SprintNet: A High Performance Server-Centric Network Architecture for Data Centers

Ting Wang, Zhiyang Su, Yu Xia, Yang Liu, Jogesh Muppala, Mounir Hamdi
Department of Computer Science and Engineering
Hong Kong University of Science and Technology, Hong Kong
{twangah, zsuab, rainsia, liuyangcse, muppala, hamdi}@cse.ust.hk

***Abstract*—This paper presents the design, implementation and evaluation of *SprintNet*, a novel network architecture for data centers. *SprintNet* achieves high performance in network capacity, fault tolerance, and network latency. *SprintNet* is also a scalable, yet low-diameter network architecture where the maximum shortest distance between any pair of servers can be limited by no more than four and is independent of the number of layers. The specially designed routing schemes for *SprintNet* strengthen its merits. Both theoretical analysis and simulation experiments are conducted to evaluate its overall performance with respect to average path length, aggregate bottleneck throughput, and fault tolerance.**

***Keywords*—Data center, Network topology, Server-centric**

I. Introduction

Data centers, implemented as an agglomeration of massive number of servers, are increasingly playing an important role in supporting enterprise computing needs, cloud computing services (such as web search, email, online gaming, social networking, etc) and infrastructure-based services (like GFS [4], BigTable [2], MapReduce [3], Dryad [8], etc). To support the growing cloud computing needs, the number of servers in today's data centers are increasing exponentially, thus resulting in enormous challenges to efficient network design for interconnecting these servers. As a result, several novel proposals, such as Fat Tree [1], DCell [6], BCube [7], HyperBcube [9], Portland [14], FlatNet [10], have been proposed aiming to efficiently interconnect the servers inside a data center to deliver peak performance to users. The data center architecture is regarded as the most important factor, since it not only determines the reliability of a data center, but also plays a dominate role in network capacity, fault tolerance, latency, and routing efficiency. Generally, the design goals of data center networks are high scalability, good fault tolerance, low latency and high network capacity [6][12].

Scalability: In order to meet the increasing demands for services and better performance, the physical structure must have good scalability enabling incremental expansion without affecting the existing servers. Correspondingly, the routing algorithm should also be scalable and easily adapt to the new expanded interconnection.

Fault tolerance: A fault-tolerant architecture allows the system to continue with its current task even in the presence of failures. From the perspective of the network design, good fault tolerance can be achieved through redundant physical connections and fault-tolerant routing schemes.

Latency: Primarily the network latency consists of the queuing delay at each hop, transmission delay and propagation delay, of which the buffer queuing at each hop is the major contributor to latency. Therefore, a smaller network diameter, which leads to lowering the latency should be offered as a basic feature of the network design enabling the data center to provide faster services.

Network capacity: Large-scale data centers providing cloud services are usually bandwidth hungry. Hence, the data center should provide high network capacity to support the high volumes of traffic generated by many online infrastructure services, such as GFS [4] and MapReduce [3].

To meet these challenging design goals, in this paper, we propose a novel server-centric data center network architecture named *SprintNet*, which is recursively defined with rich connectivity, exhibits good fault tolerance and scalability. Furthermore, the diameter of *SprintNet* can be controlled to be within four, which implies potentially low network latency. Moreover, *SprintNet* also demonstrates very good performance in terms of bisection bandwidth and aggregate bottleneck throughput, compared to other data center network architectures.

The primary contributions of this paper can be summarized as follows:

1) A new high performance architecture enjoying four major desirable features for data center networks is proposed.
2) Theoretical analysis of typical features of *SprintNet* and compare with other proposals from various aspects.
3) Design of two routing schemes for *SprintNet* taking some practical issues into consideration.
4) Conduct extensive simulations to evaluate the performance of *SprintNet*.

The rest of the paper is organized as follows. First we briefly review the related research literature in Section II. Then Section III introduces the *SprintNet* structure in detail. Subsequently, the routing schemes designed for *SprintNet* are described in Section IV followed by the system evaluation and experimental results in Section V. Finally, Section VI concludes the whole paper.

II. Related Work

Considerable research has been conducted in finding a suitable interconnection architecture for data center networks.

