Space Shuffle: A Scalable, Flexible, and High-Performance Data Center Network

Ye Yu, *Student Member, IEEE* and Chen Qian, *Member, IEEE*

Abstract—The increasing need of cloud and big data applications requires data center networks to be scalable and bandwidth-rich. Current data center network architectures often use rigid topologies to increase network bandwidth. A major limitation is that they can hardly support incremental network growth. Recent work has been investigating new network architecture and protocols to achieve three important design goals of large data centers, namely, high throughput, routing and forwarding scalability, and flexibility for incremental growth. Unfortunately, existing data center network architectures focus on one or two of the above properties and pay little attention to the others. In this paper, we design a novel flexible data center network architecture, Space Shuffle (S2), which applies greedy routing on multiple ring spaces to achieve high-throughput, scalability, and flexibility. The proposed greedy routing protocol of S2 effectively exploits the path diversity of densely connected topologies and enables key-based routing. Extensive experimental studies show that S2 provides high bisectional bandwidth and throughput, near-optimal routing path lengths, extremely small forwarding state, fairness among concurrent data flows, and resiliency to network failures.

Index Terms—Data center networks, routing protocols, cloud computing

1 Introduction

Data center networks, being an important computing and communication component for cloud services and big data processing, require high inter-server communication bandwidth and scalability [1]. Network topology and the corresponding routing protocol are determinate factors of application performance in a data center network. Recent works have been investigating new topologies and routing protocols with a goal of improving network performance in the following aspects.

1) *High performance:* Many applications of current data center networks are data-intensive and require substantial intra-network communication, such as MapReduce [2], Hadoop [3], and Dryad [4]. Data center networks should have densely connected topologies which provide high bisection bandwidth and multiple parallel paths between any pair of servers. However, simply providing high-bandwidth is not enough. Routing protocols that can effectively exploit the network bandwidth and path diversity are essential. The routing protocol should achieve high throughput on the network topology. In addition, the routing protocol should result in short paths to achieve low latency.

2) *Flexibility:* A data center network may change after its deployment. According to a very recent survey [5], 93 percent US data center operators and 88 percent European data center operators will definitely or probably expand their data centers in 2013 or 2014. Therefore, a data center network should support *incremental growth* of network size, i.e., adding servers and network bandwidth incrementally to the data center network without destroying the current topology or replacing the current switches.

3) *Routing/forwarding scalability:* Routing and forwarding in a data center network should rely on small forwarding state of switches and be scalable to large networks. Forwarding table scalability is highly desired in large enterprise and data center networks, because switches use expensive and power-hungry memory to achieve increasingly fast line speed [6], [7], [8]. If forwarding state is small and does not increase with the network size, we can use relatively inexpensive switches to construct large data centers and do not need switch memory upgrade when the network grows.

Unfortunately, existing data center network architectures [7], [9], [10], [11], [12], [13], [14] focus on one or two of the above properties and pay little attention to the others. For example, the widely used multi-rooted tree topologies [9], [11] provide rich bandwidth and efficient routing, but their "firm" structures cannot deal with incremental growth of network size. The recently proposed Jellyfish network [14] uses random interconnect to support incremental growth and near-optimal bandwidth [15]. However, Jellyfish has to use inefficient k-shortest path routing whose forwarding state is big and cannot be aggregated. CamCube [13] and Small World Data Centers (SWDCs) [7] propose to use greedy routing for forwarding state scalability and efficient key-value services. Their greedy routing protocols do not produce shortest paths and can hardly be extended to perform multi-path routing that can fully utilize network bandwidth.

Designing a data center network that satisfies all three requirements seems to be challenging. The flexibility requirement introduces irregularity of network topologies. However, high-throughput routing protocols on irregular topologies, such as k-shortest path, are hard to scale. In this paper, we present a new data center network architecture, named Space Shuffle (S2), including *a scalable greedy routing*

• *The authors are with the Department of Computer Science, University of Kentucky, Lexington, KY 40506.*
E-mail: ye.yu@uky.edu, qian@cs.uky.edu.

Manuscript received 20 May 2015; revised 29 Nov. 2015; accepted 28 Jan. 2016. Date of publication 23 Feb. 2016; date of current version 12 Oct. 2016.
Recommended for acceptance by X. Gu.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TPDS.2016.2533618

TABLE 1
Desired Properties of Data Center Network Architectures

	FatTree [9]	CamCube [13]	SWDC [7]	Jellyfish [14]	S2
Network bandwidth	Benchmark	No Comparison	> Camcube	> FatTree and SWDC	≈ Jellyfish
Multi-path routing	✓	?	?	✓	✓
Incremental growth	×	?	?	✓	✓
Forwarding state per switch	$O(N^{\frac{1}{3}}))$	constant	constant	$O(kN \log N)$	constant
Key-based routing	×	✓	✓	×	✓
Switch heterogeneity	×	×	×	✓	✓

N: # switches, M: # links. Question mark means such property is not discussed in the paper.

protocol that achieves *high-throughput* and *near-optimal path lengths* on *flexible and bandwidth-rich networks* built by random interconnection.

S2 networks are constructed by interconnecting an arbitrary number of commodity ToR switches. Switches maintain coordinates in multiple *virtual spaces*. We also design a novel greedy routing protocol called *greediest routing*. Greedy routing guarantees to find multiple paths to any destination on an S2 topology. Unlike the existing greedy routing protocols [16], [17], which use only one single space, greediest routing makes decisions by considering the coordinates of the switches in multiple spaces. The routing path lengths are close to shortest path lengths. In addition, coordinates in multiple spaces enable efficient and high-throughput multi-path routing of S2. S2 also effectively supports key-based routing, which is able to fit many current data center applications using key-value stores [13].

Table 1 compares S2 and four other recent data center networks qualitatively in six desired properties, namely high bandwidth, multi-path routing, flexibility for incremental growth, small forwarding state, key-based routing, and support of switch heterogeneity. S2 achieves almost all desired properties while every other design has a few disadvantages.

We use extensive simulation results to demonstrate S2's performance in different dimensions, including routing path length, bisection bandwidth, throughput of single-path and multi-path routing, fairness among flows, forwarding table size, and resiliency to network failures. Compared to two recently proposed data center networks [7], [14], S2 provides significant advantages in some performance dimensions and is equally good in other dimensions.

Compared to the earlier version of this paper published in the Proceedings of ICNP 2014, this version includes the following differences and improvements: 1) We present the workflow of the S2 network to introduce the general structure of the proposed topology and protocols. 2) We introduce key-based routing, a unique and powerful function that the S2 network is able to provide compared to other data center network design. We also add the performance evaluation of key-based routing. 3) We present a failure recovery method, shortcut discovery. S2 is able to find an alternative path when the original path fails to delivery packets. We also show its experimental results. 4) We detail the experimental comparison of S2 and FatTree [9] by adding more simulation results in forwarding state, throughput, and routing path length. 5) We add more discussion of S2 including the wiring difficulties, an example of cabling plan, server multi-homing, and possible implementation approaches using Click [18] and Intel Data Plane Development Kit [19].

The rest of this paper is organized as follows. We present related work in Section 2. We describe the S2 topology and its construction in Section 3. In Section 4, we present the routing protocols and design considerations. We discuss the failure recovery mechanism in Section 5. We evaluate the performance of S2 in Section 6. We discuss a number of practical issues in Section 7 and finally conclude this work in Section 8.

2 Related Work

Recent studies have proposed a number of new network topologies to improve data center performance such as bisection bandwidth, flexibility, and failure resilience. Al-Fares et al. [9] propose a multi-rooted tree structure called FatTree that provides multiple equal paths between any pair of servers and can be built with commodity switches. VL2 [10] is a data center network that uses flat addresses and provides layer-2 semantics. Its topology is a Clos network which is also a multi-rooted tree [20]. Some data center network designs use direct server-to-server connection in regular topologies to achieve high bisection bandwidth, including DCell [12], BCube [21], CamCube [13], and Small-World data centers [7]. However, none of these designs have considered the requirement of incremental growth of data centers.

A number of solutions have been proposed to provide network flexibility and support incremental growth. Scafida [22] uses randomness to build an asymmetric data center network that can be scaled in smaller increments. In LEGUP [23], free ports are preserved for future expansion of Clos networks. REWRITE [24] is a framework that uses local search to find a network topology that maximizes bisection bandwidth whiling minimizing latency with a give cost budget. None of these three [22], [23], [24] have explicit routing design to utilize the network bandwidth of the irregular topologies. Jellyfish [14] is a recently proposed data center network architecture that applies random connections to allow arbitrary network size and incremental growth. Jellyfish can be built with any number of switches and servers and can incorporate additional devices by slightly changing the current network. Using k-shortest path routing, Jellyfish achieves higher network throughput compared to FatTree [9] and supports more servers than a FatTree using the same number of switches. However, to support k-shortest path routing on a random interconnect, forwarding state in Jellyfish switches is big and cannot be aggregated. Using the MPLS implementation of k-shortest path as suggested in [14], the expected number of forwarding entries per switch is proportional to $kN\log N$, where N is the number of switches in the network. In addition, k-shortest path algorithm is extremely time consuming. Its complexity is $O(kN(M + NlogN))$ for a single source (M is the number of links) [25]. This may result in slow convergence under network dynamics. Hence, Jellyfish may suffer from both *data plane* and *control plane scalability* problems. PAST [26] provides another multi-path solution for Jellyfish, but the throughput of Jellyfish may be degraded. A very recent study [15] discusses the near-optimal-throughput topology design for both homogeneous and heterogeneous networks. However, it does not provide routing protocols that achieves the throughput in practice.

As a scalable solution, greedy routing has been applied to enterprise and data center networks [7], [8], [13]. CamCube [13] employs greedy routing on a 3D torus topology. It provides an API for applications to implement their own routing protocols to satisfy specific requirements, called symbiotic routing. The network topologies of Small-World data centers are built with directly connected servers in three types: ring, 2D Torus, and 3D Hex Torus. ROME [8] is a network architecture to allow greedy routing on arbitrary network topologies and provide layer-2 semantics. For all three network architectures [7], [8], [13], multi-path routing is not explicitly provided.

SWDC, Jellyfish, and S2 all employ randomness to build physical topologies. However, they demonstrate substantially different performance because of their different logical organizations and routing protocols. SWDC applies scalable greedy routing on regularly assigned coordinates in a single space and supports key-based routing. Jellyfish provides higher throughput using k-shortest path routing, but it sacrifices forwarding table scalability. S2 gets the best of both worlds: it uses greedy routing on randomly assigned coordinates in multiple spaces to achieve both high-throughput routing and small forwarding state.

