each other; this indicates a severe bottleneck on the single link, and limited bandwidth between two clusters, although detour routing can help to mitigate the problem to some degree. Due to the limited number of shortest paths between two servers in FCell, we can expect FCell to have poor adaptive routing performances. Thus, FCell is only good for scenarios where the inter-cluster communication is minimal. Typical application cases for FCell include intermittent backup from one cluster to another cluster, and separated clusters for independent groups/organizations that have minimal interactions, etc. FSquare uses the greatest number of switches among the three; this provides abundant connections between servers in the architecture. Thus, FSquare has good performances even when inter-cluster communications are intensive. Since FSquare have abundant shortest paths between two servers, we can expect FSquare to have even better performances when adaptive routing is taken into consideration. Typical application cases for FSquare can be distributed file systems, and big data processing frameworks such as Hadoop and Spark, etc. FRectangle can be used for cases where the inter-cluster

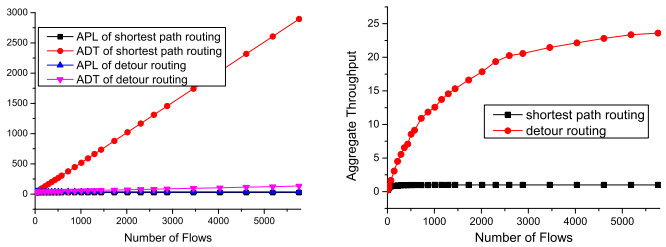
TABLE 2
Average Path Length and Diameter Comparison

	d_v	1	2	3	4	5
FCell	d	9.000	12.000	15.000	18.000	21.000
	ASPL	8.547	11.492	14.438	17.384	20.329
FRectangle	d	10.000	14.000	18.000	22.000	26.000
	ASPL	8.492	11.821	15.150	18.479	21.808
FSquare	d	8.000	10.000	12.000	14.000	16.000
	ASPL	7.613	9.585	11.558	13.530	15.503



(a) APL and ADT. (b) AggTh.

Fig. 8. APL, ADT and AggTh versus no. of flows (random traffic).



(a) APL and ADT. (b) AggTh.

Fig. 9. APL, ADT and AggTh versus no. of flows (bursty traffic).

communications are light to modest. Typical application cases for FRectangle include three-tier web applications, and applications that use light-weight Message Passing Interface (MPI) communications etc.

8 Evaluations

We develop a proprietary simulator to conduct routing simulations in FCell, FRectangle, and FSquare. Modern switches, even low-end ones may have very complex designs inside. We do not intend to model all the details, and just build a basic model for store-and-forward switches. Both switches and servers are assumed to have 1 Gbps full duplex ports. We consider single-packet flows and a fixed packet size. Thus, we have a fixed transmission delay, which is considered as one unit of time, i.e., $d_{w,t}$=$d_{v,t}$=1. This time unit has a typical value around 2 μs. The switch's and the server's processing delays, $d_{w,p}$ and $d_{v,p}$ are normalized by this time unit. We set the switch's processing delay $d_{w,p}$=1, and set the server's processing delay $d_{v,p}$=4. Queuing delay happens when multiple packets compete for the same output port (either on a switch or on a server) simultaneously.

All of the flows are generated and pushed to the network at the same time. We calculate the Aggregate Throughput (AggTh), the Average Path Length and the Average Delivery Time (ADT) of simulations. Aggregate throughput is defined as the average amount of data transmitted in one unit of time when all the flows are delivered to their destinations, i.e., the total data amount divided by the maximum delivery time among all flows. For APL, the path lengths are calculated based on our unified definition in (1).

8.1 Simulation for FCell

In simulations for FCell, we consider two traffic patterns: random and bursty traffic patterns [14], for both Shortest Path Routing and Detour Routing. In random traffic patterns, the source server and the destination server of each packet are randomly generated among all the servers. In bursty traffic patterns, servers in one cluster of FCell have a flow destined at other servers in another cluster. We choose the zeroth cluster and the first cluster as the source and destination clusters, respectively. In both traffic patterns, we can choose different numbers of flows to be generated, to reflect different traffic loads in the network.

