Dual-Centric Data Center Network Architectures

Dawei Li, Jie Wu
Department of Computer and Information Sciences,
Temple University, Philadelphia, USA
{dawei.li, jiewu}@temple.edu

Zhiyong Liu, Fa Zhang
Institute of Computing Technology,
Chinese Academy of Sciences, Beijing, P. R. China
{zyliu, zhangfa}@ict.ac.cn

***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. We propose two typical dual-centric DCN architectures: FSquare and FRectangle, both of which are based on the folded Clos topology. 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 performances; thus, FRectangle has a larger diameter and a smaller bisection bandwidth. By investigating FSquare and FRectangle, and by comparing them with existing architectures, we demonstrate that, these two 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 choices in designing DCN architectures.**

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

I. Introduction

Nowadays, data centers have become important infrastructures to support various cloud computing services, varying from web search, email, video streaming, and social networking, to distributed file systems such as GFS [1], and distributed data processing engines, such as MapReduce [2]. The Data Center Networks (DCNs) specify how servers and various other components are interconnected in data centers. DCNs have a significant influence on the quality of the services that data centers provide to various applications.

Existing DCN architectures have been classified into two categories: switch-centric architectures and server-centric architectures [3]. In switch-centric designs [4]–[6], routing intelligence is placed on switches; a server usually uses only one NIC port, and is not involved in forwarding, or more accurately, relaying packets for other servers. In server-centric designs [7]–[12], 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 nodes and packet forwarding nodes. Switch-centric architectures enjoy the fast switching capability of switches, but switches are less programmable than servers; thus, complex network control and management have to resort to high-end switches that are not only expansive, but also power hungry. Server-centric architectures enjoy the high programmability of servers, but servers usually have larger processing delays than do switches. Can we achieve the advantages of and mitigate the drawbacks of both categories? Motivated by this aspect, 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. By involving both servers and switches in routing, dual-centric DCNs can enjoy both the fast switching capability of switches and the high programmability of servers. With this mixed design philosophy, dual-centric architectures are expected to provide us with more flexible choices in designing DCN architectures, and to enjoy various other advantages of both server-centric and switch-centric designs. As switches are becoming more programmable via software defined networking [13], and servers are tending to utilize specialized hardware for packet forwarding [14], both switches and servers will carry both packet forwarding capability and routing intelligence. Thus, the proposed dual-centric design philosophy will certainly become a potential candidate in future DCN architecture designs.

We propose two typical dual-centric DCN architectures, named FSquare and FRectangle. Both of them are based on the folded Clos topology [6]. FSquare is a two-dimensional architecture; each dimension is a two-level folded Clos topology. FRectanle is also a two-dimensional architecture; in one dimension, it has a folded Clos topology, while in the other dimension, it applies sparse interconnection rules to reduce the number of switches. All servers in FSquare and FRectangle can help in forwarding packets for other servers. By comparing them with various existing DCN architecture, and by investigating themselves, we show that the two newly proposed architectures have various appealing properties.

Performances and power efficiency are two crucial considerations in designing DCNs. Two important performance metrics for a DCN architecture are end-to-end delays in the DCN and the bisection bandwidth of the DCN. End-to-end delays translate directly or indirectly to applications' response times in various situations. Bisection bandwidth provides key information on the potential throughput that the network can provide, and also indicates the fault-tolerance capabilities. As servers in a data center are becoming more and more power efficient, the DCN tends to consume 50% of the total IT power

[5] of a data center; thus, the DCN power consumption has also become an important issue.

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, the path lengths are calculated differently by existing works. For switch-centric architectures, the length of a path is calculated as the number of links in the path [15], [16]; for server-centric architectures, the length is calculated as the number of servers (excluding the source and the destination) in the path between the two servers, plus 1 [7]–[12]. A diameter of 6 in Fat-Tree [4] means a totally different thing than a diameter of 6 in BCube [8]. However, a lot of works still compare these two different kinds of diameters [3], [7], [15], [16]. This somewhat confuses the understanding of the end-to-end delay in a general DCN. Motivated by this issue, we present a *unified path length* definition and, accordingly, a *unified diameter* definition for a general DCN: either a switch-centric one, a server-centric one, or a dual-centric one.

As can be seen, in a general DCN, a server may be involved in forwarding packets for other servers. In this case, the power consumption of the server's packet forwarding engine should also be included as part of the power consumption of the DCN. Based on this observation, we present a DCN power consumption model for general DCNs, which takes into consideration the power consumption of the server's packet forwarding engine.

