Computer Networks 79 (2015) 283–296

Contents lists available at ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet

Designing efficient high performance server-centric data center network architecture

Ting Wang *, Zhiyang Su, Yu Xia, Jogesh Muppala, Mounir Hamdi

Department of Computer Science and Engineering, Hong Kong University of Science and Technology, Clear Water Bay, Kowloon, Hong Kong

ARTICLE INFO

Article history:
Received 28 August 2014
Received in revised form 26 December 2014
Accepted 13 January 2015
Available online 29 January 2015

Keywords:
Data center network
Network topology
Interconnection architecture
Server-centric
Flat network

ABSTRACT

Data center network (DCN) architecture is regarded as one of the most important determinants of network performance. As the most typical representatives of DCN architecture designs, the server-centric scheme stands out due to its good performance in various aspects. In this paper, we firstly present the design, implementation and evaluation of *SprintNet*, a novel server-centric 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. However, all of these kind of server-centric architectures still suffer from some critical shortcomings owing to the server's responsibility of forwarding packets. With regard to these issues, in this paper, we then propose a hardware based approach, named "Forwarding Unit" to provide an effective solution to these drawbacks and improve the efficiency of server-centric architectures. Both theoretical analysis and simulations are conducted to evaluate the overall performance of *SprintNet* and the Forwarding Unit approach with respect to cost-effectiveness, fault-tolerance, system latency, packet loss ratio, aggregate bottleneck throughput, and average path length. The evaluation results convince the feasibility and good performance of both *SprintNet* and Forwarding Unit.

1. 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, and social networking) and infrastructure-based services (like GFS [9], BigTable [6], MapReduce [7], Dryad [14], etc.). With data availability and security at stake, the role of the data center is more critical than ever. 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 network design for interconnecting these servers. As a result, several novel proposals, such as Fat Tree [3], DCell [12], BCube [11], VL2 [10], FiConn [15], Portland [20], FlatNet [17], HyperBcube [16], NovaCube [24], CamCube [2] and Small-World [22] 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

* Corresponding author.
E-mail addresses: twangah@cse.ust.hk (T. Wang), zsuab@cse.ust.hk (Z. Su), rainsia@cse.ust.hk (Y. Xia), muppala@cse.ust.hk (J. Muppala), hamdi@cse.ust.hk (M. Hamdi).

http://dx.doi.org/10.1016/j.comnet.2015.01.006
1389-1286/

of data center networks are high scalability, good fault tolerance, low latency and high network capacity [23,12,19].

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. However, the richly connected network may be confronted with a high wiring complexity, where a satisfactory compromise should be made.

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. Besides, the design of traffic-aware routing algorithm or deadline-aware transport protocol also help reducing the overall latency.

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 [9] and MapReduce [7].

Aiming 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 limited within four, which implies potentially low network latency. Moreover, *SprintNet* also demonstrates high performance in terms of bisection bandwidth and aggregate bottleneck throughput.

Compared with other architectures (e.g. switch-centric architecture), the server-centric network scheme stands out owing to its own many superiorities. Firstly, the level-based and recursively defined architectures are more likely to achieve a good scalability and easier to design a large-scale DCN (e.g. DCell scales double exponentially and FlatNet scales at $O(n^3)$ with only 2 layers of network using n-port switches). Secondly, it achieves better performance in aggregate throughput and average packet delay for small sized network [5]. Thirdly, the server-centric architectures are more cost-effective than other candidates (e.g. Fat Tree) [21]. Fourthly, it holds good fault-tolerance because of numerous node-disjointed paths [12,11,19]. Lastly but not least, it only uses cheaper low-end commodity switches without any modifications to them.

Although the server-centric scheme receives numerous highlights, there still exist many practical issues which are enumerated in Section 5. In responding to these issues, this paper then provides an effective approach which can solve these shortcomings and improve the efficiency of server-centric architectures including *SprintNet*.

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

(1) A new high performance server-centric architecture *SprintNet* enjoying four major desirable features for data center networks is proposed.
(2) Theoretical analysis of typical features of *SprintNet* and a comprehensive comparison with other proposals from various aspects.
(3) Design of two routing schemes for *SprintNet* taking some practical issues into consideration.
(4) Address the critical shortcomings which exist and need to be solved in server-centric data center network architectures.
(5) Propose a hardware based approach to achieve a logically flat DCN and solve the issues of server-centric architectures.
(6) Conduct extensive simulations to evaluate the performance of *SprintNet* and the feasibility of the hardware based approach.

The rest of the paper is organized as follows. First we briefly review the related research literature in Section 2. Then Section 3 introduces the *SprintNet* structure in detail. Subsequently, the routing schemes designed for *SprintNet* are described in Section 4. Afterwards, Section 5 addresses the existing drawbacks of server-centric architectures followed by Section 6 which illustrates the hardware based solution to these drawbacks. Section 7 demonstrates the evaluations of *SprintNet* and forwarding unit. Finally, Section 8 concludes the paper.

2. 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. The server-centric architectures well implement the network *locality* forming the servers in close proximity of each other, which can help increase the communication efficiency. DCell [12], BCube [11], FiConn [15], FlatNet [17], CamCube [2], Small-World [22] and NovaCube [24] are examples of *server-centric* architectures. The switch-centric architectures are typically constructed around top of rack (ToR) switches, which are then interconnected through one or more higher levels of switches (such as aggregation level and core level). Fat Tree [3], Portland [20], and VL2 [10] are some typical representatives of switch-centric architectures.

2.1. Switch-centric architecture

Fat Tree [3] 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 $n^2/2$ edge switches respectively, and supports $n^3/4$ servers in total. However, the wiring complexity is $O(n^3)$ which is a serious challenge.

Portland [20] is a scalable, easily manageable, fault-tolerant, and efficient layer 2 Ethernet-compatible routing, forwarding and address resolution protocol for data center environments. It uses a pre-determined Fat-Tree topology, hierarchical pseudo-MAC (PMAC) address, and logically centralized directory (Fabric Manager) to resolve PMAC to actual MAC (AMAC) addresses to enable Ethernet-compatible plug-n-play networking. Besides, based on multi-rooted Fat-Tree architecture, it exploits the knowledge of the underlying topology of a data center, and completely avoids broadcast-based mechanisms, and stresses on fast and easy virtual machine mobility within the data center.

VL2 [10] 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 centralized directory system may become a bottleneck in the case of heavy network load.

