JieLin: A Scalable and Fault Tolerant Server-Centric Data Center Network Architecture

*Ting Wang, ‡Mounir Hamdi
*Hong Kong University of Science and Technology, ‡Hamad Bin Khalifa University
{twangah, hamdi}@cse.ust.hk

***Abstract*—To support the fast growing cloud computing services and provide a core infrastructure to meet the increasing computing and storage requirements, the number of servers in today's data centers is expanding exponentially, which leads to enormous challenges in designing an efficient and cost-effective data center network. Traditional proposals either suffer from poor reliability, endure performance bottleneck, scale too slowly, or are expensive to construct. Motivated by these challenges, this paper presents the design, analysis, and implementation of *JieLin*, a novel server-centric network architecture that has many desirable features for data center networking. Besides the excellent scalability and good fault tolerance, *JieLin* also achieves high performance in bisection bandwidth, average path length, aggregate bottleneck throughput, and cost-effectiveness. Moreover, in order to maximize the theoretical performance of *JieLin* a congestion-aware fault-tolerant adaptive routing algorithm has been specially designed. In addition to theoretical analysis, extensive simulations are conducted to further prove the feasibility and good performance of *JieLin*.**

***Keywords*—Data center, Network topology, Server-centric, Fault Tolerance, Adaptive Routing, Scalability**

I. Introduction

Implemented as a centralized repository hosting a massive number of servers, data centers provide the backbone infrastructure to support many large-scale enterprise IT needs as well as emerging cloud computing services. The network interconnection of the data center not only plays an important role in the performance and robustness of the services, but also determines the network scalability, cost-effectiveness, network capacity, and so on. The traditional three-tiered tree-based data center network, besides being very costly, results in significant bandwidth oversubscription towards the network core which may become the bottleneck [1]. As a result, many alternative approaches such as Fat Tree [2], DCell [3], BCube [4], VL2 [5], HyperBCube [6], SprintNet [7], NovaCube [8], and CLOT [9] have been proposed, but these proposals also have their own limitations. Generally, the data center network architecture should not only be scalable to a larger number of servers, but also fault tolerant, low network diameter and cost-effective to construct. Moreover, according to the findings in [10] the applications in data centers usually lack of communication locality, which implies that the network infrastructure must provide high aggregate bisection bandwidth for worst-case traffic patterns.

In order to meet these challenges and to achieve a scalable, agile and cost-effective interconnect for data centers, in this paper we propose a novel data center network architecture termed *JieLin*, which inherits the advantages of previous approaches while avoiding their limitations. Benefiting from rich physical connectivity and the fault-tolerant routing algorithm, *JieLin* achieves good fault tolerance, without any single point of failure. Besides all that, *JieLin* provides an excellent scalability of $O(n*(an)^{(k-1)})$ where it can easily expand to a larger sized network. Moreover, the bisection bandwidth of *JieLin* reaches $O(n^3)$, which is far higher than other server-centric schemes whose bisection bandwidth is only $O(n^2)$. Furthermore, *JieLin* also yields good cost-effectiveness where it is built using only low-end commodity switches without any switch upgrade, and the linking and switching complexity per server of *JieLin* is only 2/3 and 2/5 as that of Fat Tree, respectively. In addition, a congestion-aware adaptive routing algorithm is specially designed aiming to help *JieLin* achieve a better overall performance.

The rest of the paper is organized as follows. First we briefly review the related research literature in Section II. Section III demonstrates the structure design of *JieLin*. Then, a congestion-aware routing algorithm is proposed for *JieLin* in Section IV. Section V gives the evaluation results of *JieLin*. Finally, Section VI concludes the whole paper.

II. Related Work

Considerable research and investigations have been conducted to design an efficient data center network architecture so as to deliver peak performance to users. The state-of-art data center network topologies can be generally classified into four categories: switch-centric, server-centric, hybrid network architectures and direct network interconnection.

The switch-centric topology is the most typical data center architecture, which places the interconnection and routing intelligence on switches. Examples of this scheme include Fat Tree [2], VL2 [5], and Portland [11]. Comparatively, in the server-centric scheme, not only do the switches need to forward packets, the servers are also expected to participate in relaying traffic. Typical representatives of server-centric topologies are DCell [3], BCube [4], HyperBCube [6] and SprintNet [12]. These kinds of architectures are usually recursively constructed with good scalability. The hybrid network architecture augments the packet switching network with a high speed optical circuit-switched network to supply high bandwidth to applications. However, the optics cannot achieve full bisection bandwidth at packet granularity and suffer from slow switching speed which costs as long as tens of milliseconds. Helios [13] and c-Through [14] can be grouped into this category. The last type of data center interconnection is known as direct network architecture, where the servers are