Our main contributions in this paper are as flows:

- We introduce a new category of DCN architectures, i.e., the dual-centric DCN architectures, to complement the current DCN architecture classifications. To the best of our knowledge, we are the first to formally introduce this dual-centric design philosophy. We also propose two novel typical dual-centric architectures: FSquare and FRectangle.
- To enable fair and meaningful comparisons among various existing DCN architectures and our proposed dual-centric architectures, we propose a unified path length definition, and, accordingly, a unified diameter definition, for general DCNs. Also, to characterize the power efficiency of a general DCN, we propose a unified DCN power consumption model.
- By investigating the two proposed architectures and by comparing them with existing DCN, we show that dual-centric architectures can have appealing properties for practical DCN designs. Also, routing simulations are conducted for FSquare and FRectangle to justify their performances under various traffic patterns and loads.

The rest of the paper is organized as follows. Section II presents the unified path length, DCN diameter definitions, and DCN power consumption model. We describe our novel DCN architectures, FSquare and FRectangle, in Section III, where we also describe their routing schemes and basic properties. We further investigate properties of FSquare and FRectangle in Section IV. Supporting simulations are conducted in Section V. Additional discussions are provided in Section VI. Conclusions and future work directions are sketched in Section VII.

II. Preliminaries

To scale the data center to a large size, existing works have chosen to "scale out" the data center, which means using a large number of low-end Commodity Off The Shelf (COTS) devices to extend the data center, instead of using a small number of high-end expansive ones. For the ease of presentation, we assume that all the switches and servers are COTS ones, and 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. Another reason for not considering the queuing delay is that, under various control mechanisms, the queuing delay can be significantly reduced to minimal values; there have been various works on this [17], [18]. For simplicity, we 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 $2\mu s$ [19]. $d_{w,t} = S_{packet}/r_{bit}$, where S_{packet} is the size of the packet and r_{bit} is the data transmission rate. S_{packet} varies between 64 bytes and 1514 bytes. Given data transmission rate r_{bit} = 1Gbps, $d_{w,t}$ 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 is much higher than that on a switch, with a typical value of about $10\mu s$ [19]. Depending on CPU load and NIC configuration, this value varies significantly. In hardware-based forwarding, $d_{v,p}$ can be reduced to be close to the processing delay on a switch [14]. We do not delve into the detailed implementation of the packet forwarding schemes on servers. 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 1 to several times of d_w.

Network links have *propagation delay*, d_l, which can be calculated by dividing the length of the link (L_{link}) by the speed of the signal in the transmission medium: $d_l = L_{link}/(\eta c)$, where η is a constant around 0.7 and c is the speed of light in a 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 $10/(0.7\times3\times10^8)s = 0.048\mu s$. 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 involved in 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 switch-centric architectures. 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 server-centric architectures. Under this unified path length definition, we define the *diameter of a general DCN* as the maximum path length (based on Eq. (1)) of the shortest paths between all pairs of servers in the DCN:

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

where $\mathcal{P}$ is the set of shortest paths between all pairs of servers in the DCN.

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, either software-based or hardware-based, 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 [20] or the additional hardware's power consumption for hardware-based forwarding [14]), 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, $\alpha = 0$. For DCNs where servers are involved in forwarding packets for other servers, α depends on various factors; for simple and fair comparison, we can choose $\alpha{=}1$. A practical value of p_w for a switch with 48 1Gbps ports is about 150 Watts [21]; a practical value of p_{nic} for 1Gbps NIC port is 2 Watts [22]. As reported in [20], 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 5Watts if NIC ports are 10Gbps. The value for p_{fwd} will be lower if NIC ports are 1Gbps. In hardware-based forwarding, p_{fwd} may also have quite different values [14].

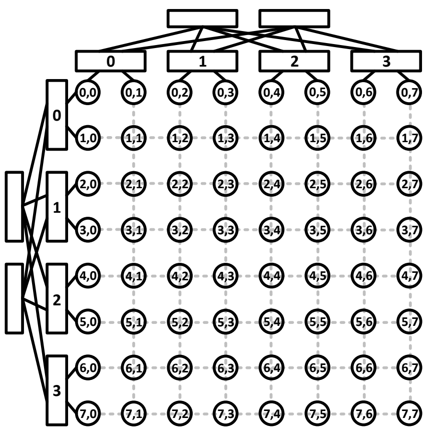


Fig. 1. FSquare(4). We consistently use rectangles to represent switches and circles to represent servers, respectively.