3 Space Shuffle Data Center Topology

3.1 Workflow of the system

The Space Shuffle topology is an interconnect of commodity top-of-rack (ToR) switches. In S2, all switches play an equal role and execute the same protocol. We assume there is no server multi-homing, i.e., a server only connects with one switch.

During the construction of S2, the switches and hosts are assigned with virtual coordinates as presented in Section 3.2. Using these coordinates, the operation personnel is able to compute a corresponding wiring plan and connect the switches and hosts accordingly following the steps in Section 3.3. S2 supports incremental deployment. When adding a new switch into the network, a set of virtual coordinates is assigned for the additional switch, and the topology is accordingly changed. The network operation personnel then disconnects some existing connections in the network and wires the new switch following the steps in Section 3.3.

When the topology construction is finished, S2 switches use greedy routing presented in Section 4 to forward data packets. The destination coordinates are encapsulated as the routable address in each packet. Upon receiving a packet, the switch finds out the next hop of the packet using the proposed greedy routing method. Such computation only involves basic arithmetic operations and is executed by the switch.

An S2 switch also includes a module that detects and deals with network failures, as described in Section 5. When it detects failure events, the failure recovery method, namely shortcut discovery, will be executed to route a packet to its destination.

3.2 Virtual Coordinates and Spaces

Each switch s is assigned a set of *virtual coordinates* represented by a L-dimensional vector $\langle x_1, x_2, \ldots, x_L \rangle$, where each element x_i is a randomly generated real number $0 \leq x_i < 1$[1]. There are L virtual ring spaces. In the ith space, a switch is *virtually* placed on a ring based on the value of its ith coordinate x_i. Coordinates in each space are circular, and 0 and 1 are superposed. Coordinates are distinct in a single space. In each space, a switch is physically connected with the two adjacent switches on its left and right sides. Two physically connected switches are called neighbors. For a network built with w-port switches,[2] it is required that $2L < w$. Each switch has at most $2L$ ports to connect other switches, called inter-switch ports. The rest ports can be used to connect servers. A neighbor of a switch s may happen to be adjacent to s in multiple spaces. In such a case, s needs less than $2L$ ports to connect adjacent switches in all L spaces. Switches with free inter-switch ports can then be connected randomly.

Fig. 1 shows an S2 network with 9 switches and 18 hosts in two spaces. As shown in Fig. 1a, each switch is connected with two hosts and four other switches. Fig. 1b shows coordinates of each switch in the two spaces. Figs. 1c and 1d are the two virtual spaces, where coordinate 0 is at top and coordinates increase clockwisely. As an example, switch B is connected to switches A, C, F, and G, because A and C are adjacent to B in space 1 and F and G are adjacent to B in space 2. A only uses three ports to connects adjacent switches I, B, and H, because it is adjacent to I in both two spaces. A and E are connected as they both have free inter-switch ports.

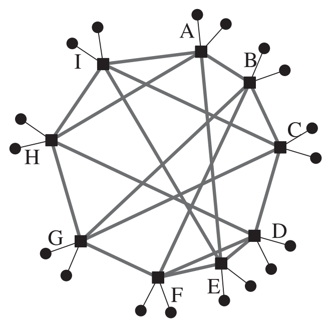
3.3 Topology Construction

As a flexible data center network, S2 can be constructed by either deploy-as-a-whole or incremental deployment.

For the deploy-as-a-whole construction of a network with N switches and H servers, each switch is assigned $\lfloor \frac{H}{N} \rfloor$ or $\lfloor \frac{H}{N} \rfloor + 1$ servers. The number of spaces L is then set to $\lfloor \frac{1}{2}(w - \lceil \frac{H}{N} \rceil) \rfloor$. Switch positions are randomly assigned in each space. For each space, cables are placed to connect every pair of adjacent switches. If there are still more than one switches with free ports, we randomly select switch pairs and connect each pair. We will discuss more cabling issues in Section 7.1.

1. x_is are real numbers. For implementation, we suggest using fixed-point number types with minimum representable value less than $\frac{1}{3N^2}$.

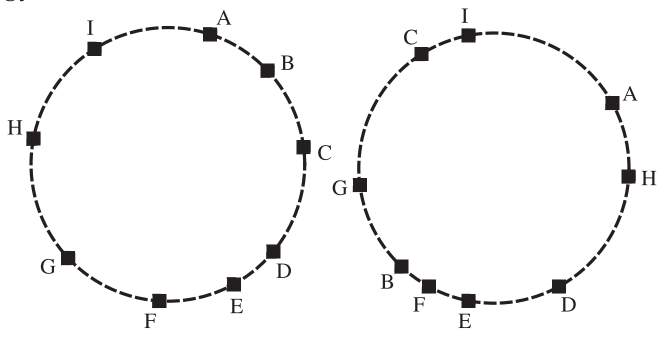
2. We now assume homogenous switches. We will discuss switch heterogeneity in Section 7.4.



(a) Space Shuffle topology

ID	x_1	x_2
A	0.05	0.17
B	0.13	0.62
C	0.23	0.91
D	0.36	0.42
E	0.42	0.53
F	0.51	0.58
G	0.63	0.73
H	0.78	0.26
I	0.91	0.97

(b) Switch coordinates



(c) Space 1 (d) Space 2

Fig. 1. Example S2 network with 9 switches and 18 servers in 2 spaces. Squares are switches and circles are servers.

S2 can easily support any expansion of the data center network using the incremental deployment algorithm. Suppose we decide to expand the data center network by m servers. A switch can connect $w - 2L$ servers, and we can determine the number of new switches is $\lceil m/(w - 2L) \rceil$. For each new switch s, we assign it a set of random coordinates. We find s's two adjacent nodes u and v in each space, which is currently connected. Then, the operator removes the cable between u and v and let s connect to both of them. New switches and servers can be added serially by iterative execution of this procedure.

Similar to Jellyfish [14], S2 can be constructed with any number of servers and switches. For incremental network expansion, only a few cables need to be removed and a few new cables are placed. Hence, there is very little network update cost.

At this point, coordinate generation is purely random. We will discuss the impact of coordinate randomness to the S2 routing protocol and introduce a method to generate desired random coordinates in Section 4.3.

Essentially, SWDC [7], Jellyfish [14], and S2 use similar random physical interconnects to approximate random regular graphs (RRGs). However, their logical organizations and routing protocols are substantially different, which result in different network performance such as throughput and forwarding table size.

4 Routing Protocols

A desired routing protocol in data center networks should have several important features that satisfy application requirements. First, a routing protocol should guarantee to find a loop-free path to delivery a packet from any source to any destination, i.e., *delivery guarantee* and *loop-freedom*. Second, the routing and forwarding should be scalable to a large size of servers and switches. Third, it should utilize the bandwidth and exploit path diversity of the network topology.

A straightforward way is to use shortest path based routing, such as OSPF on S2. However, shortest path routing has a few potential scalability problems. First, in the data plane, each switch needs to maintain a forwarding table whose size is proportional to the network size. The cost of storing the forwarding table in fast memory such as TCAM and SRAM can be high [7]. As the increasing line speeds require the use of faster, expensive, and power-consuming memory, there is a strong motivation to design routing protocol that only uses a small size of memory and does not require memory upgrades when the network size increases [6]. Second, running link-state protocols introduces non-trivial bandwidth cost to the control plane.

4.1 Greediest Routing

Since the coordinates of a switch can be considered geographical locations in L different spaces, we design a new greedy geographic routing protocol for S2, called *greediest routing*.

Routable address. The routable address of a server h, namely $\vec{X}$, is the virtual coordinates of the switch connected to h (also called h's access switch). Since most current applications uses IP addresses to identify destinations, an address resolution method is needed to obtain the S2 routable address of a packet, as ARP, a central directory, or a DHT [8], [27]. The address resolution function can be deployed on end switches for in-network traffic and on gateway switches for incoming traffic. In a packet, the destination server h is identified by a tuple $\langle \vec{X}, ID_h \rangle$, where $\vec{X}$ is h's routable address (virtual coordinates of the access switch) and ID_h is h's identifier such as its MAC or IP address. The packet is first delivered to the switch s that has the virtual coordinates $\vec{X}$, and then s forwards the packet to h based on ID_h.

MCD. We use the *circular distance* to define the distance between two coordinates in a same space. The circular distance for two coordinates x and y ($0 \leq x, y < 1$) is

$$CD(x, y) = \min\{|x - y|, 1 - |x - y|\}.$$

In addition, we introduce the *minimum circular distance* (MCD) for routing design. For two switches A and B with virtual coordinates $\vec{X} = \langle x_1, x_2, \ldots, x_L \rangle$ and $\vec{Y} = \langle y_1, y_2, \ldots, y_L \rangle$ respectively, the MCD of A and B, $MCD(\vec{X}, \vec{Y})$, is the minimum circular distance measured in the L spaces. Formally,

$$MCD(\vec{X}, \vec{Y}) = \min_{1 \leq i \leq L} CD(x_i, y_i).$$

Forwarding decision. The greediest routing protocol works as follows. When a switch s receives a packet whose destination is $\langle \vec{X}_t, ID \rangle$, it first checks whether $\vec{X}_t$ is its own coordinates. If so, s forwards the packet to the server whose identifier is ID. Otherwise, s selects a neighbor v such that v minimizes $MCD(\vec{X}_v, \vec{X}_t)$ to the destination, among all neighbors. The pseudocode of GREEDIST ROUTING ON SWITCH s

TABLE 2
MCDs to C from H and its Neighbors in Fig. 1

	Cir dist in Space 1	Cir dist in Space 2	Min cir dist
H	0.45	0.35	0.35
A	0.18	0.26	0.18
D	0.13	0.49	0.13
G	0.40	0.18	0.18
I	0.32	0.06	0.06

is presented in the appendix, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TPDS.2016.2533618.

For example, in the network shown in Fig. 1, switch H receives a packet whose destination host is connected to switch C, hence the destination coordinates are $\vec{X}_C$. H has four neighbors A, D, I, and G. After computing the MCD from each neighbor to the destination C as listed in Table 2, H concludes that I has the shortest minimal circular distance to C and then forwards the packet to I.