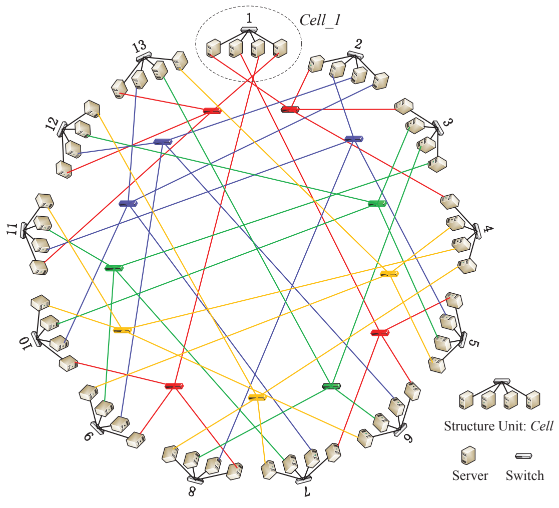
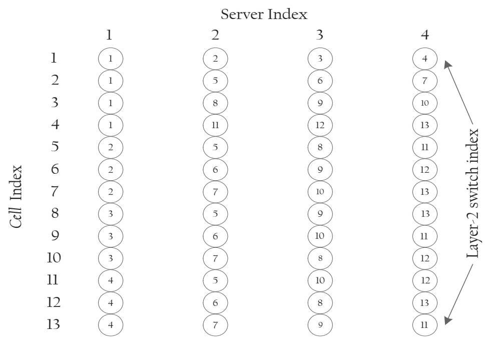


Fig. 1. A 52-server *JieLin* network using 4-port switches.


Server Index
Cell Index
Layer-2 switch index

Cell Index \ Server Index	1	2	3	4
1	1	2	3	4
2	1	5	6	7
3	1	8	9	10
4	1	11	12	13
5	2	5	8	11
6	2	6	9	12
7	2	7	10	13
8	3	5	9	13
9	3	6	10	11
10	3	7	8	12
11	4	5	10	12
12	4	6	8	13
13	4	7	9	11

Fig. 2. The $13 * 4$ connection matrix for a 52-server sized *JieLin* network.

directly connected with each other based on a regular pattern (e.g. torus), such as NovaCube [8], CamCube [15] and Small-World [16]. They are switchless architectures without any switches, routers or other network devices. In this architecture, the servers should also get involved in forwarding packets by using multiple NIC ports.

III. *JieLin* Structure Design and Analysis

A. *JieLin Physical Structure*

JieLin is a recursively constructed server-centric network architecture with two layers, where a higher layer *JieLin* is built from a certain number of lower layer *JieLin*. The most basic construction unit (or the first layer) named $Cell$, which is used to construct a larger *JieLin*, consists of one n-port commodity switch (layer-1 switch) and n servers. Each server of any $Cell$ has two ports, where one port is connected to the switch in the same $Cell$ and the other port is used for inter-$Cell$ connections.

The second layer of *JieLin* contains $n^2 - n + 1$ different $Cells$, which are interconnected using $n^2 - n + 1$ n-port switches (layer-2 switch). For any certain $Cell$, each server of this $Cell$ is connected to the other $n - 1$ different $Cells$ using a layer-2 switch, and the connected inter-$Cells$ of each server in the same $Cell$ are different from each other. As a

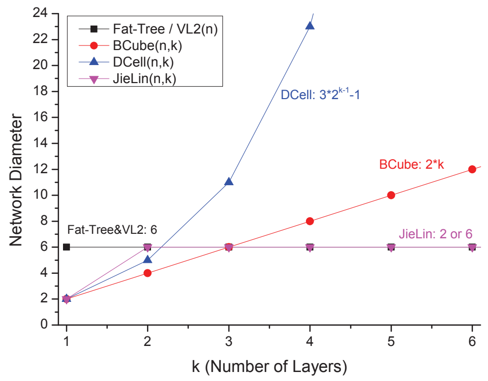


Fig. 3. The comparisons of network diameter between different architectures.

result, each $Cell$ is connected to all other $n * (n - 1)$ $Cells$, and there are $n * (n - 1) + 1 = n^2 - n + 1$ different $Cells$ in total. Consequently, the total number of *JieLin*'s servers is calculated as $n * (n^2 - n + 1) = n^3 - n^2 + n$, and the total number of layer-2 switches is $\frac{n^3 - n^2 + n}{n} = n^2 - n + 1$. In this way, all the $Cells$ are fully connected by the $n^2 - n + 1$ layer-2 switches, and any two different $Cells$ can communicate with each other through only one layer-2 switch without transitions to any other intermediate $Cells$ or switches. Fig.1 shows an example of *JieLin* network topology using 4-port switches, where the number of different $Cells$ is $4^2 - 4 + 1 = 13$ and the total number of severs is $4^3 - 4^2 + 4 = 52$.