III. Dual-Centric DCNs: FSquare & FRectangle

A. FSquare Construction

An FSquare has two dimensions, which are called rows and columns, respectively. We denote an FSquare built from servers with 2 NIC ports and n-port switches by FSquare(n). Since switches in data centers usually have large switch port counts, in the rest of the paper, we assume that n is greater than 4 and is an even number. In each row and in each column, there are two levels of switches: $n/2$ level 2 switches and n level 1 switches; every level 2 switch is connected to every level 1 switch. In other words, the set of $n/2$ level 2 switches and the set of n level 1 switches form a complete bipartite graph. Then, there are $n/2$ ports remaining on each of the level 1 switches; we use these ports to connect $n/2$ servers. As a result, the switches and servers in each row and each column form a simple instance of the folded Clos [6] topology. We denote the server located at the ith row and the jth column by $a_{i,j}$ ($0{\le}i,j{\le}n^2/2{-}1$). A level 1 switch is also called a *Top of Rack (ToR) switch*. We can see that, each server in an FSquare connects to two ToR switches, one in the row and one in the column; we call them the *row ToR switch* and the *column ToR switch* of the server, respectively. 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. The level 2 switches are also called *row level 2 switches* or *column level 2 switches*, depending on whether they connect to row or column ToR switches. An FSquare(4) is shown in Fig. 1, where we only draw the zeroth row and the zeroth column; other rows and columns are represented by grey dash lines.

B. Routing in FSquare

We consider shortest path routing in FSquare(n). We denote the source and destination servers by $a_{i,j}$ and $a_{k,l}$ ($0 \leq i, j, k, l \leq n^2/2 - 1$), respectively. We first consider how the shortest path can be constructed, and then discuss how the routing scheme can be implemented.

If the source and destination servers are in the same row, i.e., $i = k$, then the shortest path is within in this row. If $\lfloor j/(n/2) \rfloor = \lfloor l/(n/2) \rfloor$, i.e., the source and destination servers are connected to the same row ToR switch, the the shortest path consists of one switch, i.e., the common row ToR switch; otherwise, the shortest path consists of three switches: the source server's row ToR switch, a randomly chosen row level 2 switch, and the destination server's row ToR switch. If the source and destination servers are in the same column, the shortest path can be constructed similarly.

If $i \neq k$ and $j \neq l$, we can choose one from two *intermediate servers*: $a_{i,l}$ and $a_{k,j}$. In other words, the first choice is to traverse along the row, and then along the column; the second choice is to first traverse along the column, and then along the row. We can randomly choose one from the two. After that, the shortest path within the row and the shortest path within the column will be the same as in the cases when $i = k$, and when $j = l$, respectively.

We let the source server determine the intermediate server in the shortest path, if there is any, and then push the servers in the shortest path (including the destination, while excluding the source) from the last one to the first one into a structure called *server_stack*, and label the intermediate server as a *fake* destination and label the real destination as a *true* destination. Then, it sends the packet to the corresponding ToR switch, using the server at the top of the *server_stack*, as the temporary destination. When another server receives the packet, it pops the next server in the *server_stack*, and checks whether itself is a fake or true destination. If it is a true destination, it consumes the packet; otherwise, it continues to send the packet to a corresponding ToR switch using the next server as the temporary destination. When a switch receives the packet, it can only see the temporary destination set by the previous sending server, either the source, or the intermediate server, and makes routing decision based on the temporary destination.

Notice that, when multiple intermediate servers are available, the source server can make a wiser decision on which intermediate server to use based related information along the row and/or column, instead of randomly choosing. Also, when a ToR switch needs to send a packet to a level 2 switch, it can make a wiser decision on which level 2 switch to send to, based on related information within the row/column.

Both servers and switches in FSquare have some degree of routing intelligence, which reflects the dual-centric design philosophy. Thus, both the fast switching capability of switches and the high programmability of servers can be utilized. Notice that, in a shortest path, at most one intermediate server is involved in relaying packets for other servers. This will not increase the end-to-end delay significantly.

C. FSquare Basic Properties

Property 1. *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, the number of servers is $N_v = n^2/2 \times n^2/2 = n^4/4$; and the number of switches is $N_w = 3n/2 \times n^2/2 \times 2 = 3n^3/2$. ■

Property 2. *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 not in the same row and are not 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 3 switches. Similarly, in the worst case, the shortest path from $a_{i,l}$ to $a_{k,l}$ also consists of 3 switches. According to Eq. (1) and Eq. (2), the diameter of an FSquare(n) is $d = 6d_w + 2d_v$. ■

We assume that all the links in a DCN have a unit bandwidth, 1. Then, 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 FSquare has the following property.