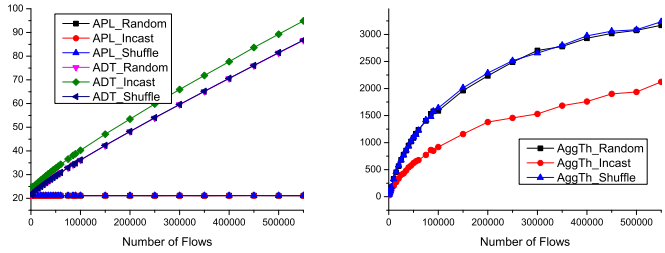
We conduct simulations on a complete FCell(12). The number of servers in one cluster is $N_v^{cluster}$=$12^2/2$=72. The total number of servers is N_v=5,256. For random traffic, we vary the number of flows from 657 to 525,600; for bursty traffic, we vary the number of flows from 9 to 5,760. Step sizes are different in different ranges.

Fig. 8 shows the performances of the two routing schemes under random traffic. As we can see, APLs of SRouting and DRouting remain constant, because APLs do not depend on the number of flows. APL of DRouting is greater than that of SRouting, because DRouting does not choose the shortest path. When the number of flows is small, the aggregate throughput increases almost linearly with the number of flows, and ADTs are very close to APLs; this is because the network has a very light traffic load and the main end-to-end delays come from processing delays and transmission delays, instead of queuing delays. When the number of flows is large, the increase of aggregate throughput becomes slower and slower and ADTs of both SRouting and DRouting increase almost linearly; this is because the network tends to be saturated and queuing delays become an important part of end-to-end delays. Notice that the upper bound of the aggregate throughput is the bisection bandwidth $B \approx N_v/4 = 1,314$. When the number of flows is 525,600, SRouting achieves an aggregate throughput of 1,142.6, which is 87.96 percent of the ideal maximum throughput. Thus, SRouting has good performances under random traffic. DRouting has a lower aggregate throughput because it has a larger maximum delivery time.

Fig. 9 shows the performances of the two routing schemes under bursty traffic. Though APL of DRouting is greater than that of SRouting, when the number of flows increases, ADT of DRouting is much smaller than that of SRouting. This is because DRouting experiences significantly less queuing delay by avoiding the congested link. When the number of flows increases, the aggregate throughput of SRouting is limited by the capacity of the congested link, 1, while the aggregate throughput of DRouting continues increasing significantly, because DRouting largely avoids the congested link and can use other links' capacities. Notice that, under bursty traffic, the aggregate throughput is also upper bounded by the sending and receiving rates of servers in the two clusters. If on average, only half of the servers are sending packets at each time unit, it indicates an upper bound on the maximum aggregate throughput of $N_v^{cluster}/2 = 36$. It takes some effort to calculate the true upper bound; we just want to show that this is the reason why the aggregate throughput of DRouting tends to approximate an upper bound around 23.5. We can see that DRouting helps both reducing ADTs and increasing the aggregate throughput under bursty traffic.

8.2 Simulation for FRectangle and FSquare

In simulations for FRectangle and FSquare, we consider their shortest path routing for three typical traffic patterns in data centers, referring to [32]. In *Random* traffic pattern, for each flow, the source and destination servers are



(a) APL and ADT. (b) AggTh..

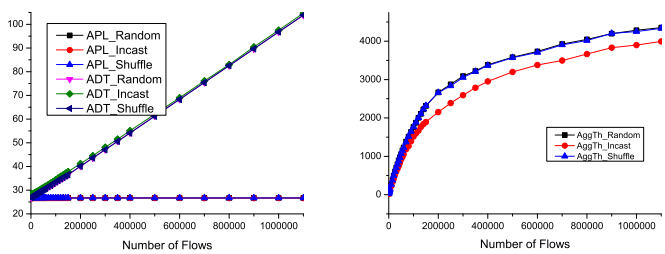
Fig. 10. APL, ADT and AggTh versus no. of flows in FSquare.

randomly chosen among all of the servers. In *Incast* traffic pattern, a server receives traffic flows from multiple random servers in the network; this traffic pattern simulates the shuffle stage of the widely-used MapReduce framework; we assume that a server receives flows from 10 other random servers. In rack *Shuffle* traffic pattern, servers in a rack send traffic flows to servers in several different racks; this simulates the traffic when the administrator is trying to balance the load between racks through VM migration; this traffic pattern is common in elastic data centers, where servers are turned off at off-peak hours; in our simulations, we assume that servers in a column rack send traffic flows to servers in other column racks.