2.2. Server-centric architecture

DCell [12] uses servers with multiple ports and low-end mini-switches to build its recursively defined architecture. In DCell, the most basic element named $DCell_0$ consists of n servers and one n-port switch. Each server in a $DCell_0$ is connected to the switch in the same $DCell_0$. Generally, $DCell_k$ (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 $k+1$, and the level-i link connects to a different $DCell_{i-1}$ within the same $DCell_i$. 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 [11] is also a recursively defined structure, which is specially designed for shipping container based modular data centers. Its most basic element $BCube_0$ is the same as $DCell_0$: n servers connect to one n-port switch. While constructing a $BCube_1$, n extra switches are used, connecting to exactly one server in each $BCube_0$. More generally, $BCube_k$ 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.

FiConn [15] shares similar design principle as DCell. The difference is that the degree of each server in FiConn is always two while in DCell it is $k+1$. Likewise, a high-level FiConn is derived from number of low-level FiConns. And the routing algorithm in FiConn is traffic-aware which can help it in utilizing the link capacities referring to traffic states, resulting in a good aggregate throughput. However, the fault tolerance and network capacity in FiConn are not so good. Besides, the average path length is relatively long.

FlatNet [17] is a 2-layer server-centric architecture, which scales at a speed of n^3. Briefly, the first-layer of FlatNet consists of n servers which connect to one n-port switch. And the second-layer of FlatNet is constructed by n^2 1-layer FlatNets. Different subsystems (1-layer Cells) are interconnected to each other using extra n^2 n-port switches.

Another special server-centric architecture proposed by researchers with great novelty is to directly connect servers to other servers. This is a switchless network interconnection without any switches, routers, or other network devices. CamCube [2], Small-World [22] and NovaCube [24], which are built based on a regular pattern, for example cube and torus, can be classified into this category. The CamCube is designed targeting at shipping container-sized data centers. With the benefit of Torus architecture and the flexibility offered by CamCube API, it allows applications to implement their own routing protocols so as to achieve better application-level performance. However, this kind of design consistently suffers from poor routing efficiency compared to other designs and this was mainly due to the relatively long routing paths, for example, the network diameter in 3D torus is $O(N^{1/3})$ hops within a N-server sized network.

3. SprintNet network structure

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

3.1. 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 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 $\frac{c}{c+1}n+1$ *Cells*, 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 $\frac{c}{c+1}n$ *Cells* each time and fully connected to each other in the same way. There will be $k\times\frac{cn}{c+k}+1$ different *Cells*. $\frac{cn}{c+k}$ ports of each n-port switch connect to $\frac{cn}{c+k}$ servers within

a *Cell*, and $\frac{kn}{c+k}$ ports interconnects different *Cells*. As a result, a k-layer *SprintNet* can support $k \times (\frac{c}{c+k})^2 n^2 + \frac{c}{c+k} n$ 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.

3.2. 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 1, which also present comparisons with other network architectures from different aspects.

3.2.1. Network 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 1, *SprintNet* achieves a lower network diameter. The physical interconnection of *SprintNet* determines the path length between any two servers within a *Cell* is 2, and 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 (limited by the port numbers on switches, a larger sized network usually needs more layers to scale up), 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 with smaller number of hops and lower queuing delay (buffering at each hop), and also lower transmission latency in practice [19].

3.2.2. Scalability and physical cost

For a 2-layer architecture, as shown in Table 1, *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, VL2 and FlatNet, whose wiring complexity is $O(n^3)$. Additionally, the number of switches used in *SprintNet* depends on the value of c and increases at the speed of $O(n)$ which also outperforms Fat Tree, VL2 and FlatNet.

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

3.2.3. 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 [8].

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

3.2.4. Network capacity

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

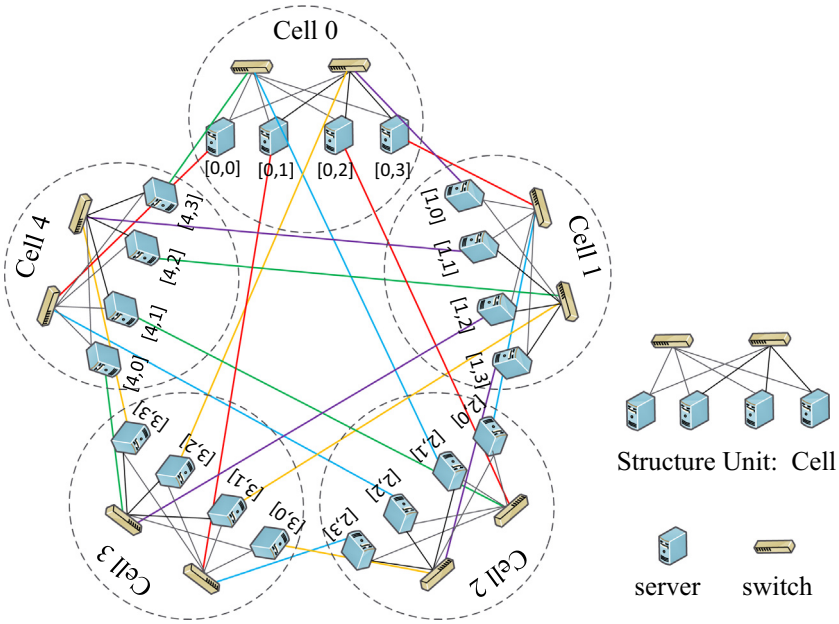


Fig. 1. A 20-server *SprintNet* using two 6-port switches in each cell.

Table 1
The comparison between different network architectures.

	Fat tree (3 layers)	VL2 (3 layers)	DCell (2 layers)	BCube (2 layers)	FlatNet (2 layers)	SprintNet (2 layers)
Number of servers	$\frac{n^3}{4}$	$\frac{(n-2)n^2}{4}$	$n(n+1)$	n^2	n^3	$(\frac{c}{c+1})^2 n^2 + \frac{c}{c+1} n$
Number of links	$\frac{3n^3}{4}$	$\frac{(n+2)n^2}{4}$	$\frac{3n(n+1)}{2}$	$2n^2$	$2n^3$	$\frac{c^2 n^2}{c+1} + cn$
Per server	3	$\frac{n+2}{n-2}$	$\frac{3}{2}$	2	2	$\geqslant 2$
Number of switches	$\frac{5n^2}{4}$	$\frac{3n}{2} + \frac{n^2}{4}$	$n+1$	$2n$	$2n^2$	$\frac{c^2}{c+1} n + c$
Per server	$\frac{5}{n}$	$\frac{n+6}{n^2-2n}$	$\frac{1}{n}$	$\frac{2}{n}$	$\frac{2}{n}$	$\frac{c+1}{n}$
Bisection bandwidth	$\frac{n^3}{8}$	$\frac{n^2}{4}$	$\frac{n^2}{4} + \frac{n}{2}$	$\frac{n^2}{2}$	$\frac{n^3}{4}$	$\frac{c^2 n^2}{2(c+1)^2} + cn$
Per server	$\frac{1}{2}$	$\frac{1}{n-2}$	$\approx \frac{1}{4}$	$\frac{1}{2}$	$\frac{1}{4}$	$\frac{1}{2} + \frac{(2c+1)(c+1)}{2(cn+c+1)}$
Network diameter	6	6	5	4	8	4
Number of node-disjointed paths	0	0	2	2	2	$c+1 (\geqslant 2)$