Property 3. *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 halve, 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 4. *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 2 NIC ports, and servers may be involved in forwarding packets for other servers. ■

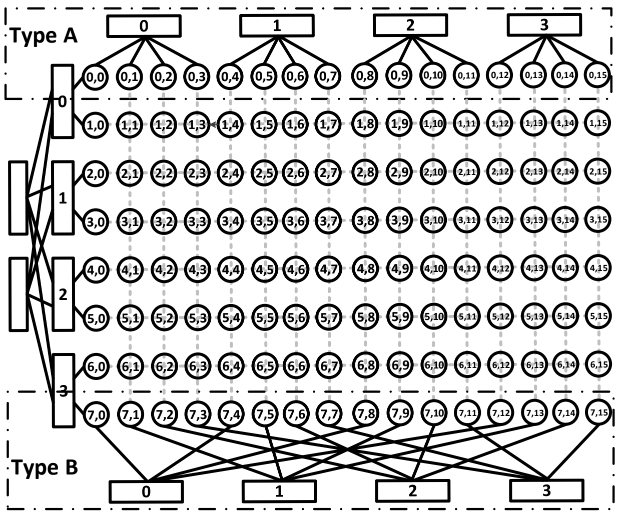


Fig. 2. FRectangle(4)

D. FRectangle Construction

As we can see, the FSquare architecture uses too many switches, which significantly increases the power consumption of the DCN. To reduce the power consumption, we aim to reduce the number of switches in the architecture. A direct approach is to replace the folded Clos topology in one dimension with a traditional single-rooted tree topology; however, the traditional single-rooted tree topology is known to have a large oversubscription ratio, which generally results in a low bisection bandwidth. Thus, we adopt another strategy, which uses two types of simple interconnections to replace the folded Clos topology in all the rows. The new architecture is named FRectangle. In FRectangle(n), the interconnections in all the columns are the same as in FSquare(n). However, in each row, there are n^2 servers. In each row, n switches are used to interconnect n servers. Each row of the FRectangle architecture chooses one type of interconnections from the following:

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

We let FRectangle choose from the two types of interconnections in an interleaved fashion; in other words, if $i\%2 = 0$, the ith row chooses the type A interconnections; if $i\%2 = 1$, the ith row chooses the type B interconnections. The *row ToR* switches, *column ToR* switches, and *column level 2 switches* in FRectangle are similarly defined to those in FSquare. There are no *row level 2 switches* in FRectangle. Fig. 2 shows an FRectangle(n). Notice that we only draw the zeroth column, the zeroth and last rows; other columns and rows are represented by grey dash lines.

E. Routing in FRectangle

Again, we consider the shortest path routing scheme in FRectangle. Before presenting the routing scheme, we would like to introduce some characteristics in FRectangle.

Characteristic 1. *For two servers in the same row, communications among them may or may not be completed in the current row. 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 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 $n/2$ servers connecting to the kth column ToR switch in the jth column, i.e., servers $a_{kn/2,j}$, $a_{kn/2+1,j}$, $\cdots$, $a_{kn/2+n/2-1,j}$. Since rows choose type A and type B interconnections in a interleaved fashion, and $n/2 \geq 2$, this characteristic follows directly. Specifically, if $n/2$ is even, there will be $n/4$ servers belonging to the type A row and $n/4$ servers belonging to the type B row. If $n/2$ is odd, when k is even, there will be $(n+2)/4$ servers belonging to the type A row and $(n-2)/4$ servers belonging to the type B row; when k is odd, there will be $(n-2)/4$ servers belonging to the type A row and $(n+2)/4$ servers belonging to the type B row. ■

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 $a_{k,l}$ ($0 \leq i, k \leq n^2/2-1$ and $0 \leq j, l \leq n^2$–1), respectively. Again, we first consider how the shortest path can be constructed, and then discuss how the routing scheme can be implemented. 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 that in FSquare, 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 characteristics observed before, 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.

If 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, $a_{i,\lfloor j/n \rfloor n}$, $a_{i,\lfloor j/n \rfloor n+1}$, $a_{i,\lfloor j/n \rfloor n+2}$, $\cdots$, $a_{i,\lfloor j/n \rfloor n+n-1}$ 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, $a_{k,(l\%n)}$, $a_{k,(l\%n)+n}$, $a_{k,(l\%n)+2n}$, $\cdots$, $a_{k,(l\%n)+(n-1)n}$ 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.

If $a_{i,j}$ belongs to a type B row, and $a_{k,l}$ 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 discussions here.

If $a_{i,j}$ and $a_{k,l}$ 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 **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 3 switches; and the path from the third relay server to $a_{k,l}$, which includes one switch. Fig. 3 shows an example that illustrates the shortest path construction in this case.

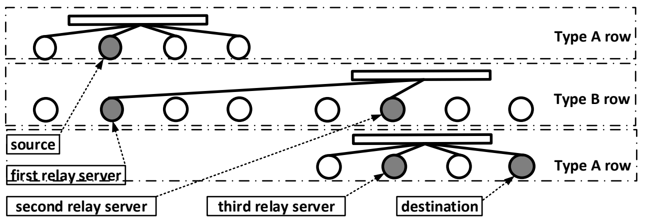


Fig. 3. Shortest path for the case when the source and destination both belong to type A rows.

If $a_{i,j}$ and $a_{k,l}$ 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.