We name our protocol as "greediest routing" because it selects a neighbor that has a smallest MCD to the destination among all neighbors in all spaces. Existing greedy routing protocols only try to minimize distance to the destination in a single space (Euclidean, or in other kinds).

The algorithm of executing greediest routing on switch s is presented in Algorithm 1.

Algorithm 1. Greediest Routing on Switch s

```
input: Coordinates of all neighbors of switch $s$, destination
    addresses $\langle \vec{X}_t, ID \rangle$.
1 if $\vec{X}_s = \vec{X}_t$
2 Then $h \leftarrow$ the server connected to $s$ with identifier $ID$
3 Forward the packet to $h$
4 return
5 Compute $MCD_L(\vec{X}_v, \vec{X}_t)$ for all $s$'s neighbor switch $v$
6 Find $v_0$ such that $MCD_L(\vec{X}_{v0}, \vec{X}_t)$ is the smallest
7 Forward the packet to $v_0$
```

Greediest routing on S2 topologies provides delivery guarantee and loop-freedom. To prove it, we first introduce two lemmas.

Lemma 1. *In a space and given a coordinate x, if a switch s is not the switch that has the shortest circular distance to x in the space, then s must have an adjacent switch s' such that $CD(x, x_{s'}) < CD(x, x_s)$.*

Lemma 2. *Suppose switch s receives a packet whose destination switch is t and the coordinates are $\vec{X}_t$, $s \neq t$. Let v be the switch that has the smallest MCD to $\vec{X}_t$ among all neighbors of s. Then $MCD(\vec{X}_v, \vec{X}_t) < MCD(\vec{X}_s, \vec{X}_t)$.*

The proofs of the above two Lemma are presented in the Appendix, available in the online supplemental materialx. Lemma 2 states that if switch s is not the destination switch, it must find a neighbor v whose MCD is smaller than s's to the destination. Similar to other greedy routing protocols, when we have such "progressive and distance-reducing" property, we can establish the proof for delivery guarantee and loop-freedom.

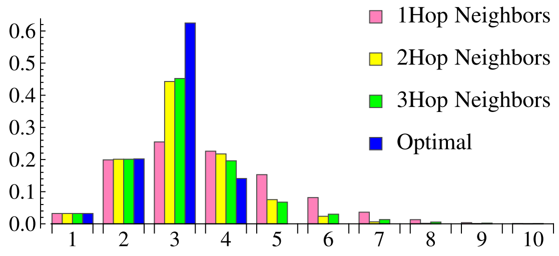


Fig. 2. Distribution of routing path lengths.

Proposition 3. *Greediest routing finds a loop-free path of a finite number of hops to a given destination on an S2 topology.*

Proof. (1) Suppose switch s receives a packet whose destination switch is t. If $s = t$, the destination host is one of the servers connected to s. The packet can be delivered.

(2) If $s \neq t$, according to Lemma 2, s will find a neighbor v such that $MCD_L(\vec{X}_v, \vec{X}_t) < MCD_L(\vec{X}_s, \vec{X}_t)$, and forward the packet to v.

(3) The MCD from the current switch to the destination coordinates strictly reduces at each hop. Greediest routing keeps making progress. Therefore, there is no routing loop. Since the number of switches is finite, the packet will be delivered to t. □

Like other greedy routing protocols [7], [8], greediest routing in S2 is highly scalable and easy to implement. Each switch only needs a small routing table that stores the coordinates of all neighbors. The forwarding decision can be made by a fixed and small number of numerical distance computation and comparisons. More importantly, the routing table size only depends on the number of ports and does not increase when the network grows. In the control plane, decisions are made locally without link-state broadcast in the network wide.

4.1.1 Reduce Routing Path Length

An obvious downside of greedy routing is that it does not guarantee shortest routing path. Non-optimal routing paths incur longer server-to-server latency. More importantly, flows routed by longer paths will be transmitted on more links, and thus consumes more network bandwidth [14]. To resolve this problem, we allow each switch in S2 stores the coordinates of 2-hop neighbors. To forward a packet, a switch first determines the switch v that has the shortest MCD to the destination, among all 1-hop and 2-hop neighbors. If v is an 1-hop neighbor, the packet is forwarded to v. Otherwise, the packet is forwarded to an one hop neighbor connected to v. Delivery guarantee and loop-freedom still holds. According to our empirical results, considering 2-hop neighbors can significantly reduce routing path lengths.

As an example, in a 250 10-port switch network, the distribution of switch-to-switch routing path lengths of k-hop neighbor storage is shown in Fig. 2, where the optimal values are the shortest path lengths. Storing 2-hop neighbors significantly reduces the routing path lengths compared with storing 1-hop neighbor. The average routing path length of greediest routing with only 1-hop neighbors is 5.749. Including 2-hop neighbors, the value is decreased to 5.199, which is very close to 4.874, the average shortest path length. However, including 3-hop neighbors does not improve the

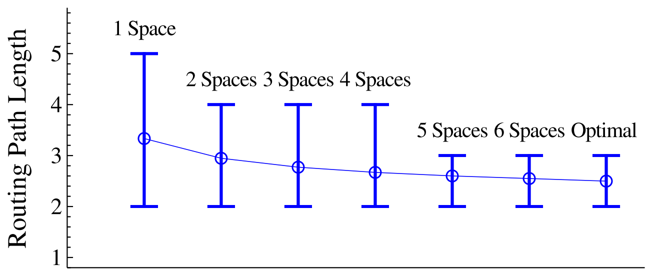


Fig. 3. Routing path length using different numbers of spaces.

routing path much compared with using 2-hop neighbors. Therefore, we decide to store 2-hop neighbors for S2 routing. Although storing 2-hop neighbors requires more state, the number of 2-hop neighbors are bounded by d^2, where d is the inter-switch port number, and this number is much lower than d^2 in practice. As forwarding state is independent of the network size, S2 routing is still highly scalable.

4.1.2 Impact of the Space Number

Proposition 3 holds for any $L \geq 1$. Therefore, greediest routing can use the coordinates only in the first d spaces, $d < L$, and apply the MCD in the first d spaces (d-MCD) as the greedy routing metric. In an extreme case where $d = 1$, greediest routing degenerates to greedy routing on one single ring using the circular distance as the metric. For $d < L$, the links connecting adjacent switches in the $d, d+1, \ldots,$ Lth spaces are still included in routing decision. They serve as random links that can reduce routing path length and improve bandwidth.

For all d, $1 \leq d \leq L$, greedy routing using d-MCD provides delivery guarantee and loop-freedom. We evaluate how the value of d affects routing performance by showing the number of spaces d versus the average routing path length of a typical network topology in Fig. 3. The two error bars represent the 10th and 90th percentile values. Only switch-to-switch paths are computed. The optimal results shown in the figure are shortest path lengths, which in average is 2.498. We find that the routing path lengths significantly reduce when the second and third spaces are included in greedy routing. Using more than 4 spaces, the average length is is close to the optimal value. Hence greediest routing in S2 always use as many spaces as switch port capacity allows. Commodity switches have more than enough ports to support 5 or more spaces.

4.2 Multi-Path Routing

Multi-path routing is essential for delivering full bandwidth among servers in a densely connected topology and performing traffic engineering. Previous greedy routing protocols can hardly apply existing multi-path algorithms such as equal-cost multi-path (ECMP) [28] and k-shortest paths [14], because each switch lacks of global knowledge of the network topology. Consider a potential multi-path method for greedy routing in a single Euclidean space. For different flows to a same destination, the source switch intentionally forwards them to different neighbors by making not-so-greedy decisions. This approach may result longer routing paths. In addition, these paths will share a large proportion of overlapped links because all flows are sent to a same direction in the Euclidean space. Overlapped links can easily be congested. Therefore, designing multi-path greedy routing in a single space is challenging.

Greediest routing on S2 supports multi-path routing well, due to path diversity across different spaces. According to Lemma 2, if a routing protocol reduces the MCD to the destination at every hop, it will eventually find a loop-free path to the destination. Based on this property, we design a multi-path routing protocol presented as follows. When a switch s receives the first packet of a new flow whose destination switch t is not s, it determines a set V of neighbors, such that for any $v \in V$, $MCD(\vec{X}_v, \vec{X}_t) < MCD(\vec{X}_s, \vec{X}_t)$. Then s selects one neighbor v_0 in V by hashing the 5-tuple of the packet, i.e., source address, destination address, source port, destination port, and protocol type. All packets of this flow will be forwarded to v_0, as they have a same hash value. Hence, packet reordering is avoided. This mechanism only applies to the first hop of a packet, and on the remaining path the packet is still forwarded by greediest routing. The main consideration of such design is to restrict path lengths. According to our observation from empirical results, multi-pathing at the first hop already provides good path diversity. The pseudocode of the multi-path routing protocol is presented in Algorithm 2.

Algorithm 2. Multi-path Routing on Switch s

```
   input: Coordinates of all neighbors, destination addresses
      ⟨$\vec{X}_t, ID$⟩
1  if $\vec{X}_s = \vec{X}_t$
2     then $h \leftarrow$ the server connected to $s$ with identifier $ID$
3        Forward the packet to $h$
4        return
5  if the packet is not from a server connected to $s$
6     then Perform greediest routing
7        return
8  $V \leftarrow \emptyset$
9  for each neighbor $v$ of $s$
10    do if $MCD_L(\vec{X}_v, \vec{X}_t) < MCD_L(\vec{X}_s, \vec{X}_t)$
11       then $V \leftarrow V \cup \{v\}$
12 Select $v_0$ from $V$ by hashing the source and destination
      addresses and ports
13 Forward the packet to $v_0$
```

S2 multi-path routing is also load-aware. As discussed in [29], load-aware routing provides better throughput. We assume a switch maintains a counter to estimate the traffic load on each outgoing link. At the first hop, the sender can select the links that have low traffic load. Such load-aware selection is flow-based: all packets of a flow will be sent to the same outgoing link as the first packet.