In *JieLin*, each server can be identified using two co-ordinates (s, c), which indicate the s-th server in the c-th $Cell$. Actually, the whole network can be represented as a $(n^2 - n + 1) * n$ matrix $\mathcal{M}$, where the line and column of the matrix correspond to the $Cell$ index c and the server index s in the $Cell$ c, respectively. The value of each element (s, c) in the matrix $\mathcal{M}$ indicates the layer-2 switch index i which the server (s, c) is connected to, that is, $\mathcal{M}(s, c) = i$ meaning that the server (s, c) is connected to the i-th layer-2 switch. The connection matrix $\mathcal{M}$ can be easily generated using the backtracing algorithm with a constraint of each $Cell$ connecting to other $n(n - 1)$ $Cells$. Notice that the matrix is only needed to be generated once before the installation of the network. Fig.2 illustrates an example of connection matrix for a 52-server sized *JieLin* network.

B. *Key Properties of JieLin Network*

In this subsection, some key structural properties of *JieLin* is theoretically analysed. The primary analytical results of *JieLin*'s features are summarized in Table I, which also presents comparisons between *JieLin* and some other typical DCN architectures with regard to scalability, physical cost, bisection bandwidth, and network diameter.

1) *Network Diameter and Average Path Length:*

The network diameter is known as the longest of all the calculated shortest paths in a network, that is, the maximum shortest path length among all the server pairs. Potentially, a smaller network diameter implies a lower network latency and leads to more effective routing in practice [1]. In a *JieLin*

TABLE I
THE COMPARISON BETWEEN SOME DIFFERENT NETWORK ARCHITECTURES

	Fat Tree (3 layers)	VL2 (3 layers)	DCell (2 layers)	BCube (2 layers)	SprintNet (2 layers)	JieLin (2 layers)
Servers Number	$\frac{n^3}{4}$	$\frac{(n-2)n^2}{4}$	$n(n+1)$	n^2	$(\frac{c}{c+1})^2n^2 + \frac{c}{c+1}n$	$n^3 - n^2 + n$
Links Number	$\frac{3n^3}{4}$	$\frac{(n+2)n^2}{4}$	$\frac{3n(n+1)}{2}$	$2n^2$	$\frac{c^2n^2}{c+1} + cn$	$2n^3 - 2n^2 + 2n$
per Server	3	$\frac{n+2}{n-2}$	$\frac{3}{2}$	2	≥ 2	2
Switches Number	$\frac{5n^2}{4}$	$\frac{n^2}{4} + \frac{3n}{2}$	$n + 1$	$2n$	$\frac{c^2}{c+1}n + c$	$2n^2 - 2n + 2$
per Server	$\frac{5}{n}$	$\frac{n+6}{n^2-2n}$	$\frac{1}{n}$	$\frac{2}{n}$	$\frac{c+1}{n}$	$\frac{2}{n}$
Bisection Bandwidth	$\frac{n^3}{8}$	$\frac{n^2}{4}$	$\frac{n^2}{4} + \frac{n}{2}$	$\frac{n^2}{2}$	$\frac{c^2n^2}{2(c+1)^2} + cn$	$\frac{n^3-2n^2+3n}{2}$
per Server	$\frac{1}{2}$	$\frac{1}{n-2}$	$\approx \frac{1}{4}$	$\frac{1}{2}$	$\frac{1}{2} + \frac{(2c+1)(c+1)}{2(cn+c+1)}$	$\approx \frac{1}{2}$
Network Diameter	6	6	5	4	4	6

TABLE II
THE SCALABILITY OF DIFFERENT ARCHITECTURES WITH TWO LAYERS

Port Number	Server Number				
	Fat Tree	VL2	DCell	BCube	JieLin
4	16	8	20	16	52
8	128	96	72	64	456
16	1,024	896	272	256	3,856
24	3,456	3,168	600	576	13,272
48	27,648	26,496	2,352	2,304	108,336
64	65,536	63,488	4,160	4,096	258,112
128	524,288	516,096	16,512	16,384	2,080,896

network, the distance between any two servers within a $Cell$ is two. The path length between any pair of servers in different $Cell$s is four or six. Therefore, the network diameter of *JieLin* is six, which means any pair of servers can be reached from each other within six hops. Fig.3 illustrates the relationship between the number of network layers k and the network diameter. It can be seen that the diameter of *JieLin* is a constant for $k \geq 2$, which is superior to BCube and DCell whose diameters increase linearly or exponentially as the number of layers increase.