We also let the source server determine the intermediate server(s) in the shortest path, if there is any, and then push the servers in the shortest path (including the destination, while excluding the source) from the last one to the first one into the *server_stack*, and label the intermediate server(s) as *fake* destination(s) and label the real destination as a *true* destination. When sending the packet to a ToR switch, the source server sets the next server of the packet as the temporary destination. When another server in the DCN receives the packet, it pops the *server_stack* 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 sends the packet to a corresponding ToR switch, and sets the next server in the *server_stack* as the temporary destination. When a switch receives the packet, only the temporary destination (either fake or true) set by the previous sending server (either the source or an intermediate server) is visible to the switch. The switch makes forwarding decisions based on this temporary destination.

When multiple choices are available, the source can make a wiser decision on which set of intermediate server(s) to use based on related information along the rows and/or columns, instead of randomly choosing. Also, when a column ToR switch needs to send a packet to a column level 2 switch, it can make a wiser decision on which level 2 switch to send to based on related information within the column. Thus, FRectangle is also a dual-centric design, which enjoys both the fast switching capability of switches and the high programmability of servers.

In both FSquare and FRectangle, the basic shortest path routing schemes can be easily extended to complex routing designs, where ToR switches can help in load-balancing, traffic-aware, fault-tolerant, and even multi-path routing within the row and/or the column, while servers can help in load-balancing, traffic-aware, fault-tolerant, and even multi-path routing among the rows and/or the columns.

F. FRectangle Basic Properties

Property 5. *In an FRectangle(n), the number of servers is $N_v = n^4/2$, and the number of switches is $N_w = 2n^3$.*

TABLE I
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	—	$8/24$	$8d_w{+}d_v$	$N_v/3$	$8p_w/n + p_{nic}$
FBFLY$(8,6,6)$	—	1,572,864	$8/48$	$7d_w{+}d_v$	$N_v/3$	$8p_w/n + p_{nic}$
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	$4/n$	$6d_w{+}4d_v$	$N_v/4$	$4p_w/n + 2p_{nic} + p_{fwd}$
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	$1/n$	$4d_w{+}7d_v$	$> N_v/16$	$p_w/n + 7p_{nic}/4 + 3p_{fwd}/4$

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

Property 6. *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 4 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 consists of only one switch. The third segment is a shortest path between two server in the same column, and thus consists of at most three switches. According to Eq. (1) and Eq. (2), the diameter of FRectangle is $d = 3d_w + 3d_w + (3+1)d_v = 6d_w + 4d_v$. ■

We conjecture that FRectangle has the following property on bisection bandwidth.

Property 7. *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 needs 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 8. *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 2 NIC ports, and servers may be involved in forwarding packets for other servers. ■

IV. MORE ON FSQUARE AND FRECTANGLE

In this section, we investigate more properties of FSquare and FRectangle by comparing them with typical existing architectures and by investigating themselves.

A. Typical Existing Architectures

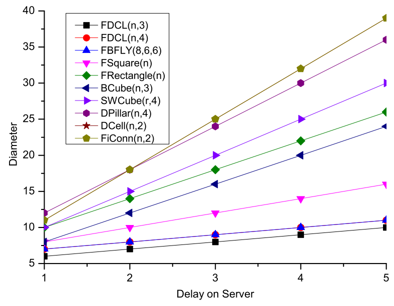
Existing DCN architectures have been classified as switch-centric architectures and server-centric architectures.

Typical switch-centric architectures include folded-Clos [6], Fat-Tree [4], Flattened Butterfly [5], and HyperX [6]. We denote a folded-Clos DCN architecture with l levels of n-port switches by FDCL(n,l). Fat-Tree is actually a folded-Clos with 3 levels, i.e., FDCL$(n,3)$. In a Flattened Butterfly (FBFLY), switches form a generalized hypercube [23]. 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). Typical server-centric architectures include BCube [8], SWCube [12], DPillar [11], DCell [7], and FiConn [9]. BCube, DCell and FiConn are recursively defined architectures, they can be represented by BCube(n,k), DCell(n,k) and FiConn(n,k), respectively, where n is the switch port number and k is the number of the recursion levels. SWCube based on a k-dimensional r-ary generalized hypercube is denoted by SWCube(r,k). DPillar(n,k) is the architecture built with n-port switches, and in which there are k server columns and k switch columns.

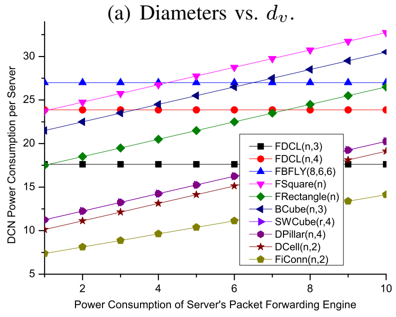
B. Performance and Power

We compare various DCN architectures, constructed by the same homogenous servers and switches, with comparable numbers of servers. For architectures using 24-port and 48-port switches, basic quantitative comparisons are presented in Table I. 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.