The proposed data center architectures so far can be generally classified into two categories: server-centric and switch-centric. Unlike switch-centric scheme which places the interconnection and routing intelligence on switches, the server-centric scheme expects the servers also to forward packets.

A. Switch-Centric Architecture

Fat Tree [1] is a three-layer Clos network built in the form of multi-rooted tree. It is a rearrangeably non-blocking structure, which provides an oversubscription ratio of 1:1 to all servers. A Fat Tree built with *n*-port switches has *n* pods, each of which contain two layers of *n*/2 switches. It consists of $(n/2)^2$ core switches, $n^2/2$ aggregation and edge switches respectively, and supports $n^3/4$ servers in total. However, the wiring complexity is $O(n^3)$ which is a serious challenge.

VL2 [5] is an agile and cost effective network architecture, which is built from numerous switches arranged into a Clos topology. VL2 employs Valiant Load Balancing (VLB) to spread traffic across network paths, and uses address resolution to support large server pools. Besides, VL2 applies flat addressing to eliminate fragmentation of resources and allow any service to be assigned to any server anywhere in the data center. However, the directory system may become a bottleneck in the case of heavy network load.

B. Server-Centric Architecture

DCell [6] uses servers with multiple ports and low-end mini-switches to build its recursively defined architecture. Typically, DCell$_k$ (i.e. level-*k* DCell) is created by $t_{k-1}+1$ DCell$_{k-1}$s, where t_{k-1} is the number of servers in each DCell$_{k-1}$. The node degree of each server in a DCell$_k$ is $k+1$. DCell scales out at a double exponential speed, and its fault-tolerant DFR routing protocol which introduces local-reroute achieves good results. However, the lower level links carry more traffic causing higher link utilization, thus may become a bottleneck and result in low aggregate bottleneck throughput.

BCube [7] is also a recursively defined structure, which is specially designed for shipping container based modular data centers. BCube$_k$ is derived from *n* BCube$_{k-1}$s and n^k *n*-port COTS switches. Different from DCell, the servers in BCube only connect to switches without connecting other servers. BCube accelerates 1-to-n traffic patterns and also demonstrates a good network capacity for all-to-all network traffic. However, BCube is deficient in scalability with relatively high wiring complexity.

III. SPRINTNET NETWORK STRUCTURE

In this section, we first describe the *SprintNet* architecture that interconnects commodity switches and servers in line with server-centric scheme, and then present its key properties.

A. SprintNet Physical Structure

SprintNet is recursively constructed, where a high-level *SprintNet* is built from a certain number of lower-level *SprintNets*. The basic building unit is named $Cell$ (or 0-layer), which is the building block to construct a larger *SprintNet*. Each $Cell$ is constructed with c n-port switches, where $\frac{c}{c+1}n$ ports

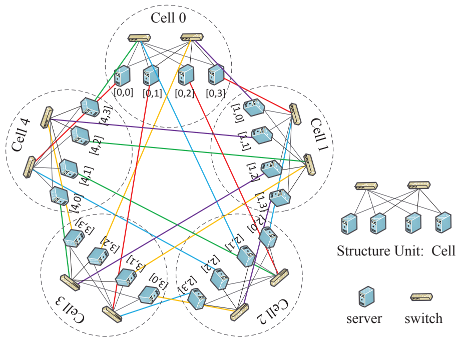

Fig. 1. A 20-server *SprintNet* using two 6-port switches in each Cell.

of each switch connect to $\frac{c}{c+1}n$ servers and $\frac{1}{c+1}n$ ports for inter-$Cell$ connections. Accordingly, each $Cell$ contains c n-port switches and $\frac{c}{c+1}n$ servers. All the switches and servers are fully-connected.

The 1-layer *SprintNet* consists of $\frac{c}{c+1}n+1$ $Cell$s, and supports $(\frac{c}{c+1})^2n^2+\frac{c}{c+1}n$ servers in total. Each server has $c+1$ ports, c ports of which connect to c switches inner $Cell$ and one port is used for inter-$Cell$ connection which connects to a switch in another $Cell$.