The network diameter of three tiered tree-like topologies Fat Tree and VL2 is also six, which is at the same level with *JieLin*. However, the average path of *JieLin* is much shorter than Fat Tree and VL2 because *JieLin* has lots of paths of length of two (within *Cell*) or four (some inter-$Cell$ server pairs). This is further proved in our simulations. Not only that, the average path length largely determines the overall network latency (such as transmission delay and the delay caused by number of times to be queued). From this perspective, *JieLin* achieves a lower traffic latency than other candidates.

2) Scalability:

JieLin achieves an excellent scalability, where *JieLin* can easily expand to be a very large data center network at a low average building cost. By using identical low-end n-port switches and only two-layer architecture, *JieLin* can host $n^3 - n^2 + n$ servers, which is approximately 4 times that of a Fat Tree/VL2 and n times that of a DCell/BCube/SprintNet. Table II presents some numerical results of different architectures using switches with different number of ports.

3) Cost Effectiveness:

Instead of using expensive high-end switches (as in FatTree, PortLand, VL2, etc.), *JieLin* is constructed only using low-end commodity switches. The linking and switching complexity of *JieLin* is $O(n^3)$ and $O(n^2)$, respectively, which is at the same level with Fat Tree and VL2 but higher than DCell and BCube. However, the average link/switch cost per server of *JieLin* is identical to a two-layer BCube. Moreover, compared with FatTree, the number of links and the number of switches required by each server in a *JieLin* is only 2/3 and 2/5 as that of a (1/4)-sized Fat Tree, respectively.

4) Bisection Bandwidth:

The bisection bandwidth of a network refers to the capacity across the links of the "narrowest" bisector which partitions the network into two equally-sized parts. Since the bisection bandwidth of a network is calculated as the minimum capacity among all possible bisections, thus, the bisection bandwidth can be used to measure the worst-case network capacity [10]. As shown in Table I, the bisection bandwidth of *JieLin* and FatTree are both at the level of $O(n^3)$ (but *JieLin* is approximately four times higher than FatTree), which is greatly better than VL2/DCell/BCube/SprintNet's $O(n^2)$.

5) Aggregate Bottleneck Throughput:

The all-to-all traffic pattern simulates the most intensive network activities, where each server communicates with all other servers. Among all the flows that a server established with all other servers, the flows receiving the smallest throughput are known as bottleneck flows. The aggregate bottleneck throughput (ABT) can be calculated by summing up the throughputs of all the bottleneck flows.

The aggregate bottleneck throughput is very sensitive to the average path length (APL). Assume that the overall network capacity is NC_{links} (the sum of all link capacities), if we use NCP_{ABT} to denote the proportion of the overall network capacity that the aggregate bottleneck throughput can reach (i.e. $\frac{ABT}{NC_{links}}$) under the all-to-all traffic pattern, then we can derive that

$$NCP_{ABT} = \frac{1}{APL} \tag{1}$$

assuming that the bandwidth of each link in one-way communication is 1 and all the links are equal in *JieLin*.

$Proof$: Define N_{flows} as the total number of flows in the network, N_{links} indicates the total number of two-way communication links (so there are actually $2N_{links}$ virtual one-way communication links), and NF_{link} represents the number of flows carried on one link (thus $\frac{1}{NF_{link}}$ indicates the throughput that per flow receives), then we have:

$$ABT = N_{flows} * \frac{1}{NF_{link}} \tag{2}$$

$$NF_{link} = \frac{N_{flows} * APL}{2N_{links}} \tag{3}$$

Combining Equation 2 and 3, we have:

$$ABT = \frac{2N_{links}}{APL} \tag{4}$$

Hence, when the link capacity is 1, there is

$$NCP_{ABT} = \frac{ABT}{NC_{links}} = \frac{ABT}{2N_{links}} = \frac{1}{APL} \tag{5}$$

which concludes the proof.

As aforementioned, compared with other candidates *JieLin* achieves lower network diameter and smaller average path length, which implies *JieLin* is more likely to achieve a higher aggregate bottleneck throughput. This preliminary derivation is verified in our simulations.

6) Fault Tolerance:

Owing to full interconnections between servers in different *Cell*s, *JieLin* achieves very good performance in fault tolerance. In Fat Tree/VL2/Portland/Clos-based architectures, any failure of a top-of-rack (ToR) switch will cause the disconnection of the whole rack of servers, and the situation is even worse for the failure of end-of-rack (EoR) switches (or aggregation switches) or core switches. Comparatively, missing one switch (layer-1 or layer-2 switch) or server does not lead to any disconnections, and the servers in the fault regions still can communicate with each other via intermediate *Cell*s. Even if one or more whole $Cell$s have failed, it still does not degrade the network performance much, and the remaining $Cell$s are still fully connected. This reveals the good reliability of *JieLin* architecture.