The overall trend in the table can be observed: from the top ones (switch-centric) to the bottom ones (server-centric), the DCN power consumption per server of the architectures is decreasing; however, the performances of the architectures



(a) Diameters vs. d_v.



(b) p_V vs. p_{fwd}.

Fig. 4. Comparison of various architectures ($n = 48$). Notice that, some of the lines are overlapped.

are also decreasing, meaning that the bisection bandwidth is decreasing, while the diameter is increasing from an overall view. We regard the delay on a 48-port switch, d_w as 1, and vary the delay on a server, d_v from 1 to 5. Fig. 4(a) shows the diameters of various DCN architectures. For switches with $n = 48$ 1Gbps ports and 1Gbps NIC ports, we set $p_w = 150$ and $p_{nic} = 2$. We vary p_{fwd} from 1 to 10. Fig. 4(b) shows the DCN power consumption per server of various architectures. We can see that FSquare has a lower diameter than all server-centric architectures and a large bisection bandwidth that is comparable to switch-centric ones, at the cost of high power consumption. FRectangle uses less switches, and thus, reduces its power consumption to a value that is generally less then switch-centric architectures; at the same time, FRectangle has a diameter that is less than most of the server-centric architectures and a bisection bandwidth that is greater then most of the server-centric architectures. FDCL$(n, 3)$ demonstrates good performances and satisfiable power efficiency; however, its number of servers is very limited, compared to all other ones.

We can see that FSquare and FRectangle reflect two nice tradeoff designs in the middle and provide DCN designers more flexible choices.

C. *Scalability and Flexibility*

Scalability means that the networking devices, typically the switches, rely on a limited amount of information, which does not increase significantly with the network size, to implement efficient routing algorithms. Since modern data centers usually have large network sizes, scalability is an important requirement. Flexibility requires that expanding the network in a fine-grained fashion should not replace the networking devices or destroy the original architecture. 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 [24], Scafida [25], and Small-World Data Center [26], to provide arbitrary-grained flexibility; however, due to their irregularity, networking devices in these architectures rely on a large amount of information for efficient routing, making them unable to scale to a large network size.

We argue that FSquare and FRectangle demonstrate both scalability and flexibility. Scalability of FSquare results from their high degree of regularity. We propose to reserve some ports on the column level 2 switches in FSquare and FRectangle for them to achieve flexibility. In this case, the level 2 switches each have $n + K$ ports; n of them are used to construct the original architecture, while K of them are reserved for future expansion. Taking FSquare as an example, if we use k $(1 \leq k \leq K)$ reserved ports on all the column level 2 switches to expand the network, we can add k column ToR switches to each column. Thus, we are able to add $kn/2$ rows of servers, i.e., $kn^3/4$ servers to the original architecture. Different k values provide us the ability to flexibly expand the network. Similar flexibility can be achieved in FRectanlge in a similar way.

D. *Elasticity*

To reduce power consumption in a data center when the traffic and/or computation load in the data center is low, network administrators may want to dynamically power off and on network switches and servers. This requires the DCN to be elastic, which basically means turning off network switches and/or servers while maintaining the connectivity in the data center. We can see that, in FSquare, reducing the topology of each row and each column to a traditional single rooted tree still maintains the connectivity. This topology shrinking leads to about one third ($(n-2)/3n$) power reduction if only switches can be powered off while switch ports cannot be powered off independently. If switch ports can be powered off independently, the potential power reduction is even greater. Further, when all the columns are shrunk to single-rooted trees, to maintain the connectivity of the entire architecture, only one row should be connected, to maintain the overall connectivity. This leads to two thirds power reduction. Thus, FSquare provides network administrators with great freedom in powering off and on different numbers of switches, under different traffic loads in the network.

In FRectangle, elasticity can be achieved similarly. One difference is, when all the columns are connected by the shrunk single-rooted tree, to maintain the connectivity of the entire architecture, only one type A row and one type B row must be maintained.

V. Simulations

We conduct routing simulations for both FSquare and FRectangle in a proprietary simulator. Our main goal is to investigate the performances of the two proposed architectures under various traffic conditions.

We build a basic model for store-and-forward switches. Both switches and servers are assumed to have 1Gbps 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 2μs. The switch's and the server's processing delays, $d_{w,p}$ and $d_{v,p}$ are normalized by this time unit. Queuing delay happens when multiple packets compete for the same output port (either on a switch or on a server) simultaneously. For both of the architectures, we only consider the basic shortest path routing schemes, since our focus in this paper is on the architectures themselves, instead of on designing efficient and/or adaptive routing algorithms for specific traffic patterns. More efficient and/or adaptive routing algorithm designs are left for future work.