We choose switch port number, n=12. In an FSquare(12), the number of servers is N_v=$(12^4)/4$=5,184. We vary the number of flows from 1,000 to 550,000. In an FRectangle(12), the number of servers is N_v=10,368. We vary the number of flows from 1,000 to 1,100,000.

Figs. 10 and 11 show the performances of FSquare and FRectangle in various traffic conditions, respectively. When the number of flows is small, the AggTh values under all traffic patterns increase almost linearly. When the number of flows is large, the increasing rates of the AggTh becomes smaller and smaller. This means that the network is becoming more and more congested. As the number of flows increases significantly and becomes congested, the ADTs in FSquare and FRectangle only increase linearly. We can see that both architectures can achieve satisfyingly large AggThs. Random traffic is expected to achieve the best performances in all cases, because it automatically balances the traffic among the network. Shuffle traffic achieves performances comparable to Random traffic. We can see that, both FSquare and FRectangle do not place extra bottlenecks on the Shuffle traffic. In the Incast traffic, a server received 10 flows from 10 different other servers. Thus, the server NIC ports themselves become the congested points, and the performances of the Incast traffic are always the worst among the three. We can see that the performances of the Incast traffic are quite close to those of Random and Shuffle in FRectangle. The reason is that, in FRectangle, the reduced row switches place greater bottlenecks for all traffic patterns. The influence of the Incast traffic's own congestion points is less significant.

We can see that, FSquare can achieve very good performances under various traffic conditions, while FRectangle's performances are worse than those of FSquare.



(a) APL and ADT. (b) AggTh.

Fig. 11. APL, ADT and AggTh versus no. of flows in FRectangle.

9 Conclusion

In this paper, we consider the tradeoffs in designing DCN architectures. We present a unified path length definition and a DCN power consumption model for general DCNs, to enable fair and meaningful comparisons among various DCNs. We introduce a novel category of DCN architectures: the dual-centric architectures. We propose three novel dual-centric DCN architectures: FCell, FRectangle, and FSquare. By comparing them with existing architectures and by investigating themselves, we show that the three architectures have various nice properties for practical data centers, and provide flexible choices in designing DCN architectures.

Future works can be cast in, but are not limited to, the following directions: 1.) designing efficient and/or adaptive routing schemes for the proposed architectures; 2.) exploring other possible dual-centric architectures that also have appealing properties; 3.) designing dual-centric architectures where each server uses more than two NIC ports; and 4.) exploring the limitations of the dual-centric design philosophy, and how to control and apply them in practical DCN designs.

Acknowledgments

This research was supported in part by US National Science Foundation grants CNS 1449860, CNS 1461932, CNS 1460971, CNS 1439672, CNS 1301774, ECCS 1231461, and NSFC grants 61520106005, 61521092.