Algorithm 1 Congestion-aware Adaptive Routing Algorithm CARA(source_node *src*, destination_node *dst*)

```
/* $src[s,c_1]$: source server;   $dst[d,c_2]$: source server;
   $sw_x^1$: the layer-1 switch in the $x$-th $Cell$;
   $sw_y^2$=$\mathcal{M}(s,c_1)$: the $y$-th layer-2 switch connecting server $(s,c_1)$;
   $sw(c_1,c_2)$: the layer-2 switch connecting $Cell$ $c_1$ and $Cell$ $c_2$; */
if $src.fail()$ || $dst.fail()$ then
    return null;
else
    if $c_1$ == $c_2$ then
        if $sw_{c_1}^1.on()$ then
            return Route0: $src \rightarrow switch\ sw_{c_1}^1 \rightarrow dst$;
        else if $sw_{c_1}^1.fail()$ then
            if $\mathcal{M}(s,c_1).on()$ then
                return Route4: $src \rightarrow \mathcal{M}(s,c_1) \rightarrow itr(e,c_3) \rightarrow CARA(itr,dst)$;
            else
                return null;
            end if
        end if
    else
        if $\mathcal{M}(s,c_1)$ == $\mathcal{M}(d,c_2)$ then
            if $\mathcal{M}(s,c_1).on()$ then
                return Route1: $src \rightarrow switch\ \mathcal{M}(s,c_1) \rightarrow dst$;
            else if $\mathcal{M}(s,c_1).fail()$ then
                return Route5: $src \rightarrow sw_{c_1}^1 \rightarrow otr(g,c_1) \rightarrow \mathcal{M}(g,c_1) \rightarrow ptr(h,c_3) \rightarrow CARA(ptr,dst)$;
            end if
        else
            suppose $sw(c_1,c_2)$ connects $ms(m,c_1)$ and $nd(n,c_2)$
            if $sw(c_1,c_2).on()$ && $sw_{c_1}^1.on()$ then
                if $d == n$ then
                    return Route2: $src \rightarrow sw_{c_1}^1 \rightarrow ms \rightarrow sw(c_1,c_2) \rightarrow dst$;
                else
                    return Route3: $src \rightarrow sw_{c_1}^1 \rightarrow ms \rightarrow sw(c_1,c_2) \rightarrow nd \rightarrow sw_{c_2}^1 \rightarrow dst$;
                end if
            else if $sw(c_1,c_2).fail()$ || $sw_{c_1}^1.fail()$ then
                if $\mathcal{M}(s,c_1).on()$ then
                    return Route4;
                else
                    return null;
                end if
            end if
        end if
    end if
end if
```

IV. Congestion-aware Adaptive Routing Design

In order to maximize the theoretical performance of *JieLin*, in this section a congestion-aware adaptive routing (CARA) algorithm is designed. CARA is a shortest-path based routing algorithm, and it is not only adaptive to the network traffics, but also behaves fault tolerant. When making routing decisions CARA takes the network state into consideration aiming to avoid network congestion and achieve good load balancing by exploiting the path diversity.