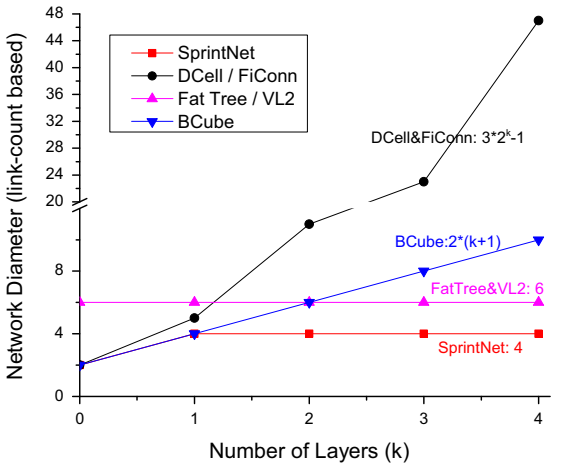


Fig. 2. The network diameters of various architectures with different layers.

the smallest throughput are named as bottleneck flows. ABT indicates the sum of the throughputs of all the bottleneck flows.

ABT performs 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 *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 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 Eq. (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. □

This further theoretically proves that *SprintNet*'s low network diameter and low average path length enable *SprintNet* in achieving a high network capacity.

3.2.5. Fault tolerance

Benefitting from the specially designed interconnection with rich physical connections, *SprintNet* achieves good fault tolerance. As shown in Table 1, compared with an architecture with two layers, *SprintNet* has $c+1$ ($c \geqslant 1$) 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 4 further strengthens the reliability of *SprintNet*.

3.2.6. 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.

3.3. 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 trade-off 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).

$$Objective: Minimize\ f(c,n) = \frac{Cost(switches)}{FT} \tag{6}$$

Subject to:

$$Cost(switches) = N_{switch} * Price_{n-port-switch} \tag{7}$$

$$= \left(\frac{c^2}{c+1} * n + c\right) * Price_{n-port-switch} \tag{8}$$

$$FT = c + 1 \tag{9}$$

$$\left(\frac{c}{c+1}\right)^2 * n^2 + \frac{c}{c+1} * n = N_{server} \tag{10}$$

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, $Price_{n-port-switch}$ 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. (6) is provided as follows:

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

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

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

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

Given a certain N_{server} sized network, the $Price_{n-port-switch}$ 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 further 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*.

4. Routing in SprintNet

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

4.1. Naïve routing scheme

The naïve routing scheme (NRS) is a shortest path routing algorithm. 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 or flooding 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. Primarily, each server only keeps the connectivity status of its neighbor servers. In order to compute the shortest routing path from server a to server b, starting from server a the broadcasting is recursively carried out in each step in the network. When the broadcasting packet arrives at server b, then it retraces back to server a along the reverse path reaching b. In this way, multiple paths between server a and b may be obtained, then the route with the shortest length will be chosen as the final routing path and stored in the routing table at server b. When there is more than one shortest path, the route will be chosen at random from the candidate routes, 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 $O(1)$.

In order to avoid ceaseless broadcasting packets in an endless loop and thus reduce the network workload, in the implementation of NRS each broadcasting packet is attached with a counter TTL to denote its maximum lifespan (the number of hops). Once the counter exceeds the predetermined threshold, the broadcasting packet is discarded. Since the network diameter of *SprintNet* is 4, which means the maximum shortest path length will not be larger than 4, hence we conservatively set the lifespan counter to 5 as default.

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.

Table 2
Routing path length distribution.

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

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

```
if src.fail()||dst.fail() then
  return null;
else
  if s1 == d1 then
    return Route0: src → switch x → dst;
  else
    if src == n && dst!= m && j.on() then
      return Route1: src → switch j → dst;
    else if src!= n && dst == m && i.on() then
      return Route2: src → switch i → 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 → switchi → m → switch y → dst;
        Route4: src → switchx → n → switch j → 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 → intermediate Cell_t → dst;
      end if
    end if
  end if
end if
```

4.2. 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.

The whole routing procedure can be generally divided into two cases as shown in the pseudocode of Algorithm 1. 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 *Cells*: $Cell_{s_1}$ 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: $src \rightarrow switchj \rightarrow dst$. When switch j fails, it takes Route3: $src \rightarrow switchi \rightarrow m \rightarrow switchy \rightarrow dst$ 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 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 2, 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.

5. Issues of the server-centric scheme

The computational servers in server-centric architectures are not only running regular applications but also acting as switches to relay the traffic in the system. Though this brings numerous advantages and convenience, it is also accompanied by several critical drawbacks:

(1) The network becomes not transparent to servers. Besides, servers cannot be independent of the network, and in some cases of routings, server's intervention is needed.

(2) The Operating System kernel or the network protocol of a server has to be modified to implement the

automatic fault-tolerant routing and flow-level load-balancing. Specifically, the OS must be able to detect current network status (e.g. connectivity, variation of delay, etc.) and discover feasible alternative routing paths in real-time. Moreover, certain routing protocol (e.g. source routing scheme) must be implemented in order to allow a single flow transfer through multiple paths simultaneously. However, modifying the OS kernel could be very complicated and time-consuming in practice (e.g. DCell involves more than 13,000 lines of C code [12]).

(3) The server may become the bottleneck of overall network performance and leads to additional packet delays, increased packet loss ratio and deceased aggregate bottleneck throughput, especially when suffering from insufficient software resources, e.g. CPU time and memory bandwidth [12], and servers cannot completely focus on computing and providing regular services.

(4) A certain number of NICs are needed to be installed on each server to meet the future scaling out, which may be not very practical since most of current data center servers only have two built-in ports.

(5) It is difficult to implement a Green Data Center since servers cannot be powered off even though they are idle for computing because they still undertake forwarding tasks.

All of the above issues are caused mainly due to putting the servers onto the network side and enabling them to forward packets. Based on this careful observation, in Section 6 we put forward a hard-ware based approach against these issues.