The higher k-layer *SprintNet* is constructed by adding $\frac{c}{c+1}n$ $Cell$s each time and fully connected to each other in the same way. There will be $k\times\frac{cn}{c+k}+1$ different $Cell$s. $\frac{cn}{c+k}$ ports of each n-port switch connect to $\frac{cn}{c+k}$ servers within a $Cell$, and $\frac{kn}{c+k}$ ports interconnects different $Cell$s. As a result, a k-layer *SprintNet* can support $k\times(\frac{c}{c+k})^2n^2+\frac{c}{c+k}n$ servers.

Fig.1 shows an example of a 20-server *SprintNet* constructed by using 6-port switches when $c=2$, $n=6$ and $k=1$. Each $Cell$ is composed of 2 switches and 4 servers.

B. Properties of SprintNet

In this subsection, some typical features of *SprintNet* are explored and analyzed. The analysis of the structural properties of *SprintNet* are summarized in Table I, which also presents some comparison with other network architectures.

TABLE I
THE COMPARISON BETWEEN DIFFERENT NETWORK ARCHITECTURES

	Fat Tree (3 layers)	DCell (2 layers)	BCube (2 layers)	SprintNet (2 layers)
Servers Number	$\frac{n^3}{4}$	$n(n+1)$	n^2	$(\frac{c}{c+1})^2n^2+\frac{c}{c+1}n$
Links Number	$\frac{3n^3}{4}$	$\frac{3n(n+1)}{2}$	$2n^2$	$\frac{c^2n^2}{c+1}+cn$
per Server	3	$\frac{3}{2}$	2	≥ 2
Switches Number	$\frac{5n^2}{4}$	$n+1$	$2n$	$\frac{c^2}{c+1}n+c$
per Server	$\frac{5}{n}$	$\frac{1}{n}$	$\frac{2}{n}$	$\frac{c+1}{n}$
Bisection Bandwidth	$\frac{n^3}{8}$	$\frac{n^2}{4}+\frac{n}{2}$	$\frac{n^2}{2}$	$\frac{c^2n^2}{2(c+1)^2}+cn$
per Server	$\frac{1}{2}$	$\approx\frac{1}{4}$	$\frac{1}{2}$	$\frac{1}{2}+\frac{(2c+1)(c+1)}{2(cn+c+1)}$
Network Diameter	6	5	4	4
Number of Node-disjointed Paths	0	2	2	$c+1(\geq 2)$

1) Diameter

The diameter indicates the maximum shortest path length (denoted by the number of links) among all the server pairs. Compared with other architecture proposals as shown in Table

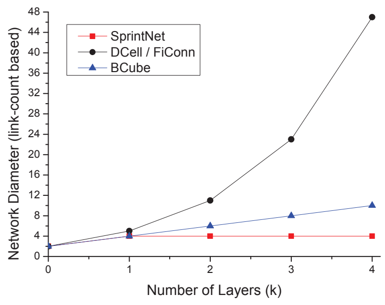

Fig. 2. The network diameters of various architectures with different layers.

I, *SprintNet* stands out because of its low network diameter. Owing to the full-connection between servers and switches in a $Cell$, the path length between any two servers within a $Cell$ is 2. The distance between any pair of servers in different $Cells$ is 2 or 4. Therefore, the network diameter of *SprintNet* can be restricted by 4, which is a constant number. Comparatively, the network diameters of other architectures are much higher, some of which (e.g. DCell, BCube) increase accordingly as the number of layers increases, as shown in Fig.2.

Furthermore, *SprintNet* has another competitive edge derived from its low network diameter - the potential low network latency, where smaller diameter leads to more effective routing and lower transmission latency in practice [12].

2) Scalability and Physical Cost

For a 2-layer architecture, *SprintNet* achieves scalability of $O(n^2)$ which is the same as DCell and BCube. The wiring complexity of *SprintNet* is slightly higher than DCell and BCube though they are all at the same level of $O(n^2)$. However, this is still better than Fat Tree whose wiring complexity is $O(n^3)$. Additionally, the number of switches used in the network depends on the value of c and increases at the speed of $O(n)$ which also outperforms Fat Tree.

Thus, to sum up the above analysis, though the scalability and link complexity of *SprintNet* are not superior to the others, yet still can be acceptable.

3) Bisection Bandwidth

The bisection bandwidth is depicted as the sum of link capacities between two equally-sized parts which the network is partitioned into. It can be used to measure the worst-case network capacity [13].