4.3 Balanced Random Coordinates

Purely uniform random generation of S2 coordinates will probably result in an imbalanced coordinate distribution. Fig. 6b shows an example coordinate distribution of 20 switches in a space. The right half of this ring has much more switches than the left half. Some switches are close to their neighbors while some are not. Theoretically, among n uniform-randomly generated coordinates, the expected value of the minimum distance between two adjacent coordinates is $\frac{1}{n^2}$, while the expected value of the maximum is

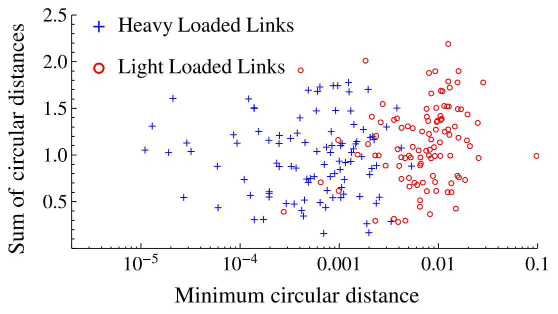


Fig. 4. Heavy and light loaded links. x-axis: MCD of a link's two endpoints; y-axis: sum of CDs of link's two endpoints in all spaces.

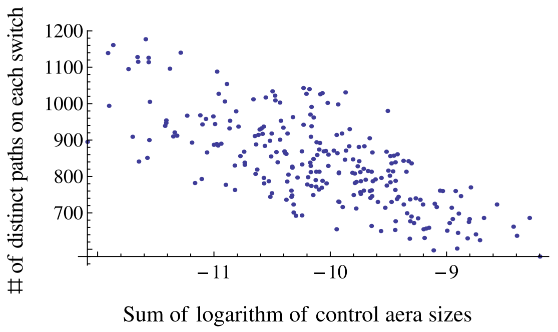


Fig. 5. Switch traffic load affected by control area sizes.

$\Theta(\frac{\log n}{n})$ [30]. Imbalance of coordinate distribution is harmful to S2 routing in two main aspects. First, greediest routing may intend to choose some links and cause congestion on them. We conjecture as follows. Consider two connected switches A and B whose coordinates are extremely close in one space. If one of them, say A, is the destination of a group of flows, other switches may intend to send the flows to B if they are unaware of A. These flows will then be sent from B to A and congest the link. Second, imbalanced key-value store occurs if switches are not evenly distributed on a ring. Previous work about load balancing in ring spaces cannot be applied here because they do not consider greediest routing.

We perform empirical study of the impact of coordinate distribution to routing loads. In a typical S2 network with 250 switches and $L = 4$, we run greediest routing for all pairs of switches to generate routing paths and then count the number of distinct paths on each link. We find the top 10 percent links and bottom 10 percent links according to the numbers of distinct paths and denote them by heavy loaded links and light loaded links respectively. We plot the heavy and light loaded links in a 2D domain as shown in Fig. 4, where the x-axis is the MCD of a link's two endpoints and the y-axis is the sum of circular distances of a link's two endpoints in all spaces. We find that the frequency of heavy/light loaded links strongly depends on the MCD of two endpoints, but has little relation to the sum of circular distances. If the MCD is shorter, a link is more likely to be heavy loaded. Hence, it is desired to avoid two switches that are placed very closely on a ring, meanwhile to enlarge the minimal circular distance for links.

We further study the impact of coordinate distribution to per-switch loads. We define the *control area* of switch s in a space as follows: Suppose switch s's coordinate in this space is x, s has two adjacent switches, whose coordinates are y and z respectively. The control area of s on the ring is the arc between the mid-point of $\widehat{y,x}$ and the mid-point of $\widehat{x,z}$. The *size of s's control area* in the space is defined as $\frac{1}{2}CD(x,y) + \frac{1}{2}CD(x,z)$. For the same network as Fig. 4, we count the number of different routing paths on each switch. We then plot this number versus the sum of logarithm of control area sizes of each switch in Fig. 5. It shows that they are negatively related with a correlation coefficient -0.7179. Since the sum of control area sizes of all switches is fixed , we should make the control areas as even as possible to maximize the sum-log values. This is also consistent to the load-balancing requirement of key-value storage. Based on the above observations, we present a Balanced Random Coordinate Generation algorithm below, which is also detailed in Algorithm 3.

Algorithm 3. Balanced Random Coordinate

```
   input: $n$ coordinates $x_1, x_2, \ldots, x_n$ in a circular space
   output: One new coordinate $x_{new}$
1  if $n = 0$ then return $RandomNumber(0,1)$
2  if $n = 1$
3    then $a \leftarrow x_1, b \leftarrow x_1 + 1$
4    else find $x_{r1}, x_{r2}$ among $x_1, x_2, \ldots, x_n$ such that $x_{r1} < x_{r2}$
       and $CD(x_{r1}, x_{r2})$ is the smallest
5  if $x_{r2} - x_{r1} < \frac{1}{2}$
6    then $a \leftarrow x_{r1}, b \leftarrow x_{r2}$
7    else $a \leftarrow x_{r2}, b \leftarrow x_{r1} + 1$
8  $t \leftarrow RandomNumber(a + \frac{1}{3n}, b - \frac{1}{3n})$
9  if $t > 1$ then $t \leftarrow t - 1$
10 return $t$
```

When a switch s joins the network with n switches, in every space we select two adjacent switches with the maximum circular distance, whose coordinates are y and z. By the pigeonhole principle, $CD(y,z) \geq \frac{1}{n}$. Then we place s in somewhere between y and z. To avoid being too close to either of y and z, we generate s's coordinate x in the space as a random number inside $(y + \frac{1}{3n}, z - \frac{1}{3n})$, so that $CD(x, y) \geq \frac{1}{3n}$ and $CD(x,z) \geq \frac{1}{3n}$. This algorithm can be used for either incremental or deploy-as-a-whole construction. It is guaranteed that the MCD between any pair of switches is no less than $\frac{1}{3n}$. An example of balanced random coordinates is shown in Fig. 6.

For 10-port 250-switch networks, we calculate the greediest routing path for every pair of switches. We show a typical distribution of routing load (measured by the number of distinct routing paths) on each link in Fig. 7, where we rank the links in increasing order of load. Compared with purely random coordinates, balanced random coordinates increase the load on under-utilized links (before rank 300) and

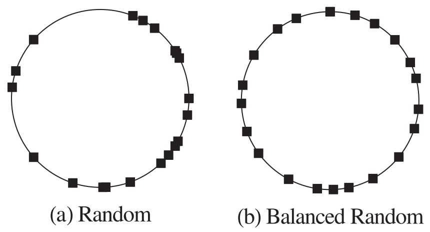


Fig. 6. Examples of random and balanced random coordinates.

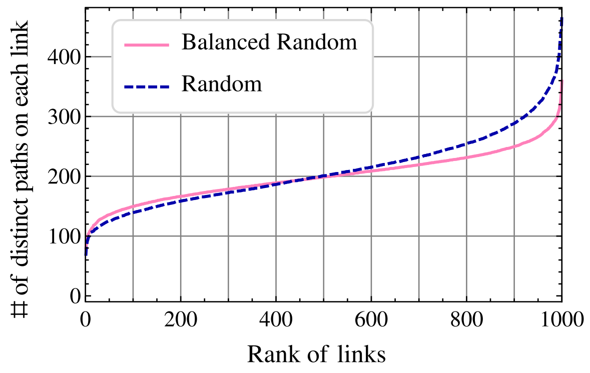


Fig. 7. Distribution of the number of paths on a link.

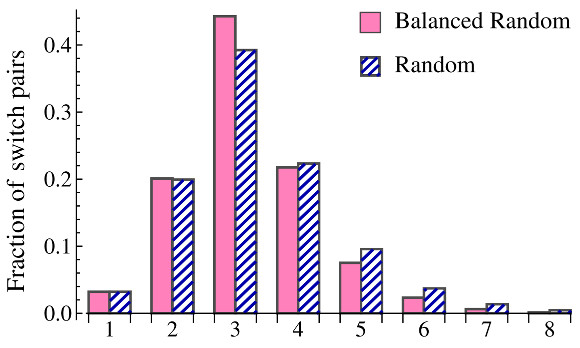


Fig. 8. Routing path length.

evidently decrease the load on over-utilized links (after rank 600). About 8 percent links of purely random coordinates have more than 300 paths on each of them, and only 1 percent links of balanced random coordinates have that number. The maximum number of distinct paths that a link is on also decreased from 470 to 350 using balanced random coordinates. Balanced random coordinates provide better fairness among links, and thus improve the network throughput.

We also examine the routing path lengths using balanced random coordinates. Fig. 8 shows the distribution of switch-to-switch routing path lengths of the same network discussed above. Balanced random coordinates slightly reduce the average routing path lengths from 3.35 to 3.20.

4.4 Key-Based Routing

Many applications running in today's data center networks use keys to identify data or users, such as MapReduce [2], Amazon Dynamo [31], Microsoft Dryad [4], and Facebook Photos [32]. Key-based routing enables direct data access without knowing the IP address of the server that stores the data. CamCube [13] is a prototype built by Microsoft to implement key-based routing in data centers. Moreover, key-based routing is an important building block of several network services such as host-discovery and multicast [8]. S2 supports efficient key-based routing based on the principle of consistent hashing. Only small changes are required to the greediest routing protocol.

The key-based routing of S2 utilizes two important properties.

(1) Given a destination coordinate y, greedy routing in a space using the circular distance guarantees to find *the switch closest to* y in the space.

(2) Given d-dimensional destination coordinates Y, greediest routing using d-MCD guarantees to find a switch s, such that *at least in one space* r $(1 \leq r \leq d)$, s *is the switch closest to* y_r.

We prove the correctness of the two properties in the appendix, available in the online supplemental material.

Each piece of data is stored in S2 based on a key. Let K_a be the key of a piece of data a. In S2, a should be stored in d multiple copies at different servers. In S2 key-based routing, a set of globally known hash functions $H_1, H_2, \ldots, H_d$ can be applied to K_a. We use $H(K_a)$ to represent a hash value for K_a mapped in $[0, 1]$. The routable address of K_a is defined as $\langle H_1(K_a), H_2(K_a), \ldots, H_d(K_a)\rangle$. Hereafter, we simply use $H(K_a)$ to represent the mapped hash value in the range of $[0, 1]$. For each space r, $1 \leq r \leq d$, the switch s whose coordinate $x_{s,r}$ is closest to $H_r(K_a)$ among all switches is called the *home switch* of K_a *in space* r. K_a has at most d home switches in total.[3] A replica of a is assigned to one of the servers connected to the home switch s. The server selection is determined by re-mapping $H(K_a)$ to the range $[1, n_h]$, where n_h is the number of servers connected to s. The selected server is called a *home server* which actually stores a. If s happens to be the home switch of K_a again in another space r', it selects the same server for a and use only one replica for a on that server.