6. Approach to improve the performance of server-centric scheme

The forwarding tasks of servers in DCell and BCube are software implemented, where they use software for packet forwarding which relies too much on CPU, and the CPU becomes the major bottleneck hindering the system to achieve all potential capacity gains according to their experimental results. As noted in [11,12], the researchers also expect (not implemented) to use a specially-designed network card in the server in future so that server can execute the forwarding without the involvement of CPU. To some extent this design can reduce the forwarding delay, however, it cannot make the data center network completely transparent to servers. And in the cases of fault-tolerant routings, it still needs server's intervention.

In order to totally overcome the drawbacks addressed above, intuitively the most beneficial way is to shift the forwarding/routing tasks from a server to the network completely which makes servers independent from the network. In view of the above, we propose an efficient hardware-based approach by introducing a dedicated hardware named "Forwarding Unit". The basic idea under this approach is as illustrated in Fig. 3. The forwarding unit isolates the server from the network completely and operates as a middleware. In other words, this forwarding unit is totally responsible for the forwarding tasks and the server does not relay the traffic any more and thus no longer cares about the network status. Relieving the CPU from the forwarding work, the server only focuses on computing and its CPU resources can be saved to achieve a better performance of services. Furthermore, from the perspective of users, the entire data center network can be simply regarded as a giant switch as shown in Fig. 3 (right), and the architecture is then reduced down to a logical single-layer data center network in a high level view.

In order to better illustrate the design of forwarding unit, without loss of generality we take a two-layered *SprintNet* as the representative of server-centric architectures. Fig. 4 shows the internal structure of a forwarding unit which is specially designed for *SprintNet*. For a certain given node in *SprintNet*, there are at most $\frac{c}{c+1}n+1$ possibilities of routing path (because each layer contains only $\frac{c}{c+1}n+1$ subsystems), and each node has only up to $3*\frac{c}{c+1}n-1$ choices for one hop (because each node can reach $3*\frac{c}{c+1}n-1$ nodes in one hop via one switch), hence a route table with the size of only $(\frac{c}{c+1}n+1)log_2(3*\frac{c}{c+1}n-1)$ bits can describe its routing behaviors. Therefore, for a *SprintNet* with 9312 servers where $n=128, c=3$, the route table stored in the forwarding unit is just 99 bytes which implies a very small hardware cost. This provides the possibility that all the route tables can be pre-calculated and be stored in the forwarding units

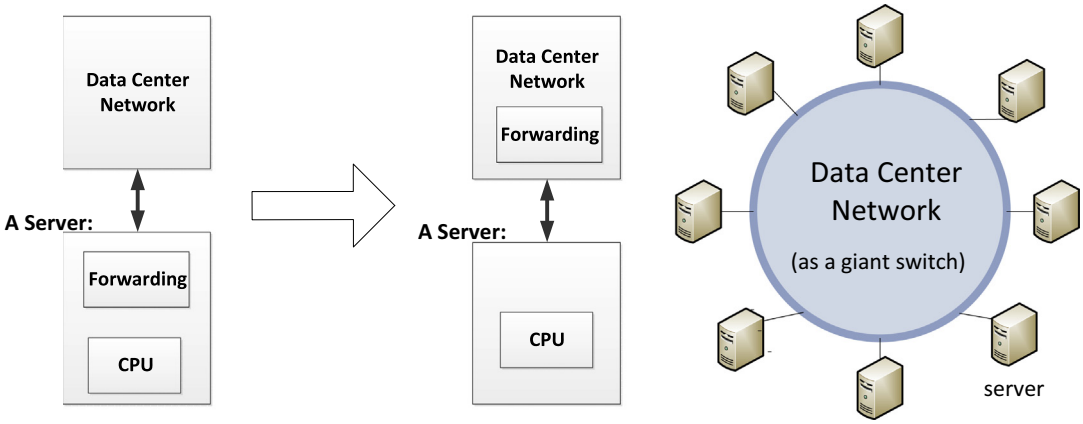


Fig. 3. Shifting forwarding task from a server to the network.

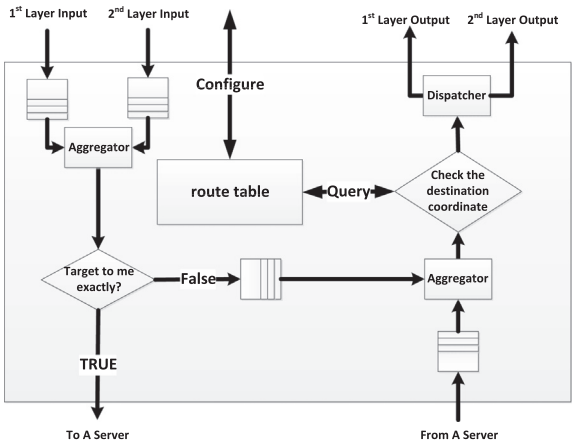


Fig. 4. The internal structure of a forwarding unit.

in advance, and then in the future be used directly at the complexity of $O(1)$, which can save much time cost in computing routing paths. In this way, the system behaves more effectively without wasting time in calculating the route tables before determining a path and avoids the high network overhead brought by broadcasting path-probing packets. Under the circumstances of fault-tolerance or network virtualization, we can offline calculate the routing strategies, and then make appropriate corresponding configurations to the forwarding units.

Moreover, this forwarding unit based approach which shifts the control of routing/forwarding from servers to the network is also in line with the principle of current trend of Software Defined Network. The system can adopt OpenFlow-like technology to achieve a global configuration and management for the routing behaviors. The controller can be an independent server or hosted on a compute server. In order to deal with the single point failure of the controller, multiple controllers can be used, where these controller have different roles: OFPCR_ROLE_EQUAL, OFPCR_ROLE_MASTER, and OFPCR_ROLE_SLAVE (as specified in [1]) to guarantee the robustness of the system. The forwarding unit should be Openflow enabled by installing flow tables and group tables so as to implement the pipeline processing. The controller-to-switch messages (such as feature messages, switch configuration messages, flow table configuration messages, modify-state messages, read-state messages, packet-out, barrier message, and role request messages), asynchronous messages (OFPT_PACKET_IN, OFPT_FLOW_REMOVED, OFPT_PORT_STATUS, and OFPT_ERROR_MSG) and symmetric messages (Hello messages, Echo messages and Experimenter messages) can be transmitted through Openflow secure channel either in band or out of band. With the help of Openflow technology, the system can gain a better flow scheduling in data center networks, and can better realize the network virtualization as well. Besides, against the shortcomings that the routing paths in server-centric architectures are comparatively long and multiple additional forwarding actions are needed, the OpenFlow-like technology is much easier to manage them globally which can further improve the system's flexibility.