The bisection bandwidth of *SprintNet* is $\frac{c^2n^2}{2(c+1)^2} + cn$, which is around two times that of DCell's. Moreover, it can be seen from Table I that the bisection bandwidth per server also outperforms the other candidates.

4) Hardware Fault Tolerance

Benefitting from the well designed interconnection with rich physical connections, *SprintNet* enjoys very good fault tolerance. As shown in Table I, compared with an architecture with two layers, *SprintNet* has $c+1$ ($c \geq 1$) parallel node-disjoint paths, which is better than DCell and BCube, while Fat Tree has no node-disjoint paths. Apart from the sufficient physical connections, the fault tolerant routing scheme designed in Section IV further strengthens the reliability of *SprintNet*.

5) Incremental Expansion

SprintNet can be easily expanded by adding $\frac{cn}{c+1}$ $Cells$ each time which account for $(\frac{cn}{c+1})^2$ more servers. Besides, *SprintNet* is a highly symmetric architecture, hence missing one or more $Cells$ of servers does not degrade the performance of the system much, and the remaining $Cells$ are still fully connected.

C. Towards a Cost-effective Structure

As described above, the total number of servers depends on n and c. If given a certain number of servers, a larger c implies better fault tolerance and smaller n, but at the cost of more switches. If considering the cost of the device, high-port-count (larger n) switches are more expensive than low-port-count ones , but a larger n leads to less switches (e.g. supporting 24 servers in each Cell should use two 36-port switches or only one 48-port switch). In order to find the most beneficial c and n so as to make a reasonable tradeoff between the cost of switches and fault tolerance, we formulate this problem into an optimization model as below (for simplicity, without loss of generality, we consider the case of *SprintNet* with two layers).

$$Objective: \ Minimize \ \ f(c,n) = \frac{Cost(switches)}{FT} \tag{1}$$

$$Subject \ \ to:$$

$$Cost(switches) = N_{switch} * Price_{n\text{-}port\text{-}switch}$$

$$= (\frac{c^2}{c+1} * n + c) * Price_{n\text{-}port\text{-}switch} \tag{2}$$

$$FT = c+1 \tag{3}$$

$$(\frac{c}{c+1})^2 * n^2 + \frac{c}{c+1} * n = N_{server} \tag{4}$$

where $Cost(switches)$ denotes the total cost of switches, FT means fault tolerance indicating the number of parallel node-disjointed paths, N_{switch} is defined as the total number of switches, $Price_{n\text{-}port\text{-}switch}$ gives the price of a n-port switch, and N_{server} represents the total number of servers the network can support.

The goal is to achieve as high fault tolerance as possible with the minimum cost of switches, which leads to a minimal ratio $f(c,n)$. The solution to Eq.(1) is provided as follows:

$$\frac{Cost(switches)}{FT} = \frac{(\frac{c^2}{c+1} * n + c) * Price_{n\text{-}port\text{-}switch}}{c+1}$$

$$= (\frac{c^2}{(c+1)^2} * n + \frac{c}{c+1}) * Price_{n\text{-}port\text{-}switch}$$

$$= (\frac{c^2}{(c+1)^2} * n^2 + \frac{c}{c+1} * n) * \frac{Price_{n\text{-}port\text{-}switch}}{n}$$

$$= N_{server} * \frac{Price_{n\text{-}port\text{-}switch}}{n}$$

Given a certain N_{server}, the $Price_{n\text{-}port\text{-}switch}$ is exponential or in direct proportion to n, so we can compute the most cost-effective n based on the actual prices of the devices, and futher obtain the corresponding c, where

$$c = \frac{\sqrt{4N_{server}+1}-1}{2n-\sqrt{4N_{server}+1}+1}.$$

This provides a satisfactory trade-off between cost and reliability to construct a cost-effective N_{server} sized *SprintNet*.

IV. Routing In SprintNet

This section presents specially designed routing algorithms for *SprintNet*, which aim to help *SprintNet* achieve its maximum theoretical performance.

A. Naïve Routing Scheme

The naïve routing is a shortest path routing scheme. It is required to compute the shortest paths for all the servers to every destination and store the routing table on each node. A node may use broadcasting based technique to compute the routing table, or it may resort to the already existing protocols, such as the OSPF (Open Shortest Path First) protocol.