We consider three typical traffic patterns in data centers, referring to [27].

Random Traffic Pattern, abbreviated as **Random**: For each flow, the source and destination servers are randomly chosen among all of the servers, as long as they are not identical.

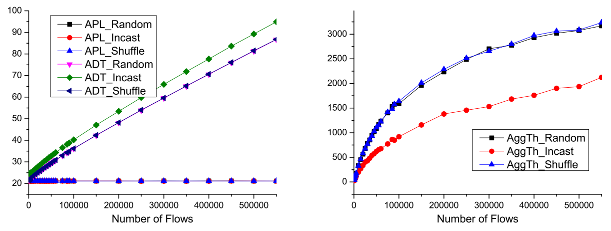
Incast Traffic Pattern, abbreviated as **Incast**: A server receives traffic flows from multiple random servers in the network. This traffic pattern simulates the shuffle stage of the widely-used MapReduce framework. We assume that a server receives flows from 10 other random servers.

Rack Shuffle Traffic Pattern, abbreviated as **Shuffle**: 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.

In all of the traffic patterns, we generate different numbers of flows, to reflect different traffic loads in the network. All flows are generated and pushed to the network at the same time. We calculate the Aggregate Throughput (AggTh), the Average Path Length (APL) and the Average Delivery Time (ADT) for the two architectures. 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 number of flows divided by the maximum delivery time among all flows. For APL, the path lengths are calculated based on our unified definition in Eq. (1).

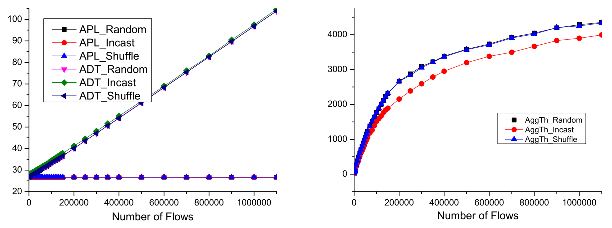
A. Simulation Settings

We choose switch port number, n=12. We set the switch's processing delay $d_{w,p} = 1$, and set the server's processing delay $d_{v,p} = 4$. 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. Step sizes are different during different ranges. For



(a) APL and ADT. (b) Aggregate throughput.

Fig. 5. APL, ADT and AggTh vs. No. of flows in FSquare.



(a) APL and ADT. (b) Aggregate throughput.

Fig. 6. APL, ADT and AggTh vs. No. of flows in FRectangle.

each number of flows and each traffic pattern, we randomly generate several different sets of flows. Then, the APL, ADT and AggTh for each traffic pattern given the number of flows is calculated as the average APL, ADT, and AggTh of the corresponding sets of flows. 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. We also randomly generate several different sets of flows for each number of flows and each traffic pattern, and then calculate and compare the average values.

B. Simulation Results

Fig. 5 and Fig. 6 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 become congested, the ADTs in FSquare and FRectangle only increase linearly. We can see that both architecture 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 Incast traffic are always the worst among the three. We can see the performances of Incast 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.

VI. Additional Discussions

Designing DCN architectures is never a simple task. Various issues should be addressed. We briefly discuss other issues related to our designs.

Floor Planning. Both FSquare and FRectangle are two-dimensional architectures. Since practical data centers usually have two-dimensional floor planning, both FSquare and FRectangle can fit into practical data centers quite well.

New Technologies. Recently, two important new technologies have been introduced into DCNs: the 60 GHz wireless technology and the optical switching technology. In this paper, our default settings are in traditional electrical switching technology; however, the dual-centric design philosophy can certainly be applied to hybrid DCNs, where new technologies are used.

Various Other Issues. Actually, the dual-centric design is a mix of switch-centric and server-centric designs. We expect that various related issues in dual-centric designs can be addressed by existing works on both switch-centric and server-centric designs, after minimal or moderate adaptations and modifications.

VII. Conclusion and Future Works

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. We propose two typical dual-centric DCN architectures: FSquare and FRectangle. By comparing them with existing architectures and by investigating themselves, we show that these two dual-centric DCN architectures have various nice properties for practical data centers, and provide flexible choices in designing DCN architectures. As switches are becoming more programmable via software defined networking, and servers are tending to utilize specialized hardware for packet forwarding, both switches and servers will carry both packet forwarding capability and routing intelligence. The proposed dual-centric design philosophy will certainly become a potential candidate in future DCN architecture designs.

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

Acknowledgement

This work is supported in part by NSF grants CNS 149860, CNS 1461932, CNS 1460971, CNS 1439672, CNS 1301774, ECCS 1231461, ECCS 1128209, and CNS 1138963.

References

[1] S. Ghemawat, H. Gobioff, and S.-T. Leung, "The Google file system," in *ACM SOSP*, 2003, pp. 29–43.