S2 provides a function $\texttt{STORE}(K_a, r)$ that can send a copy of a to the home server in space r from any source. By running $\texttt{STORE}(K_a, r)$, a copy of a is forwarded by greedy routing using circular distance in space r. If a switch finds none of its neighbors has a smaller circular distance to $H(K_a)$, it can determine that the closest switch to $H(K_a)$ is itself and assign a to a server.

Obviously, any server can find the replica of data a in space r by sending a query message which will be forwarded using the same algorithm as $\texttt{STORE}(K_a, r)$. However, we design an algorithm $\texttt{Retrieve}(K_a)$ that results shorter paths to find a, by utilizing d-way replication. The basic idea is to use the second property, i.e., greediest routing using the d-MCD as the routing metrics guarantees to find at least one replica of the data.

In fact, if greediest routing in the first d spaces cannot make progress on switch s, then s is a home switch of K_a. S2 supports key-based routing by executing greediest routing to coordinate $\langle H_1(K_a), H_2(K_a), \ldots, H_d(K_a)\rangle$ in the first d spaces. Since the key to server mapping is not changed, $\texttt{Retrieve}(K_a)$ can find a replica of a.

Failure resiliency of key-value stores is an important issue but out of the scope of this paper. Therefore we leave it to future work.

5 Failure Recovery

Data center networks suffer from network failures. The switch and links may fail.*As a result, the correctness of the greediest routing may be violated, resulting in packet failure.*

Due to the dense interconnection of S2, a destination server is highly likely to remain reachable upon link failures

3. If there are two switches which are both closest to $H_r(K_a)$ with a same circular distance, the tie is broken by selecting the one with larger coordinate. For the ease of presentation, we consider there is only one switch closest to a coordinate hereafter.

or switch failures. Proposition 3 states that the greediest routing keeps making progress as long as the MCD from the current switch to the destination strictly reduces at each hop. Hence, S2 is able to deliver most of the packets under failures by maintaining a list of available neighbors on each switch and avoiding sending packets to failed switches or through failed links.

Failure detection. We categorize the failures in S2 into two classes :(1) *link failure*: a link connecting a pair of switches fails to transmit packets correctly; (2) *switch failure*: a switch fails to forward packets correctly. In S2, switches send `keepalive` message to their neighbors periodically.

We propose a shortcut path discovery mechanism for S2, so that the S2 network still guarantees to deliver all packets upon network failures, as long as there is an available path from the current switch to the destination.

Shortcut discovery. S2 handles these two kinds of failures by maintaining a list of available neighbors. A neighbor switch v of switch s is considered to be available, if and only if v and the link between v and s both exist. This list serves as the candidate set of greediest routing.

Upon network failures, the property of the topology is affected and is reflected in the available neighbor list. As a result, a packet with destination $\langle Y, ID\rangle$ may be routed to a *local minimum* switch p, such that for all p's neighbor s, $MCD(\vec{X}_p, Y) \leq MCD(\vec{X}_s, Y)$.

S2 uses a broadcast-confirm mechanism to discover an alternative shortcut for destination coordinates Y from switch p. Each S2 switch maintains a shortcut table to store the discovered shortcut paths. Each entry of the shortcut table includes the destination coordinates and next-hop switch, $\langle Y, s\rangle$. When switch p receives a packet with destination Y it should forward the packet to s.

When a switch p finds that it is the local minimum of a packet to destination coordinate Y, it initiates the process to discover a shortcut path. Switch p tags the packet as a probe packet and sends replicas of the probe packet along its all available outgoing links. These probe packets are then duplicated and transmitted along the network.

Once a probe packet reaches the home switch of the destination server, a shortcut path is established. Then the home switch sends a reply packet backwards to p to confirm the path. Upon receiving a reply packet, a switch adds an entry in its shortcut table. The rest packets to the same destination will match the shortcut table and be forwarded along the shortcut path.

The pseudocode of the failure-resilient routing algorithm of S2 is presented in the appendix, available in the online supplemental material. Note that, the packets can be tagged as REGULAR, PROBE and REPLY. The default type of the packets traverse through the network is REGULAR.

We evaluate the effectiveness of the failure-resilient routing mechanism in Section 6.9.

6 Performance Evaluation

In this section, we conduct extensive experiments to evaluate the efficiency, scalability, fairness, and reliability of S2 topologies and routing protocols. We compare S2 with two recently proposed data center networks, namely Small-World data center [7] and Jellyfish [14].

6.1 Methodology

Most existing studies use custom-built simulators to evaluate data center networks at large scale [7], [14], [15], [22], [23], [24], [33]. We find many of them use a certain level of abstraction for TCP, which may result in inaccurate throughput results. We develop our own simulator[4] to perform fine-grained packet-level event-based simulation. TCP New Reno is implemented in detail as the transport layer protocol. We simulate all packets in the network including ACKs, which are also routed by greedy routing. Our switch abstraction maintains finite shared buffers and forwarding tables.

We evaluate the following performance criteria of S2.

Bisection bandwidth describes the network capacity by measuring the bandwidth between two equal-sized part of a network. we calculate the empirical minimum bisection bandwidth of the network by randomly separating the servers into two partitions and computing the *maximum flow* between the two parts. The minimum bisection bandwidth value of a topology is computed from 50 random partitions. Each value shown in figures is the average of 20 different topologies.

Ideal throughput characterizes a network's raw capacity with perfect load balancing and routing (which do not exist in reality). A flow can be split into infinite subflows which are sent to links without congestion. Routing paths are not specified and flows can take any path between the source and destination. We model it as a *maximum multi-commodity network flow* problem and solve it using the IBM CPLEX optimizer [34]. The throughput results are calculated using a specific type of network traffic, called the *random permutation traffic* used by many other studies [14], [15], [33]. Random permutation traffic model generates very little local traffic and is considered easy to cause network congestion [33].

Practical throughput is the measured throughput of random permutation traffic routed by proposed routing protocols on the corresponding data center topology. It reflects how a routing protocol can utilize the topology bandwidth. We compare the throughput of S2 with Jellyfish and SWDC for both single-path and multi-path routing.

Scalability. We evaluate forwarding state on switches to characterize the data plane scalability. We measure the number of forwarding entries for shortest-path based routing. However, greedy routing uses distance comparison which does not rely on forwarding entries. Therefore we measure the number of coordinates stored. The entry-to-coordinate comparison actually gives a disadvantage to S2, because storing a coordinate requires much less memory than storing a forwarding entry.

Routing path lengths are important for data center networks, because they have strong impact to both network latency and throughput. For an S2 network, we calculate the routing path length for every pair of source and destination switches and show the average value.

Fairness. We evaluate throughput and completion time of different flows.

4. We experienced very slow speed when using NS for data center networks. We guess the existing studies do not use NS because of the same reason. Our simulator is available at https://github.com/sdyy1990/S2Simulator

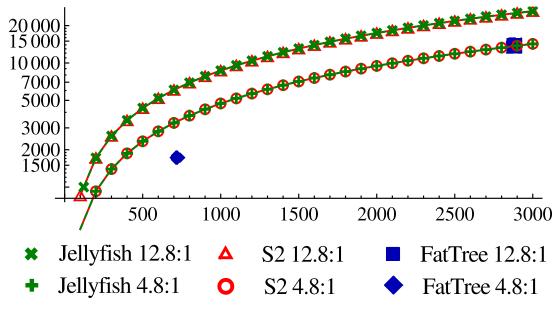


Fig. 9. Bisection bandwidth of S2, FatTree, and Jellyfish for two sever-to-switch ratios (12.8:1 and 4.8:1).

Resiliency to network failures reflects the reliability of the network topology and routing protocol. We evaluate the routing path length and routing success rate of greedy routing under switch failures. We further evaluate the practical throughput of our failure-resistant routing algorithm.

SWDC allows each node to store 2-hop neighbors. The default SWDC configuration has six inter-switch ports. For SWDC configurations with more than six inter-switch ports, we add random links until all ports are used. For Jellyfish, we use the same implementation of k-shortest path algorithm [35], [36] as in [14].

Each result shown by a figure in this section, unless otherwise mentioned, is from at least 20 production runs using different topologies.

6.2 Bisection Bandwidth

We compare the minimum bisection bandwidth of S2, Jellyfish, SWDC, and FatTree. For fair comparison, we use two FatTree networks as benchmarks, a 3,456-server 720-switch (24-port) FatTree and a 27,648-server 2,880 switch (48-port) FatTree. Note that, FatTree can only be built in fixed sizes with specific numbers of ports. The ratio of server number to switch number in above two configurations are 4.8:1 and 12.8:1 respectively. For experiments of S2 and Jellyfish, we fix the server-to-switch ratio in these two values and vary the number of switches. In Fig. 9, We show the bisection bandwidth of S2, FatTree, and Jellyfish, in the two server-to-switch ratios. The isolated diamond and square markers represent the minimum bisection bandwidth of FatTree. Both S2 and Jellyfish are free to support arbitrary number of servers and switches. They have identical bisection bandwidth according to our results. When using the same number of switches as FatTree (732 and 2,880), both S2 and Jellyfish provide substantially higher bisection bandwidth than FatTree. SWDC only uses a fixed 1:1 server-to-switch ratio and 6-port switches as presented in the SWDC paper [7]. In such configuration, S2, SWDC, and Jellyfish have similar bisection bandwidth. However, it is not clear whether SWDC can support incremental growth.

6.3 Ideal Throughput

We model the computation of ideal throughput as a maximum multi-commodity network flow problem: each flow is a commodity without hard demand. We need to find a flow assignment that maximizes network throughput while satisfying capacity constraints on all links and flow conservation. Each flow can be split into an infinite number of subflows and assigned to different paths. We solve

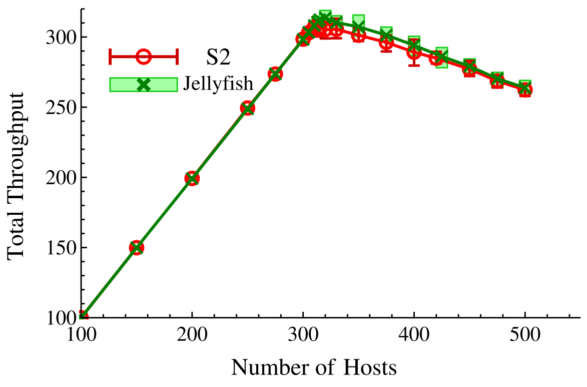


Fig. 10. Ideal throughput of S2 and Jellyfish for a 125-switch network.

it through linear programming using the IBM CPLEX optimizer [34] and then calculate the maximized network throughput. We show the throughput versus the number of servers of a typical 10-port 125-switch network in Fig. 10. When the server number is smaller than 320, the total throughput increases with the server number. After that the network throughput decreases because inter-switch ports are taken to support more servers. S2 is marginally worse than Jellyfish, which has been shown to have clearly higher throughput than FatTree with the same network equipments [14].