The principle of the broadcasting-based method is simple. In order to compute the shortest path from server a to b, starting from server a the broadcasting is recursively carried out in the network. When the broadcasting packet arrives at server b, then it returns to server a along the reverse path reaching b. In this way, several paths between server a and b may be obtained, then the shortest one will be chosen as the final routing path. When there is more than one shortest path, the route will be chosen at random from the candidates, which can positively contribute to the load balancing of the system to some extent. The routing tables can be pre-computed and then in the future used directly at the complexity of $O(1)$.

Although this routing scheme is simple in implementation and can achieve the optimal shortest path, there are some implicit shortcomings. On the one hand, the broadcasting increases the network workload resulting in additional bandwidth cost. On the other hand, the size of the routing table stored on each node is linear to the size of the data center, which may become a heavy burden for the limited memory resource and TCAM. Finally, naïve routing is not traffic-aware and the routing decisions are made without regard to the network state. This may lead to poor link utilization and even congestion. With response to these issues, we propose the Traffic-aware Adaptive Routing scheme.

B. Traffic-aware Adaptive Routing Scheme

The traffic-aware adaptive routing (TAR) is a customized fault tolerant routing scheme for *SprintNet*, also based on shortest path routing. TAR takes the network state into consideration when making routing decisions so as to avoid network congestion and achieve good load balancing. Moreover, TAR scheme follows the way of flow-based single path routing which does not divide flows among multiple paths for the sake of avoiding packet reordering.

In the TAR scheme, each server is identified using two coordinates (cid, sid), which indicates the sid-th server in the cid-th $Cell$. Given a pair of servers $src[s_1, s_2]$ and $dst[d_1, d_2]$, a route between them with the path length of two or four can be computed according to Algorithm 1.

Algorithm 1 Traffic-aware Adaptive Routing: TAR(src, dst)

```
if $src.fail()||dst.fail()$ then
    return null;
else
    if $s_1$ == $d_1$ then
        return Route0: $src \to switch\ x \to dst$;
    else
        if $src$ == $n$ && $dst$ != $m$ && $j.on()$ then
            return Route1: $src \to switch\ j \to dst$;
        else if $src$ != $n$ && $dst$ == $m$ && $i.on()$ then
            return Route2: $src \to switch\ i \to dst$;
        else if $src$ == $n$ && $dst$ == $m$ then
            if $link(src,i).abw > link(src,j).abw$ then
                return Route1;
            else
                return Route2;
            end if
        else
            if $i.on()$ && $j.on()$ && $m.on()$ && $n.on()$ then
                Route3: $src \to switch\ i \to m \to switch\ y \to dst$; Route4:
                $src \to switch\ x \to n \to switch\ j \to dst$; return Route3 or
                Route4 at random;
            else if ($i.fail()$ || $m.fail()$) && $n.on()$ && $j.on()$ then
                return Route4;
            else if ($n.fail()$ || $j.fail()$) && $i.on()$ && $m.on()$ then
                return Route3;
            else if $(i.fail()||m.fail())\&\&(n.fail()||j.fail())$ then
                return Route5: $src \to intermediate\ Cell_t \to dst$;
            end if
        end if
    end if
end if
```

The whole routing procedure can be generally divided into two cases as shown in the pseudocode of Algorithm1. Firstly, if the source $src[s_1, s_2]$ and destination $dst[d_1, d_2]$ are located in the same $Cell$ (s_1=d_1), then they can directly reach each other via any switch within the $Cell$. In addition, switch x whose link connecting to server src has the most available bandwidth is preferentially selected. From another perspective, when a switch or port fails, its link bandwidth can be regarded as zero, so the traffic can be routed by other switches to handle network faults. Secondly, src and dst are placed in two different $Cell$s: $Cell_{s_1}$ and $Cell_{d_1}$. There are four situations for this case. We assume that switch i of $Cell_{s_1}$ connects to server $m[m_1, m_2]$ of $Cell_{d_1}$, and server $n[n_1, n_2]$ of $Cell_{s_1}$ connects to switch j of $Cell_{d_1}$. The first situation is that src is right the server n, then the traffic will traverse the path Route1: $src \to switch\ j \to dst$. When switch j fails, it takes Route3: $src \to switch\ i \to m \to switch\ y \to dst$ instead to circumnavigate the fault, where y is satisfied that link (m, y) has the most available bandwidth. If even switch i also fails, then it will firstly route to the switch, which connects to src, of an intermediate $Cell_t$ then finally transfer to dst. The second situation is that the destination server dst is just the server m, and as described in Algorithm 1 its routing strategy is similar to the first situation. The third situation happens when $src = n$ and at the same time $dst = m$, then we will choose the route with the most available bandwidth from Route1 and Route2 aiming to avoid network congestion and achieve lowest