Given the source server $src(s,c_1)$ and destination server $dst(d,c_2)$, a routing path between them can be precisely computed according to Algorithm 1. Generally, the whole routing procedure of CARA can be divided into two cases as shown in the pseudocode of Algorithm 1. Firstly, if the servers $src(s,c_1)$ and $dst(d,c_2)$ are located in the same $Cell$, i.e., $c_1 = c_2$, the destination can be reached via the layer-1 switch $sw_{c_1}^1$ of this $Cell$ with the path length of two (i.e. Route0). If the switch $sw_{c_1}^1$ fails, then the packets should be firstly routed to another server $itr(e,c_3)$ of an intermediate $Cell$ through the layer-2 switch $\mathcal{M}(s,c_1)$ and then rerouted to the destination server regarding $itr(e,c_3)$ as the new source server (i.e. Route4). There are $n-1$ equal-cost choices for the selection of an intermediate $Cell$, and the one whose corresponding queue in the layer-2 switch $\mathcal{M}(s,c_1)$ is smaller will be selected. This guarantees the flow always choose the link with most available bandwidth. From another perspective, when a port or its connected server fails, its available bandwidth should be regarded as zero, so the traffic can be routed by other ports to a different $Cell$ to bypass the network faults. If in case the switch $\mathcal{M}(s,c_1)$ also fails, then the source server will be disconnected without any feasible routes. The second case is that $src(s,c_1)$ and $dst(d,c_2)$ are located in different $Cell$, i.e., $c_1 \neq c_2$. This case has two situations. The first situation is that server $src(s,c_1)$ is directly connected to $dst(d,c_2)$ by a layer-2 switch, i.e., $\mathcal{M}(s,c_1) = \mathcal{M}(d,c_2)$. Then the route will be $src \rightarrow switch\ \mathcal{M}(s,c_1) \rightarrow dst$ (i.e. Route1) with the path length of two. If switch $\mathcal{M}(s,c_1)$ happens to fail, then the traffic is firstly forwarded to one server $otr(g,c_1)$ within the same $Cell$ via switch $sw_{c_1}^1$, then the traffic is routed to its directly connected server $ptr(h, c_3)$ in another $Cell$ through the switch $\mathcal{M}(h, c_3)$, and then traffic can be routed to the destination by considering server $ptr(h, c_3)$ as the new source server (i.e. Route5). Likewise, the server $otr(g, c_1)$ and $ptr(h, c_3)$ are selected based on the amount of their available bandwidth. The second situation is that server $src(s, c_1)$ and server $dst(d, c_2)$ are not directly connected. We suppose switch $sw(c_1, c_2)$ is the layer-2 switch that connects $Cell\ c_1$ and $Cell\ c_2$, and the connected servers are $ms(m, c_1)$ and $nd(n, c_2)$, respectively. If the destination server (d, c_2) is exactly the server $nd(n, c_2)$, then the routing path will be $src \rightarrow sw^1_{c_1} \rightarrow ms \rightarrow sw(c_1, c_2) \rightarrow dst$ (i.e. Route2). Otherwise, the route will be $src \rightarrow sw^1_{c_1} \rightarrow ms \rightarrow sw(c_1, c_2) \rightarrow nd \rightarrow sw^1_{c_2} \rightarrow dst$ (i.e. Route3). If in case switch $sw(c_1, c_2)$ and switch $sw^1_{c_1}$ cannot be used, then the traffic will take the path of Route4. If switch $\mathcal{M}(s, c_1)$ also fails, there will be no feasible paths and the server is totally disconnected. The statistics of routing path length of CARA is summarized in Table III.

Following the above procedures of CARA algorithm, the traffic will be routed along the shortest path which has the most available bandwidth if there are multiple equal-cost paths. Besides, CARA is also a fault tolerant routing algorithm which can make a detour around the faulty nodes automatically. Moreover, each node of *JieLin* has at most $n^2 - n + 1$ possibilities of routing direction (because there are only $n^2 - n + 1$ $Cell$s), and has only up to $2(n-1)$ choices for the first hop (because each node can reach $2(n-1)$ servers in one hop), therefore, a route table with the size of only $(n^2 - n + 1)log_2(2n - 2)$ bits can describe its routing behaviours, which reveals a very low hardware cost. For example, a *JieLin* network with around one million servers $(n = 100)$ only requires a 9.4KB sized routing table.

TABLE III
THE PATH LENGTH DISTRIBUTION

	Fault-free Case				**Faulty Case**	
Route	Route0	Route1	Route2	Route3	Route4	Route5
Length	2	2	4	6	10	10

V. EVALUATION

This section presents the evaluation of *JieLin* from various aspects including the average path length, aggregate bottleneck throughput, and the performance under faulty conditions. We implement *JieLin* interconnection and the CARA routing algorithm in our DCNSim [17] platform, which is a Java implemented flow-level data center network simulator, to conduct comprehensive evaluations. The all-to-all traffic pattern, which simulates the most intensive network activities, is used to evaluate the network performance under the most rigorous conditions. The exponential distribution flow mode is applied to determine the packet interval time. Moreover, all links of *JieLin* are equal and enabled with two-way communications.

A. Average Path Length

As discussed in Section III, *JieLin* achieves a relatively small average path length (APL) with a low network diameter. Table IV exhibits the result of APL comparisons between *JieLin*, Fat Tree and HyperBCube (k=2) with different network configurations. It can be seen that the APL of *JieLin* is approximately 20% and 40% smaller than that of Fat Tree and HyperBCube with a similar sized network. However, it also reveals that the APL will increase accordingly as the network size increases. Table V gives the detailed statistics of flows and routes for the case of 456-server *JieLin*.