6.4 Scalability

We consider each coordinate as an entry and compare the number of entries in forwarding tables. In practice, a coordinate requires much less space than a forwarding entry. Even though we give such a disadvantage to S2, S2 still shows huge lead in data plane scalability. *Fig. 11 shows the average and maximum forwarding table sizes of S2 and Jellyfish in networks with 10 inter-switch ports. The number of entries of S2 is less than 500 and does not increase when the network grows.* The average and maximum forwarding entry numbers of Jellyfish in MPLS implementation [14] are much higher. Note that, the curve of Jellyfish looks like linear but is in fact $\Theta(N \log N)$. When N is in a relatively small range, the curve looks similar to linear. Using the SWDC configuration, the forwarding state of SWDC 3D is identical to that of S2, and those of SWDC 1D and 2D are smaller. The number of forwarding entries of the two-level routing table in FatTree is the number of ports, shown as a single point in Fig. 11.

From our experiments on a Dell Minitower with an Intel Core I7-4770 processor and 16 GB memory, we also find that it takes hours to compute all pair 8-shortest paths for Jellyfish with more than 500 switches. Hence it is difficult

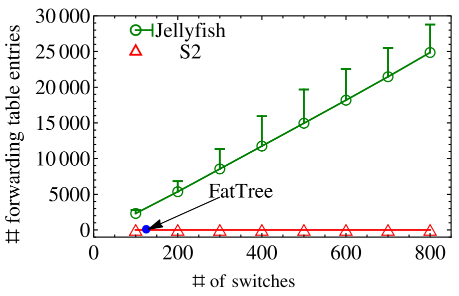


Fig. 11. Forwarding state of S2, Jellyfish, and FatTree.

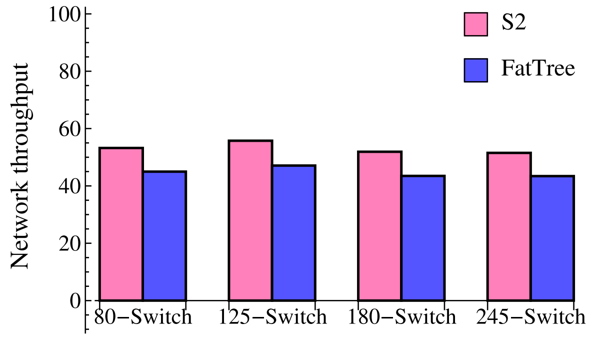


Fig. 12. S2 throughput on networks in FatTree configuration.

for switches to compute k-shortest paths of a large network in a way similar to link-state routing.

6.5 Practical Throughput

We conduct experiments to measure the practical throughput of S2, SWDC, FatTree and Jellyfish for both single-path and multi-path routing. For multi-path routing, the sender splits a flow into k subflows and sends them using S2 multi-path routing. Packets of the same subflow are forwarded via the same path. Since the multi-path routing protocol of SWDC is not clearly designed in [7], we use a multi-path method similar to that of S2.

In Fig. 12, we show the throughput of S2 and FatTree on networks in the FatTree's configuration of number of switches and switch-server ratio. We measure the throughput on a S2 network with the same number of switches and servers. The throughput is normalized to 100 with respect to the bisection bandwidth of the network. S2 shows slightly higher throughput than that of FatTree in all topologies.

In Fig. 14, we show the network throughput (normalized to 100) of S2, SWDC, and Jellyfish of a 12-port 250-switch network with 550 servers, using routing with 1, 2, 4, and 8 paths per flow. S2 and Jellyfish have similar network throughput. Using 2-path and 4-path routing, S2 has slightly higher throughput than Jellyfish, while Jellyfish has slightly higher throughput than S2 for 1-path and 8-path. Both S2 and Jellyfish overperform SWDC in throughput by about 50 percent. We find that multi-path routing improves the throughput of SWDC very little. We conjecture that multi-path greedy routing of SWDC may suffer from shared congestion on some links, since greedy routing paths to a same destination may easily contain shared links in a single space.

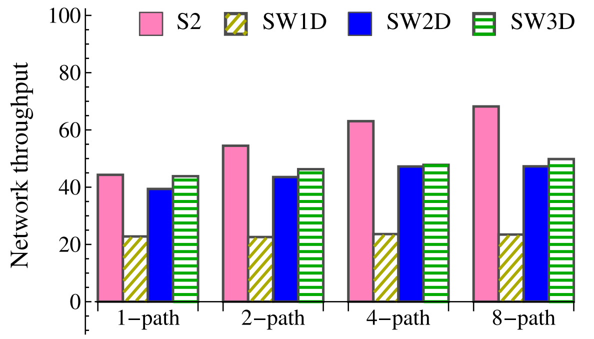


Fig. 13. Throughput of a 400-switch network in SWDC configuration.

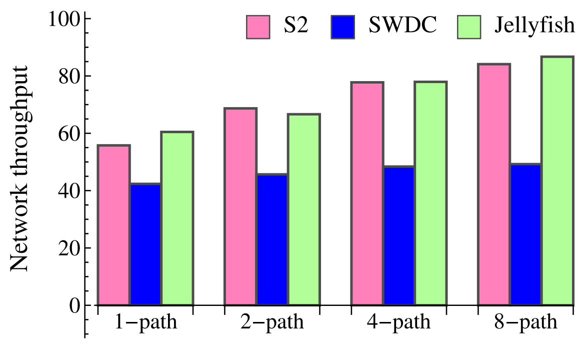


Fig. 14. Throughput of a 250-switch 500-server network.

In fact, SWDC has three variants (1D Ring, 2D Torus, and 3D Hex Torus) and special configuration (inter-switch port number is 6 and one server per switch). Hence, we conduct experiments to compare S2 with all three SWDC networks in the SWDC configuration. Fig. 13 shows that even under the S2 configuration, S2 provides higher throughput than all three types of SWDC especially when multi-pathing is used. We only show SWDC 2D in remaining results, as it is a middle course of all three types.

Flow completion time. We evaluate both all-flow and per-flow completion time of data transmission. Fig. 15 shows the time to complete transmitting all flows in the same set of experiments as in Fig. 14. Each flow transmits 16 MB data. S2 takes the least time (0.863 second) to finish all flows. SWDC 2D and 3D also finish all transmissions within 1 second, but use longer time than S2.

6.6 Fairness among Flows

We demonstrate that S2 provides fairness among flows in the following two aspects.

Throughput fairness. We evaluate the throughput fairness of S2. For the experiments conducted for Fig. 14, we show the distribution of per-flow throughput in Fig. 16 where the x-axis is the rank of a flow. It shows that S2 provides better fairness than SWDC and more than 75 percent of S2 flows can achieve the maximum throughput. Measured by the fairness index proposed by Jain et al. [37], S2 and SWDC 2D have fairness value 0.995741 and 0.989277 respectively, both are very high.

Completion time fairness. We take a representative production run and plot the cumulative distribution of per-flow completion time in Fig. 17. We found that S2 using 8-path routing provides both fast completion and fairness among flows – most flows finish in 0.2 - 0.4 second. S2 single-path completes flows more slowly, but is still competitive with

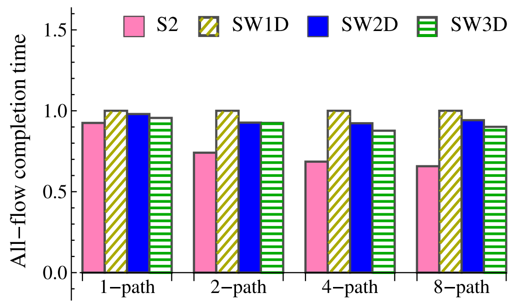


Fig. 15. All-flow completion time.

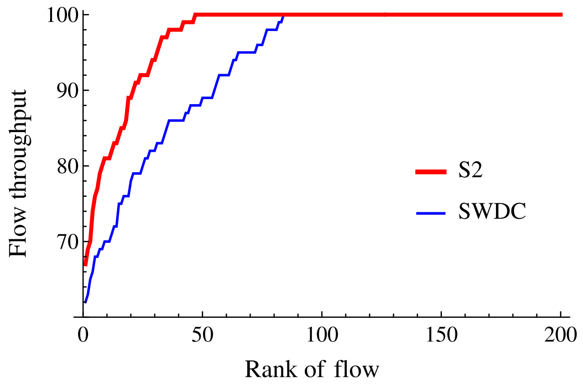


Fig. 16. Throughput fairness among flows.

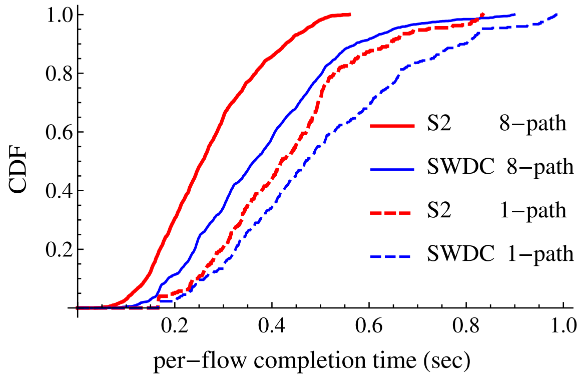


Fig. 17. Cumulative distribution of per-flow completion time.

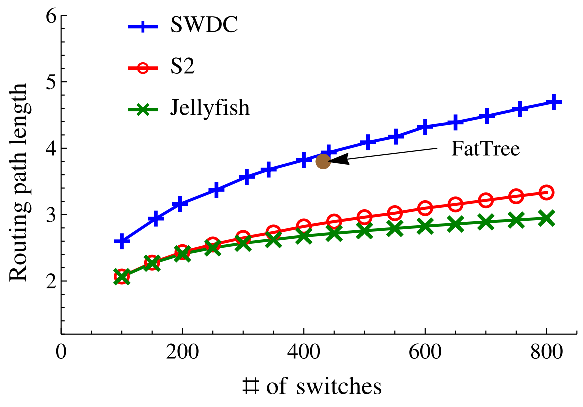


Fig. 18. Average routing path length of S2, SWDC, and Jellyfish.