TABLE II
ROUTING PATH LENGTH DISTRIBUTION

Route	Route0	Route1	Route2	Route3	Route4	Route5
Path Length	2	2	2	4	4	4

latency. The last situation mainly deals with the faulty cases when $src \neq n$ and $dst \neq m$. If all the interconnection points (i.e. switch $i,\ j$ and server $m,\ n$) connecting the $Cell_{s_1}$ and $Cell_{d_1}$ are available, then the algorithm chooses Route3 or Route4 at random trying to contribute to load balancing. If in case one endpoint of the interconnection link (i, m) fails, then Route4 will be picked as the alternative route. Similarly, Route3 will be selected for the case of link (n, j) failure. The worst case is that all the interconnection points i, j, m, n become unavailable, in this case the traffic will be rerouted to the destination via an intermediate $Cell_t$.

Following the above procedures strictly, the traffic will be forwarded along the shortest path with the most available bandwidth and be fault tolerant. The statistics of path length distribution for all the TAR routing cases are summarized as in Table II, which reveals that theoretically in any case the distance between any pair of servers in a *SprintNet* can be restricted within four. By comparison, the maximum path lengths of Ficonn_k and BCube_k are $2 * 3^k - 1$ and $2k + 2$ respectively, which are much longer than *SprintNet*. Therefore, *SprintNet* is a comparatively low-diameter network.

V. SYSTEM EVALUATION

In this section, extensive simulations are conducted to evaluate the performance of *SprintNet* under various network conditions using the TAR routing scheme. The all-to-all traffic pattern, which simulates the most intensive network activities, is used to evaluate the guaranteed performance under the most rigorous cases. Besides, the uniform distribution flow mode is applied to determine the interval time of packet generation. All simulations are conducted using our DCNSim [11] simulator. DCNSim can simulate several data center network topologies and compute various metrics, such as throughput, network latency, average path length, fault-tolerance, and so on. We implement *SprintNet* architecture and its traffic-aware routing algorithm in DCNSim to enable conducting comprehensive evaluation.

A. Experimental Results

This subsection presents the experimental results of the *SprintNet* evaluation. In order to better illustrate the overall performance of this architecture, the simulations are conducted from the following three aspects.

1) Average Path Length

In order to evaluate the overall performance of the whole network, the link-count based average path length (APL) is used, where APL has a great impact on packet delay. Applying uniform distribution flow mode and all-to-all traffic pattern, each server communicates with all other servers in the way of bilateral communication. Table III demonstrates the simulation results about APL for eight different sized 2-layer *SprintNet*s and DCells. And Table IV illustrates the detailed statistics

TABLE III
THE AVERAGE PATH LENGTH STATISTICS FOR 2-LAYER ARCHITECTURES

Network Size -Number of Servers-	Average Path Length		Diameter	
	SprintNet	DCell	SprintNet	DCell
20 (n_s=6,c=2,n_d=4)	2.95	3.68	4	5
72 (n_s=12,c=2,n_d=8)	3.38	4.25	4	5
156 (n_s=18,c=2,n_d=12)	3.56	4.48	4	5
272 (n_s=24,c=2,n_d=16)	3.66	4.60	4	5
600 (n_s=30,c=2,n_d=24)	3.77	4.72	4	5
1056 (n_s=48,c=2,n_d=32)	3.82	4.79	4	5
1332 (n_s=48,c=3,n_d=36)	3.84	4.81	4	5
2352 (n_s=64,c=3,n_d=48)	3.88	4.86	4	5

*n_s and c denote the number of n_s-port switches per Cell in *SprintNet* is c. n_d indicates n_d-port switch used in DCell