Additionally, following this scheme the entire data center network can be designed as a whole, and its internal routing is simplified, for example, there is no need to consider the case of server failures. The complex cabling within the data center network can be directly implemented on the PCB or with the help of a dedicated physical port which makes the deployment simplified. And the system becomes more reliable.

The common problem of adopting openflow technology is that the extra latency (such as round-trip time and route computing time) incurred by interaction between the forwarding unit and the controller even though such latency is acceptable for elephant flows.

Concluding from the above analysis and discussions, on the one hand the forwarding unit is a hardware-based implementation approach which logically flattens the data center network and makes the internal network work as a logical giant switch. On the other hand, it provides an efficient way to improve the performance of server-centric architectures. Additionally, it follows the core idea of software defined network, but without changing the switch, and it also gives full consideration on the hardware's cost problem of forwarding unit (more in subSection 7.3).

7. Evaluation

In this section, extensive simulations are conducted to evaluate the performance of *SprintNet* and forwarding unit approach under various network conditions using the TAR routing scheme.

7.1. Simulation overview

All simulations are conducted using our DCNSim [18] simulator, which can simulate several data center network topologies (such as DCell, FatTree, BCube, FlatNet, and HyperBCube) 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. 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. Furthermore, according to the findings in [4] about the characteristics of the packet-level communications, the packet inter-arrival time reveals an ON/OFF pattern and its distribution follows the Lognormal mode for OFF phase, while the distribution varies between Lognormal mode, Weibull mode and Exponential mode in different data centers during the application-sensitive ON phase. In our simulations, the Lognormal distribution mode (directly using the Simjava package of eduni.simjava.distributions [13]) is applied to determine the inter-arrival time of packets. All the links are capable of bidirectional communications with 1 GBps unidirectional link bandwidth. The packet size is set to be 1500 bytes which equals to the default MTU of a link. Additionally, the default TTL of a packet in TAR routing is set as 128.

7.2. Evaluation of SprintNet

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

7.2.1. 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. Table 3 demonstrates the simulation results about APL for eight different sized 2-layer *SprintNet*s and DCells. And Table 4 illustrates the detailed statistics of flows and routes for the case of 1056-server *SprintNet*. It can be seen from Table 3 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.

7.2.2. Aggregate bottleneck throughput

As illustrated in Eq. (1) in Section 3.2, the aggregate bottleneck throughput *ABT* is inversely proportion to the average path length *APL* (i.e. $NCP_{ABT} = \frac{1}{APL}$). Table 5 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, as shown in Table 5, the N_{links} of a 2353-server *SprintNet* is 9408, hence its NC_{links} is $9408*2 = 18{,}816$ (two-way communication). Given the APL of a 2352-server *SprintNet* is 3.88, its *ABT* reaches 4762, therefore its $NCP_{ABT} = \frac{4762}{18{,}816} = 25.31\%$, which is already very close to its theoretical limit ($\frac{1}{APL} = \frac{1}{3.88} = 25.77\%$). This reveals the good performance of the *SprintNet* in the aggregate bottleneck throughput.

7.2.3. Speedup of the first layer

Like other server-centric proposals, in *SprintNet*, the network traffic burden carried on different layers differs. According to the simulation results as shown in Table 4, the level-0 links undertake higher network loads which results in a higher link utilization rate. This careful observation reveals that the first layer is more likely to occur network congestion and become the system's bottleneck. Fig. 5 presents the simulation results for three different sized *SprintNet* with different speedup factors using TAR and NRS routing schemes, which exhibits that the aggregate bottleneck throughput can be further improved with certain speedup of first layer. Another intuitive finding, which can be derived from Fig. 5, is that NRS scheme obtains slightly higher ABT than TAR scheme mainly due to the induced higher extra network load by NRS. According to the evaluation results, the most beneficial speedup factor to the first layer network is suggested as 1.3∼1.5, which leads to a more cost-effective data center network.

Table 4
The statistics for the case of 1056-server *SprintNet*.

All-to-all traffic pattern & Uniform distribution flow mode	
Flow statistics (4,257,792 flows in total)	3,176,448 flows using 2112 level-0 links 1,081,344 flows using 1056 level-1 links
Route statistics (1,114,080 routes in total)	99,264 paths of length 2 1,014,816 paths of length 4
Average path length	3.82

Table 5
The ABT of 2352-server *SprintNet* and DCell.

	ABT	NC_{links}	NCP_{ABT} (%)	$\frac{1}{APL}$ (%)
All-to-all traffic pattern (2352-server network)				
SprintNet	4762	18,816	25.31	$\frac{1}{3.88} = 25.77$
DCell	1213	7056	17.19	$\frac{1}{4.86} = 20.58$

7.2.4. Network reliability

Figs. 6 and 7 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. 6, 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.

Table 3
The average path length statistics.

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.

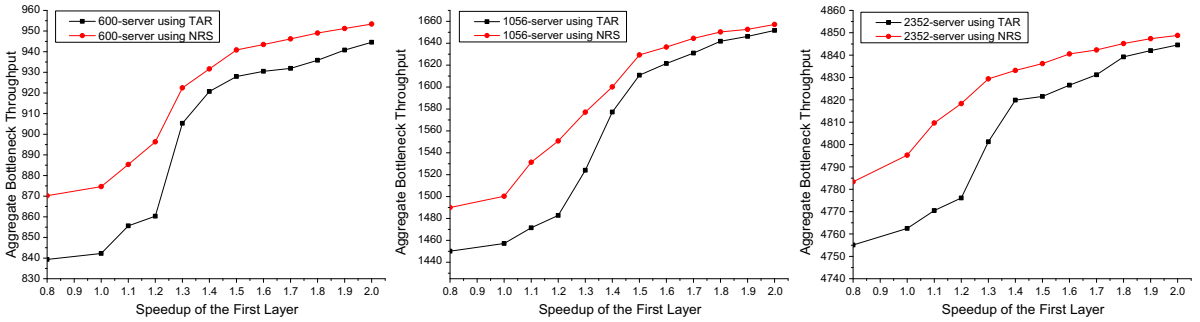


Fig. 5. The ABT performance of different sized *SprintNet* with various lower layer speedup factors.

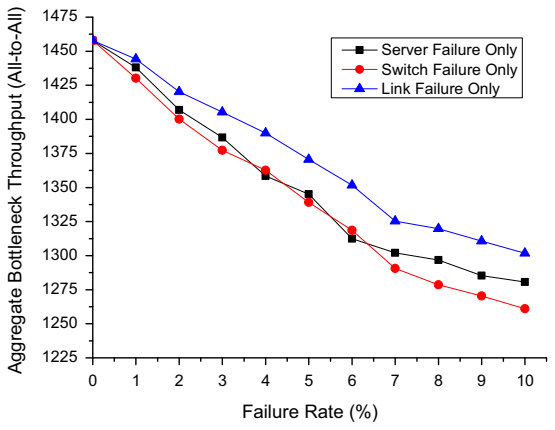


Fig. 6. The ABT performance of a 1056-server *SprintNet* under faulty conditions with TAR routing scheme.