SWDC 8-path routing. Clearly, SWDC single-path has the worst performance in completion time as well as fairness among flows. Jellyfish has similar results as S2, which is not plotted to make the figures clear.

6.7 Routing Path Length

Fig. 18 shows the average routing path length of S2, SWDC, and Jellyfish by varying the number of switches (12-port). Using 12-port switches, the FatTree topology contains exactly 180 switches, with a switch-to-switch path length equal to 3.81. The average path length of S2 is clearly shorter than that of SWDC and FatTree, and very close to that of Jellyfish, which uses shortest path routing. For 800-switch networks, the 90th percentile value is 8 for SWDC and 6 for S2 and Jellyfish. The 10th percentile values is 2 for all S2 and Jellyfish

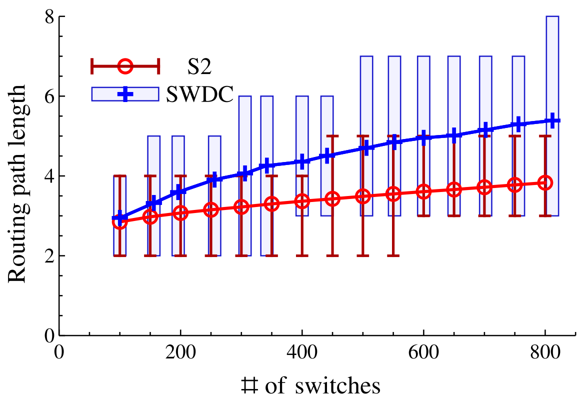


Fig. 19. Average path length of key-based routing.

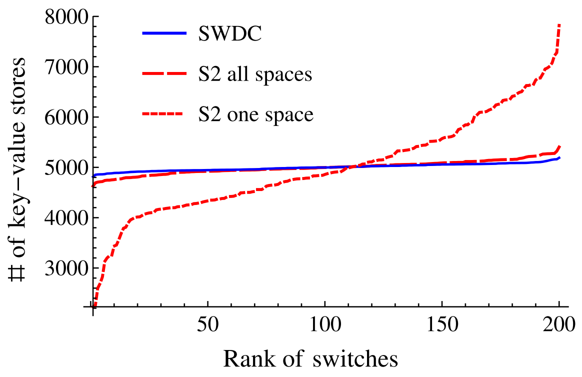


Fig. 20. Num of key-value stores per switch.

networks, and 3 for all SWDC networks with more than 500 switches. We do not plot the 10th and 90th values because they would make the figure too crowded. Results show that greediest routing in multiple spaces produces much smaller path lengths than greedy routing in a single space.

6.8 Key-Based Routing

For the performance of key-based routing, we measure routing path lengths, which reflect the store/retrieval latency, and balance of key-value load among switches.

Fig. 19 shows the average key-based routing path length of S2 and SWDC. Each value is the average path length of 100,000 different keys. Each data record is stored in 3-replication for both S2 and SWDC experiments. We find that S2 has significantly shorter routing paths, which also grow more slowly with the increase of network size compared to the SWDC paths. Fig. 20 shows the number of key-value stores per switch (we assume the switch-to-server key mapping is balanced). Since SWDC uses uniform node coordinates, it can achieve near-perfect load balance. If S2 only uses coordinates in one space, the key-values stores are very biased. However, when S2 applies replication in three spaces, key-value storage load distribution is almost as uniform as that of SWDC.

6.9 Failure Resiliency

Evaluation of failure resiliency of greedy routing protocol. In this set of experiments, we measure the routing performance of S2, SWDC, and Jellyfish, under switch link failures (a switch failure can be modeled as multiple link failures). We show the routing success rate versus the fraction of failed links in

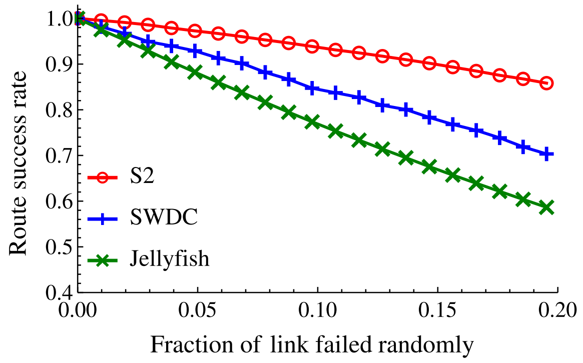


Fig. 21. Routing success rate versus failure fraction.

Fig. 21. S2 is very reliable under link failures. When 20 percent links fail, the routing success rate is higher than 0.85. SWDC and Jellyfish perform clearly worse than S2. When 20 percent links fail, the routing success rate of SWDC is 0.70 and that of Jellyfish is 0.59. S2 uses greedy routing in multiple spaces, hence it is less likely to encounter local minimum under link failure compared to SWDC. Jellyfish has the worst resiliency because it uses pre-computed paths.

Fig. 22 shows the normalized average routing path length versus the failure fraction for the same set of experiments. We compare the routing path lengths with those without failures and plot the normalized values. We find that switch link failures have no big impact to the average routing path length of S2 and SWDC. However, we should notice that a fraction of paths are broken and their lengths cannot be measured.

Evaluation of the failure-recovery mechanism. In this set of experiments, we measure the performance of the failure-recovery protocol under link failures. Using 8-port 128-switch topologies, we randomly remove a fraction of links in the topology. We measure the network performance using the single-path permutation traffic. Each plot in Fig. 23 shows the maximum throughput of S2 using the failure-recovery protocol. We compare the throughput with that of networks without failures. Using the failure-recovery protocol, S2 is able to delivery all packets. However, due to the existence of link failures, the throughput reduces slightly. When 10 percent links fail, the throughput is still higher than 0.85 of that in the static networks.

We also measure the completion time of flows under failures. We show the average flow completion time and the all-flow completion time in Fig. 24. The all-flow completion time on the complete graph is normalized to 1. Somewhat surprisingly, we observe that the all-flow completion time does not go up when failure exists. When there are 8 percent failed links, the all-flow completion time is only 94 percent of that of the complete topology. We conjecture that this is because the paths discovered by the failure-resistant protocol is different from the greedy routing paths. As a result, some heavy loaded links are avoided. Meanwhile, the average flow completion time rises slightly as the fraction of failed links increases. When 20 percent of the links fail, the average flow completion time rises by 8.6 percent.

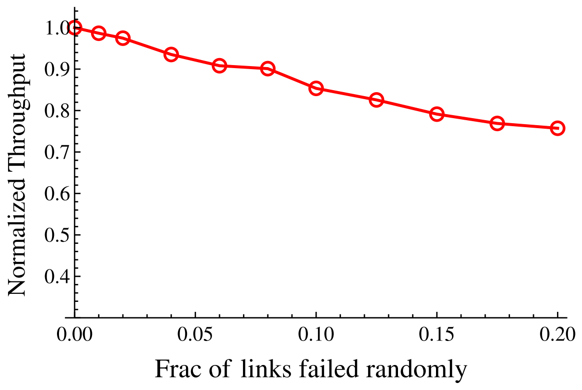


Fig. 23. Throughput versus failure fraction.

7 Discussion

7.1 Data Center Network Wiring

Labor and wiring expenses consume a significant part of financial budget of building a data center.

One important feature of hierarchical data center networks, such as FatTree, is that cabling can be simplified by taking the advantage of the local connection links within a pod. However, wiring random connection based topologies such as S2 and Jellyfish is complex. It is mainly because of the links that may connect two arbitrary switches in the network. In fact, most links in S2 and Jellyfish are not “local”. Hence cables may be very long and hard to manage. Fortunately, the coordinates of S2 still provide some locality information of the switches and can be used to reduce the wiring complexity.

In an S2 topology, the majority of cables are inter-switch ones. Thus, we propose to locate the switches physically close to each other so that to reduce cable lengths as well as labor work.

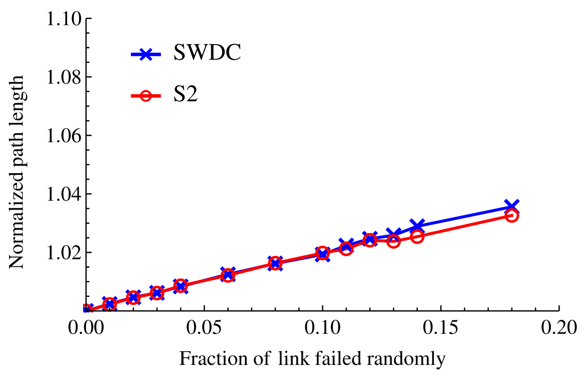


Fig. 22. Greedy routing path length versus failure fraction.

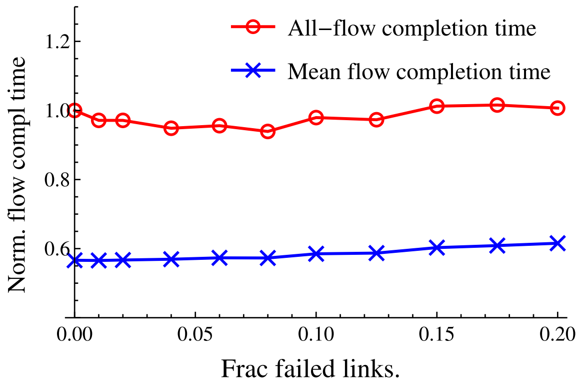


Fig. 24. Flow completion time versus failure fraction.

Benefits of coordinates. It is possible to accommodate the switches of an S2 network inside several standard racks. These racks can be put close to each other and we suggest to use aggregate cable bundles to connect them. The coordinates provide a way to help to arrange the inter-rack cables in order. A virtual space can be divided into several quadrants and we may allocate switches to racks based on corresponding quadrants.

For inner-rack cables, a unique method provided by the nature of coordinates, is using a patch panel that arranges the cables in order according to the coordinates. Coordinates make it possible to build aggregate bundle wires that are similar to flexible flat cables.

Hamedazimi et al. [38] proposed to use free-space optical communication in data center networks by putting mirrors and lens on switch racks and the ceiling of data center to reflect beams. Coordinates provide a unique way to locate the switches, and make it able to have these beams neatly ordered.