TABLE IV
THE STATISTICS FOR THE CASE OF 1056-SERVER SPRINTNET

All-to-All Traffic Pattern & Uniform Distribution Flow Mode	
Flow Statistics	3176448 flows using 2112 level-0 links
(4257792 flows in total)	1081344 flows using 1056 level-1 links
Route Statistics	99264 paths of length 2
(1114080 routes in total)	1014816 paths of length 4
Average Path Length	3.82

of flows and routes for the case of 1056-server *SprintNet*. It can be seen from Table III that *SprintNet* owns shorter APL comparing with DCell regardless of the network size. Besides, the experiment also reveals that the APL varies slightly for different sizes of *SprintNet*, and in general a larger sized *SprintNet* holds longer APLs.

TABLE V
THE ABT OF 2352-SERVER SPRINTNET AND DCELL

All-to-All Traffic Pattern (2352-server network)				
	ABT	NC_links	NCP_ABT	$\frac{1}{APL}$
SprintNet (n=64, c=3)	4762	18816	25.31%	$\frac{1}{3.88} = 25.77\%$
DCell (n=48)	1213	7056	17.19%	$\frac{1}{4.86} = \%20.58$

2) Aggregate Bottleneck Throughput

The aggregate bottleneck throughput (ABT) can be used to measure the overall network capacity of an architecture under the all-to-all traffic pattern, where every server communicates with all other servers. For each server, among all of its flows on different routes, the flows with the smallest throughput are named as bottleneck flows. And ABT indicates the sum of the throughputs of all the bottleneck flows.

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

$$NCP_{ABT} = \frac{1}{APL} \tag{5}$$

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

$Proof$: 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

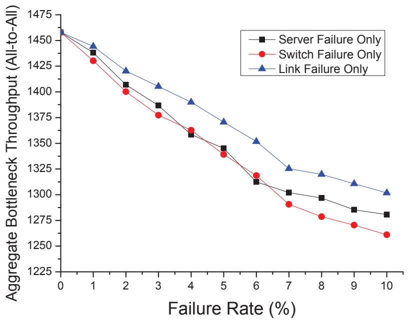

Fig. 3. The ABT performance of a 1056-server *SprintNet* under faulty conditions with TAR routing scheme.

the throughput that per flow receives), then we have:

$$ABT = N_{flows} * \frac{1}{NF_{link}} \tag{6}$$

$$NF_{link} = \frac{N_{flows} * APL}{2N_{links}} \tag{7}$$

Combining Equation (6) and (7), we have:

$$ABT = \frac{2N_{links}}{APL} \tag{8}$$

Hence, when the link capacity is 1, there is

$$NCP_{ABT} = \frac{ABT}{NC_{links}} = \frac{ABT}{2N_{links}} = \frac{1}{APL} \tag{9}$$

which concludes the proof.

Table V presents the experimental results of ABT for 2353-server *SprintNet* and DCell under all-to-all traffic pattern. The result shows that *SprintNet* achieves both higher ABT and NCP_{ABT} than DCell. Take *SprintNet* for example, the N_{links} of a 2353-server *SprintNet* is 9408, hence its NC_{links} is 9408*2=18816 (two-way communication). Given the APL of a 2352-server *SprintNet* is 3.88, its ABT reaches 4762, therefore its $NCP_{ABT} = \frac{4762}{18816} = 25.31\%$, which is already very close to its theoretical limit ($\frac{1}{APL} = \frac{1}{3.88} = 25.77\%$). This reveals the great performance of the *SprintNet* in the aggregate bottleneck throughput.

3) Performance Under Faulty Conditions

Fig. 3 and Fig. 4 demonstrate the performance of a 1056-sever *SprintNet* in ABT and APL under various faulty conditions applying the Traffic-aware Adaptive Routing scheme. Although the ABT degrades as the failure rate increases, as shown in Fig. 3, the network still achieves 86.5%, 87.9%, and 89.3% of the fault-free ABT (failure rate = 0%) when the switch/server/link failure rate reaches 10% respectively. The ABT decreases most for the switch failure case, where its ABT reduces to 1260.98 for 10% failure rate. However, its NCP_{ABT} still reaches 19.9% ($\frac{ABT}{NC_{links}} = \frac{1260.98}{2*3168} = 19.9\%$), which is even better than the DCell of its fault-free case (the ABT for 1056-server fault-free DCell is 553.40, so there is $NCP_{ABT} = \frac{ABT}{NC_{links}} = \frac{553.40}{2*1584} = 17.5\%$). This further convinces the theoretical analysis about the good performance of *SprintNet* in ABT.