TABLE IV
THE AVERAGE PATH LENGTH IN DIFFERENT ARCHITECTURES

	Fat Tree		HyperBCube		JieLin	
n	Servers	APL	Servers	APL	Servers	APL
4	16	5.47	64	6.10	52	3.96
8	128	5.72	512	7.01	456	4.49
16	1024	5.86	4096	7.50	3856	4.68
24	3456	5.91	13824	7.71	13272	4.99
48	27648	5.93	110592	7.82	108336	5.21
64	65536	5.96	262144	7.88	258112	5.36

TABLE V
THE STATISTICS OF FLOW AND ROUTES IN A 456-SERVER JIELIN

All-to-All Traffic Pattern & Exponential Distribution Flow Mode	
Flow Statistics (1373981 flows in total)	776598 flows using 456 level-0 links
	597383 flows using 456 level-1 links
Route Statistics (234232 routes in total)	35260 paths of length 2
	105980 paths of length 4
	92992 paths of length 6
Average Path Length	4.49

B. Aggregate Bottleneck Throughput

The aggregate bottleneck throughput (ABT) is usually used to evaluate the network capacity of an architecture under the all-to-all traffic pattern [4] [1]. Fig.4 compares the performance of ABT between *JieLin*, Fat Tree and HyperBCube with different network configurations. Intuitively, the ABT of the three architectures increases linearly as the number of servers increases. Also, the simulation results show that *JieLin* receives a much higher ABT than both Fat Tree and HyperBCube irrespective of the network size, and almost 50% higher than HyperBCube. Table VI illustrates the statistics of ABT and APL achieved by the three architectures with a 2000-server level sized network. According to the Equation 1 derived in Section III, given the APL of *JieLin*, Fat Tree and HyperBCube as 4.61, 5.9 and 7.4, their achieved NCP_{ABT} reaches 20.95%, 12.63%, 12.82%, respectively. It can be seen that the NCP_{ABT} of *JieLin* 20.95% is already very close to the theoretical limit 21.69%. This further proves the good performance of *JieLin* in the aggregate bottleneck throughput.

TABLE VI
THE STATISTICS OF ABT AND APL IN DIFFERENT ARCHITECTURES.

	ABT	NC_{links}	NCP_{ABT}	$\frac{1}{APL}$
JieLin n=14, 2562-server	2147	10248	20.95%	$\frac{1}{4.61}$=21.69%
Fat Tree n=22, 2662-server	2018	15972	12.63%	$\frac{1}{5.9}$=16.95%
HyperBCube n=14, 2744-server	1407	10976	12.82%	$\frac{1}{7.4}$=13.51%

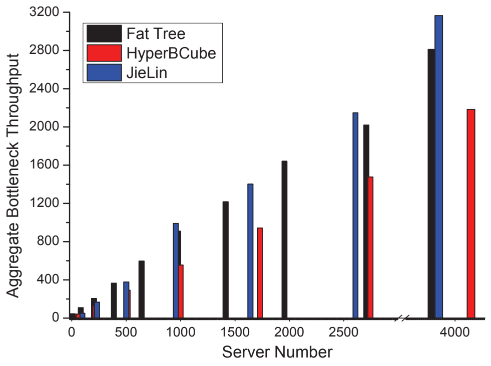


Fig. 4. The comparisons of ABT between different architectures.

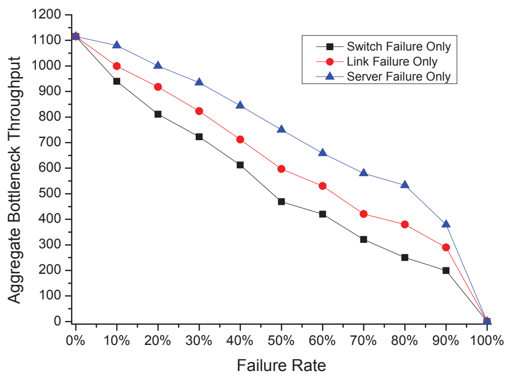


Fig. 5. The ABT performance of 1221-server *JieLin* under faulty conditions.

C. *Performance Under Faulty Conditions*

The robustness of a network not only depends on the interconnection architecture but also on the fault tolerant routing algorithm. The network faults usually result in a gradually decayed network performance as the failure rate increases. Fig.5 exhibits the performance of ABT under different link/server/switch failure ratios in a 1221-server *JieLin* network. Admittedly, the ABT of *JieLin* decreases as the failure ratio increases. However, with the benefit of a rich physical interconnection and fault tolerant routing algorithm *JieLin* still maintains a high ABT and largely outweighs Fat Tree. For example, as shown in Fig.4 a similar sized Fat Tree (1024-server) achieves an ABT of only 908.34 and NCP_{ABT} of 14.78% in a fault-free condition. Comparatively, in case of 50% server fails JieLin still achieves ABT of 750.06, and the NCP_{ABT} is $\frac{750.06}{2*2442} = 15.36\%$, correspondingly, which is even higher than that of a fault free Fat Tree.