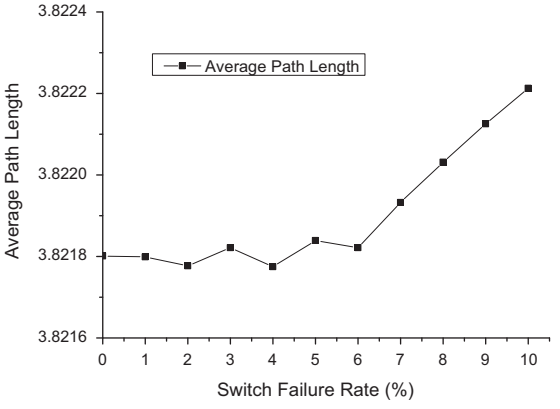


Fig. 7. The APL performance of a 1056-server *SprintNet* under faulty conditions with the TAR routing scheme.

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 simulation results illustrated in Fig. 7 reveal that the APL varies slightly and remains almost the same under different switch failure rates. For example, the APL under a fault-free condition is 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.

7.3. Evaluation of forwarding unit

In this subsection, the performance of forwarding unit based approach is evaluated from various aspects under different network conditions using TAR routing scheme.

7.3.1. System latency

Precisely the network latency consists of the queuing delay at each hop, transmission delay and propagation delay. In order to actually evaluate the overall packet delay of the network, we use the global packet lifetime (the time from packet's generation to the arrival at its destination) to measure the system latency. Fig. 8 illustrates the performance of average global packet lifetime (in millisecond) in different sized *SprintNet* networks. In this experiment, the buffer size of forwarding unit is set to 10 MTU (1 MTU = 1500 bytes) by default. It is clearly can be seen that 13%–20% reduction of the network latency is feasible when applying forwarding units, and larger sized network achieves a larger latency reduction. This reveals that the forwarding unit based approach can contribute much to the network latency.

7.3.2. Aggregate bottleneck throughput

As described and proved in Section 3.2, ABT is used to evaluate the maximum sustainable throughput over the entire network under the all-to-all traffic pattern.

The evaluation of ABT is conducted on a 1056-server *SprintNet* using two 48-port switches in each *Cell*, which has 3168 physical links. Table 6 gives the simulation results of ABT and APL. Given the APL = 3.82, the *SprintNet* applying forwarding unit achieves ABT of 1560.21, which reaches as high as 24.625% of the overall network capacity and is actually very close to the theoretical limitation (1/3.82 = 26.178%). Compared with the original *SprintNet*, forwarding unit improves the throughput by $\frac{1560.21-1457.11}{1457.11} = 7.08\%$. Clearly, the *SprintNet* applying forwarding unit

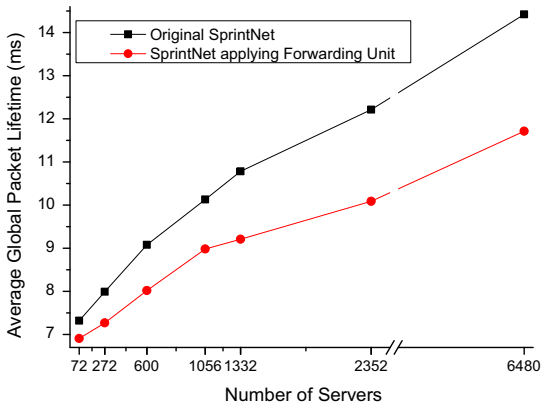


Fig. 8. The comparison of the system performance in network latency.

Table 6
The performance of ABT in a 1056-server *SprintNet*.

	Original *SprintNet*	*SprintNet* applying forwarding unit
1056-server SprintNet, APL = 3.82		
ABT	1457.11	1560.21
P_{ABT} (%)	$\frac{1457.11}{2*3168} = 22.997$	$\frac{1560.21}{2*3168} = 24.625$
$\frac{1}{APL}$ (%)	26.178	26.178

achieves a much better performance in ABT than the original *SprintNet*.

7.3.3. Hardware cost and average packet loss ratio

Table 7 presents the cost of hardware implementation and system's performance in the average packet loss ratio. As illustrated in the table, the hardware cost is very low, for instance, for a 9312-server sized data center the size of route table stored in each forwarding unit is less than 0.1 KB. Besides, the system demonstrates a favorable performance in controlling traffic congestion and packet loss. The average packet loss ratio keeps at a very low level which is no more than 0.02% (under the Lognormal flow distribution mode), and there is a certain decrease when the buffer size increases.

7.3.4. Fault tolerance

The fault tolerance mainly depends on the network architecture (like redundant physical connections) and the fault tolerant routing schemes. The network devices (e.g. switch, router) and links occurring failures greatly degrades the network performance (such as ABT, APL, routing connection success ratio). In the server-centric architectures, the server failures also greatly affect the network performance since servers are also considered as the routers in the system. However, the forwarding unit based approach which isolates the servers from the network completely has no need to consider the case of server failures any more when designing the fault tolerant routing scheme.

Figs. 9 and 10 exhibit the performance of ABT and connection failure ratios in case of link failures and switch

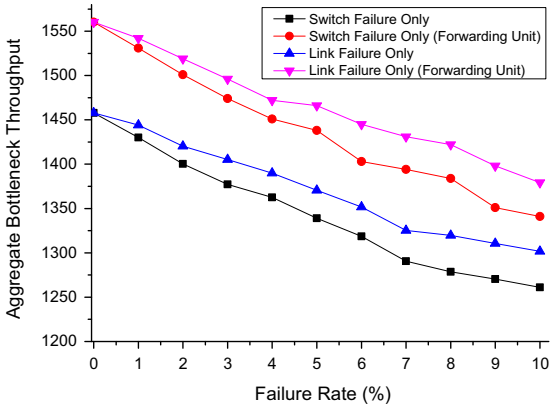


Fig. 9. The performance of ABT under faulty conditions in a 1056-server SprintNet running with forwarding units.