The experimental results illustrated in Fig.4 reveal that the APL varies slightly and remains almost the same under

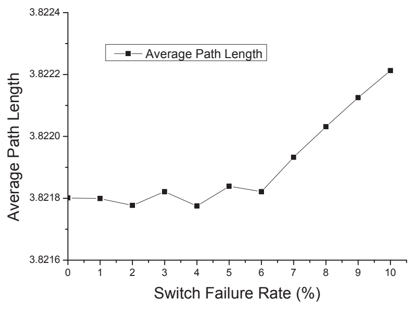

Fig. 4. The APL performance of a 1056-server *SprintNet* under faulty conditions with the TAR routing scheme.

different switch failure rates. For example, the APL under a fault-free condition is only 3.8218, and increases very little as the failure rate increases. The peak value 3.8222 appears in the 10% switch failure rate case, which only increases by 0.01% compared with the fault-free case.

VI. Conclusion

In this paper we proposed *SprintNet*, a novel server-centric architecture for data centers, and presented its design, analysis and evaluations. *SprintNet* is highlighted by its low diameter, high bisection bandwidth, good fault tolerance, and high aggregate bottleneck throughput. The specially designed traffic-aware adaptive and fault tolerant routing scheme helps *SprintNet* achieve its maximum theoretical performance. The implementation and evaluation of *SprintNet* further strengthens the theoretical analysis and demonstrates its feasibility.

Acknowledgment

This research has been supported by a grant from the Research Grants Council (RGC) with project number 612912.

References

[1] M. A., et al. A scalable, commodity data center network architecture. In *ACM SIGCOMM Computer Communication Review*. ACM, 2008.
[2] Fay Chang, et al. Bigtable: A distributed storage system for structured data. *ACM Transactions on Computer Systems (TOCS)*, 26(2):4, 2008.
[3] Jeffrey Dean, et al. MapReduce: simplified data processing on large clusters. In *OSDI'04*, Berkeley, CA, USA, 2004. USENIX Association.
[4] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. The Google file system. *SIGOPS Oper. Syst. Rev.*, 37(5):29–43, 2003.
[5] Albert Greenberg, et al. VL2: a scalable and flexible data center network. *SIGCOMM Comput. Commun. Rev.*, 39(4):51–62, 2009.
[6] C. Guo, et al. DCell: A scalable and fault-tolerant network structure for data centers. *ACM SIGCOMM*, 38(4):75–86, 2008.
[7] Chuanxiong Guo, et al. Bcube: a high performance, server-centric network architecture for modular data centers. *ACM SIGCOMM*, 2009.
[8] Michael Isard, et al. Dryad: distributed data-parallel programs from sequential building blocks. In *EuroSys '07*, NY, USA, 2007. ACM.
[9] D. Lin, Y. Liu, Mounir. Hamdi, and Jogesh Muppala. HyperBCube: A Scalable Data Center Network. IEEE ICC, 2012.
[10] Dong Lin, Yang Liu, Mounir Hamdi, and Jogesh Muppala. Flatnet: Towards a flatter data center network. In *Global Communications Conference (GLOBECOM), 2012 IEEE*, pages 2499–2504. IEEE, 2012.
[11] Y. Liu and J. Muppala. DCNSim: A data center network simulator. In *ICDCSW*. IEEE, 2013.
[12] Yang Liu, Jogesh Muppla, Malathi Veeraraghavan, Dong Lin, and Mounir Hamdi. Data center networks. http://www.amazon.ca/Data-Center-Networks-Yang-Liu/dp/3319019481, 2013.
[13] Farrington Nathan, et al. Data Center Switch Architecture in the Age of Merchant Silicon. In *IEEE Hot Interconnects*, New York, Aug 2009.
[14] R. Niranjan Mysore, et al. Portland: a scalable fault-tolerant layer 2 data center network fabric. *ACM SIGCOMM*, 2009.