References

[1] S. Sung, C. Youn, E. Kong, and J. Ryou, "A distributed mobile cloud computing model for secure big data," in *Proc. Int. Conf. Inf. Netw.*, 2016, pp. 312–316.
[2] W. Chang and J. Wu, "Reliability enhanced social crowdsourcing," in *Proc. IEEE Global Commun. Conf.*, 2015, pp. 1–6.
[3] W. Chang and J. Wu, "Progressive or conservative: Rationally allocate cooperative work in mobile social networks," *IEEE Trans. Parallel Distrib. Syst.*, vol. 26, no. 7, pp. 2020–2035, Jul. 2015.
[4] S. Ghemawat, H. Gobioff, and S.-T. Leung, "The Google file system," in *Proc. 19th ACM Symp. Operating Syst. Principles*, 2003, pp. 29–43.
[5] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," *Commun. ACM*, vol. 51, no. 1, pp. 107–113, Jan. 2008.
[6] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly, "Dryad: Distributed data-parallel programs from sequential building blocks," in *Proc. 2nd ACM SIGOPS/EuroSys Eur. Conf. Comput. Syst.*, 2007, pp. 59–72.
[7] D. Abts, M. R. Marty, P. M. Wells, P. Klausler, and H. Liu, "Energy proportional datacenter networks," in *Proc. 37th Annu. Int. Symp. Comput. Archit.*, 2010, pp. 338–347.
[8] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," in *Proc. ACM SIGCOMM Conf. Data Commun.*, 2008, pp. 63–74.
[9] C. Guo, et al., "BCube: A high performance, server-centric network architecture for modular data centers," in *Proc. ACM SIGCOMM Conf. Data Commun.*, 2009, pp. 63–74.
[10] Y. Liao, D. Yin, and L. Gao, "DPillar: Scalable dual-port server interconnection for data center networks," in *Proc. 19th Int. Conf. Comput. Commun. Netw.*, 2010, pp. 1–6.

[11] D. Li and J. Wu, "On the design and analysis of data center network architectures for interconnecting dual-port servers," in *Proc. IEEE INFOCOM*, 2014, pp. 1851–1859.
[12] C. Guo, H. Wu, K. Tan, L. Shi, Y. Zhang, and S. Lu, "DCell: A scalable and fault-tolerant network structure for data centers," in *Proc. ACM SIGCOMM Conf. Data Commun.*, 2008, pp. 75–86.
[13] D. Guo, T. Chen, D. Li, M. Li, Y. Liu, and G. Chen, "Expandable and cost-effective network structures for data centers using dual-port servers," *IEEE Trans. Comput.*, vol. 62, no. 7, pp. 1303–1317, Jul. 2013.
[14] D. Li, C. Guo, H. Wu, K. Tan, Y. Zhang, and S. Lu, "FiConn: Using backup port for server interconnection in data centers," in *Proc. IEEE INFOCOM*, 2009, pp. 2276–2285.
[15] Y. Zhang and N. Ansari, "On architecture design, congestion notification, TCP incast and power consumption in data centers," *IEEE Commun. Surveys Tuts.*, vol. 15, no. 1, pp. 39–64, Jan.–Mar. 2013.
[16] J. H. Ahn, N. Binkert, A. Davis, M. McLaren, and R. S. Schreiber, "HyperX: Topology, routing, and packaging of efficient large-scale networks," in *Proc. Conf. High Performance Comput. Netw. Storage Anal.*, 2009, pp. 41:1–41:11.
[17] A. Singla, C.-Y. Hong, L. Popa, and P. B. Godfrey, "Jellyfish: Networking data centers randomly," in *Proc. 9th USENIX Conf. Networked Syst. Des. Implementation*, 2012, pp. 17–17.
[18] L. Gyarmati and T. A. Trinh, "Scafida: A scale-free network inspired data center architecture," *SIGCOMM Comput. Commun. Rev.*, vol. 40, no. 5, pp. 4–12, Oct. 2010.
[19] J.-Y. Shin, B. Wong, and E. G. Sirer, "Small-world datacenters," in *Proc. 2nd ACM Symp. Cloud Comput.*, 2011, Art. no. 2.
[20] Y. Liu, J. K. Muppala, M. Veeraraghavan, D. Lin, and M. Hamdi, *Data Center Networks: Topologies, Architectures and Fault-Tolerance Characteristics*. Berlin, Germany: Springer, 2013.
[21] L. Gyarmati and T. A. Trinh, "How can architecture help to reduce energy consumption in data center networking?" in *Proc. 1st Int. Conf. Energy-Efficient Comput. Netw.*, 2010, pp. 183–186.
[22] J. F. Kurose and K. W. Ross, *Computer Networking: A Top-Down Approach*, 5th ed. Reading, MA, USA: Addison-Wesley, 2009.
[23] A. Greenberg and D. A. Maltz, "What goes into a data center-sigmetrics 2009 tutorial," 2009. [Online]. Available: http://research.microsoft.com/apps/pubs/default.aspx?id=81782
[24] G. Lu, et al., "ServerSwitch: A programmable and high performance platform for data center networks," in *Proc. 8th USENIX Conf. Networked Syst. Des. Implementation*, 2011, pp. 15–28.
[25] L. Popa, S. Ratnasamy, G. Iannaccone, A. Krishnamurthy, and I. Stoica, "A cost comparison of datacenter network architectures," in *Proc. 6th Int. Conf. Emerg. Netw. Experiments Technol.*, 2010, pp. 16:1–16:12.
[26] Cisco nexus 2000 series fabric extenders data sheet, 2016. [Online]. Available: http://www.cisco.com/c/en/us/products/collateral/switches/nexus-2000-series-fabric-extenders/data_ sheet_c78–507093.html
[27] Intel gigabit ET, ET2, and EF multi-port server adapters, 2016. [Online]. Available: http://www.intel.com/content/dam/doc/product-brief/gigabit-et-et2-ef-multi-port-server-adapters-brief.pdf
[28] Z. Guo and Y. Yang, "On nonblocking multicast fat-tree data center networks with server redundancy," *IEEE Trans. Comput.*, vol. 64, no. 4, pp. 1058–1073, Apr. 2015.
[29] S. Xu, B. Fu, M. Chen, and L. Zhang, "Flyover: A cost-efficient and scale-out data center network architecture," in *Proc. 24th Int. Conf. Comput. Commun. Netw.*, Aug. 2015, pp. 1–8.
[30] G. Qu, Z. Fang, J. Zhang, and S.-Q. Zheng, "Switch-centric data center network structures based on hypergraphs and combinatorial block designs," *IEEE Trans. Parallel Distrib. Syst.*, vol. 26, no. 4, pp. 1154–1164, Apr. 2015.
[31] L. Bhuyan and D. Agrawal, "Generalized hypercube and hyperbus structures for a computer network," *IEEE Trans. Comput.*, vol. C-33, no. 4, pp. 323–333, Apr. 1984.
[32] Y. J. Liu, P. X. Gao, B. Wong, and S. Keshav, "Quartz: A new design element for low-latency DCNs," in *Proc. ACM Conf. SIGCOMM*, 2014, pp. 283–294.