Table 7
Hardware cost and system performance.

n & c	Forwarding units	Route table size (bytes)	Buffer size (MTU)	Avg. packet loss ratio (%)
$n = 12$, $c = 3$	90	5.88	10	0.0176
			15	0.0162
			20	0.0125
$n = 24$, $c = 3$	342	13.60	10	0.0180
			15	0.0169
			20	0.0138
$n = 48$, $c = 3$	1332	31.18	10	0.0189
			15	0.0176
			20	0.0143
$n = 64$, $c = 3$	2352	43.85	10	0.0192
			15	0.0181
			20	0.0149
$n = 128$, $c = 3$	9312	99.00	10	0.0198
			15	0.0186
			20	0.0153

$*n$ denotes the number of ports on switches.
c indicates the number of switches per Cell.

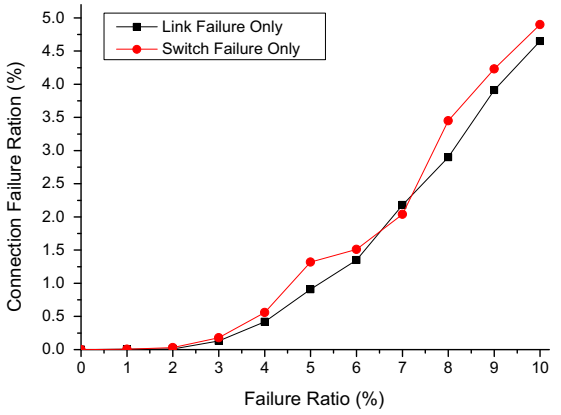


Fig. 10. The performance of fault tolerance under faulty conditions in a 1056-server SprintNet running with forwarding units.

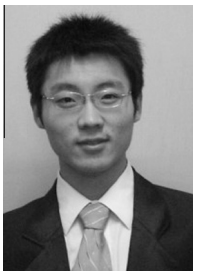
failures in a 1056-server *SprintNet* applying forwarding units. Although the system yields a gradually decayed network performance as the failure rate increases, the measured results still largely outweigh the original *SprintNet*. For example, the ABT in case of 10% switch failure rate is achieved as 1341.19, while the original *SprintNet* only obtains 1260.98 ABT for the same case, which indicates a much better improvement in ABT after introducing forwarding units. Moreover, the results shown in Fig. 10 reveal that the system holds an upper bound of connection failure ratio, i.e. if the link/switch failure rate is f then the connection failure ratio of whole system will not exceed $f/2$. All of these results reflect the good performance in fault tolerance.

8. Conclusion

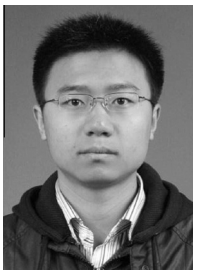
In this paper we firstly proposed *SprintNet*, a novel server-centric architecture for data centers, and presented its design, analysis and evaluations. *SprintNet* is highlighted by its low diameter, high bisection bandwidth, good fault tolerance, and high aggregate bottleneck throughput. The specially 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. Furthermore, based on careful investigations we addressed some critical shortcomings existed in the server-centric data center architectures caused by depending on servers to forward packets. In order to efficiently solve these drawbacks, we proposed an elegant approach which shifts forwarding tasks from servers to the network aiming to achieve a logically flat data center network. The forwarding unit totally isolates the servers from the network, and makes the data center network completely transparent to servers which brings abundant of advantages. In order to evaluate the feasibility and performance of this approach, extensive simulations are conducted, and the evaluation results convince its efficiency and feasibility.

References

[1] Openflow switch specification v1.3.0, 2012.
[2] Hussam Abu-Libdeh, Paolo Costa, Antony Rowstron, Greg O'Shea, Austin Donnelly, Symbiotic routing in future data centers, ACM SIGCOMM Comput. Commun. Rev. 40 (4) (2010) 51–62.
[3] Mohammad Al-Fares, Alexander Loukissas, Amin Vahdat, A scalable, commodity data center network architecture, ACM SIGCOMM Computer Communication Review, vol. 38, ACM, 2008, pp. 63–74.
[4] T. Benson, A. Akella, D.A. Maltz, Network traffic characteristics of data centers in the wild, in: Proceedings of the 10th Annual Conference on Internet Measurement, ACM, 2010, pp. 267–280.
[5] Kashif Bilal, Samee U. Khan, Limin Zhang, Hongxiang Li, Khizar Hayat, Sajjad A. Madani, Nasro Min-Allah, Lizhe Wang, Dan Chen, Majid Iqbal, et al., Quantitative comparisons of the state-of-the-art data center architecture, Concurrency and Computation: Practice and Experience, 2012.
[6] Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, Robert E. Gruber, Bigtable: a distributed storage system for structured data, ACM Trans. Comput. Syst. (TOCS) 26 (2) (2008) 4.
[7] Jeffrey Dean, Sanjay Ghemawat, Mapreduce: simplified data processing on large clusters, Commun. ACM 51 (1) (2008) 107–113.
[8] Nathan Farrington, Erik Rubow, Amin Vahdat, Data center switch architecture in the age of merchant silicon, in: 17th IEEE Symposium on High Performance Interconnects, 2009 (HOTI 2009), IEEE, 2009, pp. 93–102.
[9] Sanjay Ghemawat, Howard Gobioff, Shun-Tak Leung, The Google file system, SIGOPS Oper. Syst. Rev. 37 (5) (2003) 29–43.
[10] Albert Greenberg, James R Hamilton, Navendu Jain, Srikanth Kandula, Changhoon Kim, Parantap Lahiri, David A Maltz, Parveen Patel, Sudipta Sengupta, Vl2: a scalable and flexible data center network, ACM SIGCOMM Computer Communication Review, vol. 39, ACM, 2009, pp. 51–62.
[11] Chuanxiong Guo, Guohan Lu, Dan Li, Haitao Wu, Xuan Zhang, Yunfeng Shi, Chen Tian, Yongguang Zhang, Songwu Lu, Bcube: a high performance, server-centric network architecture for modular data centers, ACM SIGCOMM Comput. Commun. Rev. 39 (4) (2009) 63–74.
[12] Chuanxiong Guo, Haitao Wu, Kun Tan, Lei Shi, Yongguang Zhang, Songwu Lu, Dcell: a scalable and fault-tolerant network structure for data centers, ACM SIGCOMM Computer Communication Review, vol. 38, ACM, 2008, pp. 75–86.
[13] F. Howell, R. McNab, SimJava: a discrete event simulation library for java, Simul. Ser. 30 (1998) 51–56.
[14] Michael Isard, Mihai Budiu, Yuan Yu, Andrew Birrell, Dennis Fetterly, Dryad: distributed data-parallel programs from sequential building blocks, ACM SIGOPS Oper. Syst. Rev. 41 (3) (2007) 59–72.
[15] Dan Li, Chuanxiong Guo, Haitao Wu, Kun Tan, Yongguang Zhang, Songwu Lu, Ficonn: using backup port for server interconnection in data centers, in: INFOCOM 2009, IEEE, IEEE, 2009, pp. 2276–2285.
[16] D. Lin, Y. Liu, Mounir. Hamdi, Jogesh Muppala, HyperBCube: A Scalable Data Center Network, IEEE ICC, 2012.
[17] Dong Lin, Yang Liu, Mounir Hamdi, Jogesh Muppala, Flatnet: towards a flatter data center network, in: Global Communications Conference (GLOBECOM), 2012 IEEE, IEEE, 2012, pp. 2499–2504.
[18] Y. Liu, J. Muppala, DCNSim: a data center network simulator, in: ICDCSW, IEEE, 2013.
[19] Yang Liu, Jogesh Muppla, Malathi Veeraraghavan, Dong Lin, Mounir Hamdi, Data center networks, 2013. <http://www.amazon.ca/Data-Center-Networks-Yang-Liu/dp/3319019481>.
[20] Radhika Niranjan Mysore, Andreas Pamboris, Nathan Farrington, Nelson Huang, Pardis Miri, Sivasankar Radhakrishnan, Vikram Subramanya, Amin Vahdat, Portland: a scalable fault-tolerant layer 2 data center network fabric, ACM SIGCOMM Computer Communication Review, vol. 39, ACM, 2009, pp. 39–50.
[21] L. Popa, S. Ratnasamy, G. Iannaccone, A. Krishnamurthy, I. Stoica, A cost comparison of datacenter network architectures, in: Proceedings of the 6th International Conference, ACM, 2010, p. 16.
[22] Ji-Yong Shin, Bernard Wong, Emin Gün Sirer, Small-world datacenters, in: Proceedings of the 2nd ACM Symposium on Cloud Computing, ACM, 2011, p. 2.
[23] Ting Wang, Zhiyang Su, Y. Xia, M. Hamdi, Rethinking the data center networking: architecture, network protocols, and resource sharing.
[24] T. Wang, Z. Su, Y. Xia, B. Quin, M. Hamdi, NovaCube: A Low Latency Torus-Based Network Architecture for Data Centers, in: Global Communications Conference (GLOBECOM), 2014 IEEE, IEEE, 2014.