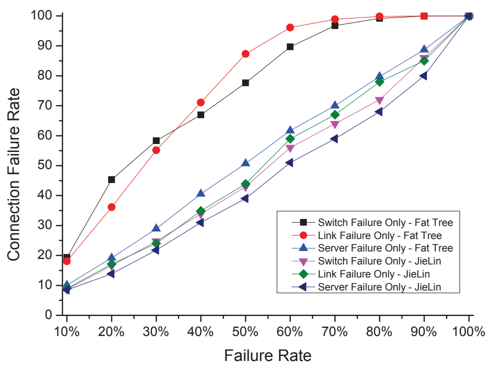


Fig. 6. The connection failure rate of 1596-server *JieLin* and Fat Tree under different kinds of failure rates.

Fig.6 demonstrates good fault tolerance of *JieLin* from another perspective, where it illustrates the connection failure ratios in case of different network faults in a 1596-server *JieLin*. Although *JieLin* experiences higher connection failure ratios for higher node (switch/link/server) failure ratios, JieLin claims a much better fault tolerance than Fat Tree. Besides, as shown in Fig.6 *JieLin* holds an upper bound of connection failure ratio, that is, if the node failure rate is f then the connection failure ratio will also not exceed f.

VI. Conclusion

This paper proposes a novel server-centric data center network architecture named *JieLin*. *JieLin* not only achieves an excellent scalability and good cost-effectiveness, but also demonstrates good overall network performance with respect to aggregate throughput, bisection bandwidth, network diameter, and fault tolerance. Furthermore, the specially designed congestion-aware adaptive routing algorithm CARA further strengthens these merits of *JieLin*. The simulation results bear witness to the feasibility and good performance of JieLin.

VII. Acknowledgment

This paper is supported in part by NPRP grant from the Qatar National Research Fund.

References

[1] T. W., et al. Rethinking the data center networking: Architecture, network protocols, and resource sharing. *IEEE Access*, 2014.
[2] M. A., et al. A scalable, commodity data center network architecture. In *ACM SIGCOMM Computer Communication Review*. ACM, 2008.
[3] C. Guo, et al. DCell: A scalable and fault-tolerant network structure for data centers. *ACM SIGCOMM*, 38(4):75–86, 2008.
[4] C. Guo, et al. BCube: A high performance, server-centric network architecture for modular data centers. *ACM SIGCOMM*, 2009.
[5] Albert Greenberg, et al. VL2: a scalable and flexible data center network. *SIGCOMM Comput. Commun. Rev.*, 39(4):51–62, 2009.
[6] D. Lin, Y. Liu, Mounir. Hamdi, and Jogesh Muppala. HyperBCube: A Scalable Data Center Network. IEEE ICC, 2012.
[7] T. Wang., et al. Sprintnet: A high performance server-centric network architecture for data centers. In *ICC*, pages 4005–4010. IEEE, 2014.
[8] T. W., et al. Novacube: A low latency torus-based network architecture for data centers. In *GLOBECOM, IEEE*, pages 2252–2257. IEEE, 2014.
[9] T. Wang, et al. Clot: A cost-effective low-latency overlaid torus-based network architecture for data centers. In *ICC*. IEEE, 2015.
[10] Farrington Nathan, et al. Data Center Switch Architecture in the Age of Merchant Silicon. In *IEEE Hot Interconnects*, New York, Aug 2009.
[11] R. Niranjan Mysore, et al. Portland: a scalable fault-tolerant layer 2 data center network fabric. *ACM SIGCOMM*, 2009.
[12] T. Wang, et al. Designing efficient high performance server-centric data center network architecture. *Computer Networks*, 79:283–296, 2015.
[13] N. Farrington, et al. Helios: a hybrid electrical/optical switch architecture for modular data centers. In *ACM SIGCOMM*, 2010.
[14] G. Wang, et al. c-Through: Part-time optics in data centers. In *ACM SIGCOMM*, 2010.
[15] H. Abu-Libdeh, et al. Symbiotic routing in future data centers. *ACM SIGCOMM Computer Communication Review*, 40(4):51–62, 2010.
[16] J. Shin, et al. Small-world datacenters. In *Proceedings of the 2nd ACM Symposium on Cloud Computing*, page 2. ACM, 2011.
[17] Y. Liu and J. Muppala. DCNSim: A data center network simulator. In *ICDCSW*. IEEE, 2013.