Dawei Li received the bachelor's degree for advanced class from the Department of Electronics and Information Engineering, Huazhong University of Science and Technology, Wuhan, Hubei, People's Republic of China. He is working toward the PhD degree in the Department of Computer and Information Sciences, Temple University, since September 2011. His research interests include energy-aware task scheduling on multicores/multiprocessors, network-on-chips, and data center networks.

Jie Wu is the associate vice provost of international affairs with Temple University. He also serves as director of Center for Networked Computing and Laura H. Carnell professor in the Department of Computer and Information Sciences. Prior to joining Temple University, he was a program director of National Science Foundation and was a distinguished professor with Florida Atlantic University. His current research interests include mobile computing and wireless networks, routing protocols, cloud and green computing, network trust and security, and social network applications. He regularly publishes in scholarly journals, conference proceedings, and books. He serves on several editorial boards, including the *IEEE Transactions on Service Computing* and the *Journal of Parallel and Distributed Computing*. He was general cochair/chair for IEEE MASS 2006, IEEE IPDPS 2008, IEEE ICDCS 2013, and ACM MobiHoc 2014, as well as program co-chair of IEEE INFOCOM 2011 and CCF CNCC 2013. He was an IEEE Computer Society distinguished visitor, ACM distinguished speaker, and chair of the *IEEE Technical Committee on Distributed Processing*. He received the 2011 China Computer Federation (CCF) Overseas Outstanding Achievement Award. He is a CCF Distinguished Speaker and a fellow of the IEEE.

Zhiyong Liu received the PhD degree in computer science from the Institute of Computing Technology (ICT), Chinese Academy of Sciences (CAS), China. He is the chair professor in ICT, CAS. His current research interests include high performance algorithm and architecture, parallel processing, and bioinformatics.

Fa Zhang received the PhD degree in computer science from the Institute of Computing Technology (ICT), Chinese Academy of Sciences (CAS), China. He is an associate professor in ICT, CAS. His current research interests include bioinformatics, biomedical image processing, and high performance computing.