Ting Wang received his Bachelor Sci. degree from University of Science and Technology Beijing, China, in 2008, and received his Master Eng. degree from Warsaw University of Technology, Poland, in 2011. From 02.2012 to 08.2012 he interned as a research assistant in the Institute of Computing Technology, Chinese Academy of Sciences. He is currently working towards the Ph.D. degree in Hong Kong University of Science and Technology. His research interests include data center networks, cloud computing, green computing, and software defined network.



Zhiyang Su received his B.E. degree in computer science and technology from China University of Geosciences (Beijing) in 2009, and M.S. degree in computer network and application from Peking University in 2012. Currently, he is pursuing Ph.D. degrees in Hong Kong University of Science and Technology. His research interests focus on software defined networking (SDN) and data center networking, especially on improving the performance of SDN.

Yu Xia is currently a postdoctoral fellow in Department of Computer Science and Engineering, the Hong Kong University of Science and Technology. He received the Ph.D. in computer science from Southwest Jiaotong University, China. He was a joint Ph.D student and a visiting scholar at Polytechnic Institute of New York University. His research interests include high-performance packet switches, data center networks and network architectures.

Jogesh Muppala received the Ph.D. degree in Electrical Engineering from Duke University, Durham, NC in 1991, the M.S. degree in Computer Engineering from The Center for Advanced Computer Studies, University of Southwestern Louisiana (now University of Louisiana at Lafayette), Lafayette, LA in 1987 and the B.E. degree in Electronics and Communication Engineering from Osmania University, Hyderabad, India in 1985. He is currently an associate professor in the Department of Computer Science and Engineering, The Hong Kong University of Science and Technology, Clear Water Bay, Kowloon, Hong Kong. He is also currently serving as the program director for the Master of Science in Information Technology (MSc(IT)) program. He also served as the Undergraduate Studies Director in the Computer Science Department from August 1999 to August 2001, and the Chair of Facilities Committee in the department from September 2005 to August 2007. He was previously a Member of the Technical Staff at Software Productivity Consortium (Herndon, Virginia, USA) from 1991 to 1992, where he was involved in the development of modeling techniques for systems and software. While at Duke University, he participated in the development of two modeling tools, the Stochastic Petri Net Package (SPNP) and the symbolic Hierarchical Automated Reliability and Performance Evaluator (SHARPE), both of which are being used in several universities and industry in the USA. He co-founded the International Workshop on Dependability of Clouds, Data Centers and Virtual Machine Technology (DCDV). He was also the program co-chair for the 1999 Pacific Rim International Symposium on Dependable Computing held in Hong Kong in December 1999. He also co-founded and organized The 1st Asia–Pacific Workshop on Embedded System Education and Research (APESER). He has also served on program committees of many international conferences.

Mounir Hamdi received the B.S. degree in Electrical Engineering – Computer Engineering minor (with distinction) from the University of Louisiana in 1985, and the MS and the PhD degrees in Electrical Engineering from the University of Pitts-burgh in 1987 and 1991, respectively. He is a Chair Professor at the Hong Kong University of Science and Technology, and was the head of department of computer science and engineering. Now he is the Dean of the College of Science, Engineering and Technology at the Hamad Bin Khalifa University, Qatar. He is an IEEE Fellow for contributions to design and analysis of high-speed packet switching. He is/was on the Editorial Board of various prestigious journals and magazines including IEEE Transactions on Communications, IEEE Communication Magazine, Computer Networks, Wireless Communications and Mobile Computing, and Parallel Computing as well as a guest editor of IEEE Communications Magazine, guest editor-in-chief of two special issues of IEEE Journal on Selected Areas of Communications, and a guest editor of Optical Networks Magazine. He has chaired more than 20 international conferences and workshops including The IEEE International High Performance Switching and Routing Conference, the IEEE GLOBECOM/ICC Optical networking workshop, the IEEE ICC High-speed Access Workshop, and the IEEE IPPS HiNets Workshop, and has been on the program committees of more than 200 international conferences and workshops. He was the Chair of IEEE Communications Society Technical Committee on Transmissions, Access and Optical Systems, and Vice-Chair of the Optical Networking Technical Committee, as well as member of the ComSoc technical activities council. He received the best paper award at the IEEE International Conference on Communications in 2009 and the IEEE International Conference on Information and Networking in 1998. He also supervised the best PhD paper award among all universities in Hong Kong.