[2] J. Dean and S. Ghemawat, "MapReduce: simplified data processing on large clusters," *Commun. ACM*, vol. 51, no. 1, pp. 107–113, Jan. 2008.

[3] Y. Zhang and N. Ansari, "On architecture design, congestion notification, tcp incast and power consumption in data centers," *Communications Surveys Tutorials, IEEE*, vol. 15, no. 1, pp. 39–64, 2013.

[4] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," in *ACM SIGCOMM*, 2008, pp. 63–74.

[5] D. Abts, M. R. Marty, P. M. Wells, P. Klausler, and H. Liu, "Energy proportional datacenter networks," in *ACM/IEEE ISCA*, 2010, pp. 338–347.

[6] J. H. Ahn, N. Binkert, A. Davis, M. McLaren, and R. S. Schreiber, "HyperX: Topology, routing, and packaging of efficient large-scale networks," in *ACM/IEEE SC*, 2009, pp. 41:1–41:11.

[7] 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*, 2008.

[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," in *ACM SIGCOMM*, 2009, pp. 63–74.

[9] D. Li, C. Guo, H. Wu, K. Tan, Y. Zhang, and S. Lu, "FiConn: Using backup port for server interconnection in data centers," in *IEEE INFOCOM*, 2009, pp. 2276–2285.

[10] 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," *Computers, IEEE Transactions on*, vol. 62, no. 7, pp. 1303–1317, July 2013.

[11] Y. Liao, D. Yin, and L. Gao, "DPillar: Scalable dual-port server interconnection for data center networks," in *IEEE ICCCN*, 2010, pp. 1–6.

[12] D. Li and J. Wu, "On the design and analysis of data center network architectures for interconnecting dual-port servers," in *IEEE INFOCOM*, 2014, pp. 1851–1859.

[13] D. Kreutz, F. Ramos, P. Esteves V., C. Esteve R., S. Azodolmolky, and S. Uhlig, "Software-defined networking: A comprehensive survey," *Proceedings of the IEEE*, vol. 103, no. 1, pp. 14–76, Jan 2015.

[14] G. Lu, C. Guo, Y. Li, Z. Zhou, T. Yuan, H. Wu, Y. Xiong, R. Gao, and Y. Zhang, "ServerSwitch: A programmable and high performance platform for data center networks," in *USENIX NSDI*, 2011, pp. 15–28.

[15] Y. Liu, J. K. Muppala, M. Veeraraghavan, D. Lin, and M. Hamdi, *Data Center Networks: Topologies, Architectures and Fault-Tolerance Characteristics*. Springer Briefs in Computer Science, 2013.

[16] L. Gyarmati and T. A. Trinh, "How can architecture help to reduce energy consumption in data center networking?" in *e-Energy*, 2010, pp. 183–186.

[17] J. Perry, A. Ousterhout, H. Balakrishnan, D. Shah, and H. Fugal, "Fastpass: A centralized "zero-queue" datacenter network," in *ACM SIGCOMM*, 2014, pp. 307–318.

[18] M. Alizadeh, A. Kabbani, T. Edsall, B. Prabhakar, A. Vahdat, and M. Yasuda, "Less is more: Trading a little bandwidth for ultra-low latency in the data center," in *USENIX NSDI*, 2012, pp. 19–19.

[19] A. Greenberg and D. A. Maltz. What goes into a data center - SIGMETRICS 2009 tutorial. [Online]. Available: http://research.microsoft.com/apps/pubs/default.aspx?id=81782

[20] L. Popa, S. Ratnasamy, G. Iannaccone, A. Krishnamurthy, and I. Stoica, "A cost comparison of datacenter network architectures," in *ACM Co-NEXT*, 2010, pp. 16:1–16:12.

[21] "Cisco nexus 2000 series fabric extenders data sheet." [Online]. Available: http://www.cisco.com/c/en/us/products/collateral/switches/nexus-2000-series-fabric-extenders/data_sheet_c78-507093.html

[22] "Intel gigabit et, et2, and ef multi-port server adapters." [Online]. Available: http://www.intel.com/content/dam/doc/product-brief/gigabit-et-et2-ef-multi-port-server-adapters-brief.pdf

[23] L. Bhuyan and D. Agrawal, "Generalized hypercube and hyperbus structures for a computer network," *Computers, IEEE Trans. on*, vol. C-33, no. 4, pp. 323–333, 1984.

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

[25] 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.

[26] J.-Y. Shin, B. Wong, and E. G. S., "Small-world datacenters," in *ACM SOCC*, 2011.

[27] Y. J. Liu, P. X. Gao, B. Wong, and S. Keshav, "Quartz: A new design element for low-latency DCNs," in *ACM SIGCOMM*, 2014, pp. 283–294.