7.2 Server Multi-Homing

When the access bandwidth of servers becomes performance bottleneck, an effective method to improve data center throughput is using server multi-homing. For example, the Dual-homed FatTree has been proposed, which provides throughput improvement with Multipath TCP (MPTCP) for a wide range of practical workloads [39]. S2 is also able to adopt multi-homing servers and MPTCP. When a server is connected to multiple S2 switches, it has multiple routable addresses, i.e., the coordinates of its access switches. The sender can split the flow to the receiver into multiple subflows, each of which is routed to different access switches using different coordinates. These subflows can be maintained by MPTCP or by multiple TCP sessions separately. Since we do not focus on resolving access bandwidth bottleneck in this paper, we leave a comprehensive study of S2 multi-homing in future work.

7.3 Direct Server Connection

Although S2 is proposed to interconnect ToR switches, we may also use the S2 topology to connect servers directly and forward packets using S2 routing protocols. Similar approaches are also discussed in CamCube [13] and SWDC [7]. There are mainly two key advantages to use this topology. First, greedy routing on a server-centric topology can effectively implement custom routing protocols to satisfy different application-level requirements. This service is called symbiotic routing [13]. Second, hardware acceleration such as GPUs and NetFPGA can be used for packet switching to improve routing latency and bandwidth [7].

7.4 Switch Heterogeneity

S2 can be constructed with switches of different port numbers. The multiple ring topology requires each switch should have at least $2L$ inter-switch ports. According to Fig. 3 and other experimental results, five spaces are enough to provide good network performance. It is reasonable to assume that every switch in the network has at least 10 inter-switch ports. Switches with less ports may carry fewer servers to maintain the required inter-switch port number.

7.5 Possible Implementation Approaches

We may use open source hardware and software to implement S2's routing logic such as NetFPGA. S2's routing logic only includes simple arithmetic computation and numerical comparison and hence can be prototyped in low cost.

S2 can also be implemented using existing open-source software utilities such as the Click Modular Router [18]. Using Click, an S2 switch can be implemented as an *element*. The coordinates of the switch is passed to the element as the initialization parameter. The elements pulls packets from the incoming ports, computes the index of the nexthop port and pushes the packet to the output port accordingly.

On x86 platforms, S2 can be implemented as an application using the Intel Data Plane Development Kit (DPDK) [19], which enables programmer to develop programs that processes packets. The application reads packets from the Rx queue, computes the next hop ports and send packets towards the corresponding Tx queue.

Any forwarding algorithm can be implemented using Click and DPDK. Since S2 only requires arithmetic computation, the time complexity of S2 forwarding on Click and DPDK is small.

8 Conclusion

The key technical novelty of this paper is in proposing a novel data center network architecture that achieves all of the three key properties: high-bandwidth, flexibility, and routing scalability. The significance of this paper in terms of impact lies in that greediest routing of S2 is the first greedy routing protocol to enable high-throughput multi-path routing. In addition, S2 supports efficient key-based routing for various data center applications. We conduct extensive experiments to compare S2 with two recently proposed data center networks, SWDC and Jellyfish. Our results show that S2 achieves the best of both worlds. Compared to SWDC, S2 provides shorter routing paths and higher throughput. Compared to Jellyfish, S2 demonstrates significant lead in scalability while provides likewise high throughput and bisectional bandwidth. We expect greedy routing using multiple spaces may also be applied to other large-scale network environments such as peer-to-peer systems due to its scalability and efficiency.

Acknowledgments

The preliminary version of this paper appeared at the IEEE International Conference on Network Protocols (ICNP), 2014. The authors were supported by National Science Foundation grant CNS-1464335 and University of Kentucky College of Engineering Startup Grant.

References

[1] A. Greenberg, J. Hamilton, D. A. Maltz, and P. Patel, "The cost of a cloud: Research problems in data center networks," *ACM Sigcomm Comput. Commun. Rev.*, vol. 39, pp. 68–73, 2009.
[2] J. Dean and S. Ghemawat, "Mapreduce: Simplified data processing on large clusters," *Commun. ACM*, vol. 51, pp. 107–113, 2008.
[3] (2015). Apache hadoop [Online]. Available: http://hadoop.apache.org/.
[4] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly, "Dryad: Distributed data-parallel programs from sequential building blocks," in *Proc. 2nd ACM SIGOPS/EuroSys Eur. Conf. Comput. Syst.*, 2007, pp. 59–72.

[5] (2013). D. Realty 2013: What is driving the north america/europe data center market [Online]. Available: http://www.digitalrealty.com/us/knowledge-center-us/?cat=Research.

[6] M. Yu, A. Fabrikant, and J. Rexford, "Buffalo: Bloom filter forwarding architecture for large organizations," in *Proc. 5th Int. Conf. Emerging Netw. Exp. Technol.*, 2009, pp. 313–324.

[7] J.-Y. Shin, B. Wong, and E. G. Sirer, "Small-world datacenters," in *Proc. 2nd ACM Symp. Cloud Comput.*, 2011, p. 2.

[8] C. Qian and S. Lam, "ROME: Routing On metropolitan-scale Ethernet," in *Proc. IEEE Int. Conf. Netw. Protocols*, 2012, pp. 1–10.

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

[10] A. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. A. Maltz, P. Patel, and S. Sengupta, "Vl2: A scalable and flexible data center network," in *Proc. ACM SIGCOMM*, 2009, pp. 51–62.

[11] R. N. Mysore, A. Pamboris, N. Farrington, N. Huang, P. Miri, S. Radhakrishnan, V. Subramanya, and A. Vahdat, "Portland: A scalable fault-tolerant layer 2 data center network fabric," *SIGCOMM Comput. Commun. Rev.*, vol. 39, pp. 39–50, 2009.

[12] C. Guo, H. Wu, K. Tan, L. Shi, Y. Zhang, and S. Lu, "Dcell: A scalable and fault-tolerant network structure for data centers," in *Proc. ACM SIGCOMM*, 2008, pp. 75–86.

[13] H. Abu-Libdeh, P. Costa, A. Rowstron, G. O'Shea, and A. Donnelly, "Symbiotic routing in future data centers," *ACM SIGCOMM Comp. Commun. Review*, vol. 41, no. 4, pp. 51–62, 2011.

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

[15] A. Singla, P. B. Godfrey, and A. Kolla, "High throughput data center topology design," in *Proc. 9th USENIX Conf. Netw. Syst. Des. Implementation*, 2014, p. 17.

[16] C. Qian and S. S. Lam, "Greedy distance vector routing," in *Proc. IEEE 31st Int. Conf. Distrib. Comput. Syst.*, Jun. 2011, pp. 857–868.

[17] S. S. Lam and C. Qian, "Geographic routing in d-dimensional spaces with guaranteed delivery and low stretch," *IEEE/ACM Trans. Netw.*, vol. 21, no. 2, pp. 663–677, Apr. 2013.

[18] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek, "The click modular router," *ACM Trans. Comput. Syst.*, vol. 18, no. 3, pp. 263–297, 2000.

[19] (2015). Data plane development kit [Online]. Available: URL http://dpdk. org.

[20] C. Clos, "A study of non-blocking switching networks," *Bell Syst. Tech. J.*, vol. 32, no. 2, pp. 406–424, 1953.

[21] C. Guo, G. Lu, D. Li, H. Wu, X. Zhang, Y. Shi, C. Tian, Y. Zhang, and S. Lu, "Bcube: A high performance, server-centric network architecture for modular data centers," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 39, no. 4, pp. 63–74, 2009.

[22] L. Gyarmati and T. A. Trinh, "Scafida: A scale-free network inspired data center architecture," *ACM Sigcomm Comput. Commun. Rev.*, vol. 40, pp. 4–12, 2010.

[23] A. R. Curtis, S. Keshav, and A. Lopez-Ortiz, "Legup: Using heterogeneity to reduce the cost of data center network upgrades," in *Proc. ACM CoNEXT*, 2010, pp. 14:1–14:12.

[24] A. R. Curtis, T. Carpenter, M. Elsheikh, A. Lopez-Ortiz, and S. Keshav, "Rewire: An optimization-based framework for unstructured data center network design," in *Proc. IEEE INFOCOM*, 2012, pp. 1116–1124.

[25] E. Bouillet, *Path Routing in Mesh Optical Networks*. Hoboken, NJ, USA: Wiley, 2007.

[26] B. Stephens, A. Cox, W. Felter, C. Dixon, and J. Carter, "PAST: Scalable ethernet for data centers," in *Proc. 8th Int. Conf. Emerging Netw. Exp. Technol.*, 2012, pp. 49–60.

[27] C. Kim, M. Caesar, and J. Rexford, "Floodless in SEATTLE: A scalable ethernet architecture for large enterprises," in *Proc. SIGCOMM*, 2008, pp. 3–14.

[28] C. Hopps, "Analysis of an equal-cost multi-path algorithm," *RFC 2992*, 2000 [Online]. Available: https://tools.ietf.org/html/rfc2992.

[29] W. Cui and C. Qian, "Difs: Distributed flow scheduling for adaptive routing in hierarchical data center networks," in *Proc. 10th ACM/IEEE Symp. Archit. Netw. Commun. Syst.*, 2014, pp. 53–64.

[30] H. David and H. Nagaraja. (2004). *Order Statistics*. Wiley [Online]. Available: http://books.google.com/books?id=bdhzFXg6xFkC

[31] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels, "Dynamo: Amazon's highly available key-value store," *SIGOPS Oper. Syst. Rev.*, vol. 41, no. 6, pp. 205–220, Oct. 2007.

[32] D. Beaver, S. Kumar, H. C. Li, J. Sobel, and P. Vajgel, "Finding a needle in haystack: Facebook's photo storage," in *Proc. 9th USENIX Conf. Operating Syst. Des. Implementation*, 2010, pp. 1–8.

[33] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat, "Hedera: Dynamic flow scheduling for data center networks," in *Proc. 7th USENIX Conf. Netw. Syst. Des. Implementation*, 2010, p. 19.

[34] (2015). Ibm cplex optimizer [Online]. Available: http://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/

[35] J. Yen, "Finding the k shortest loopless paths in a network," *Manage. Sci.*, vol. 17, pp. 712–716, 1971.

[36] (2015). Implementation of k-shortest path algorithm [Online. Available: http://code.google.com/p/k-shortest-paths/.

[37] R. Jain, D. Chiu, and W. Hawe, "A quantitative measure of fairness and discrimination for resource allocation in shared computer systems," DEC Res., Maynard, MA, USA, Tec. Rep. TR-301, 1984.

[38] N. Hamedazimi, Z. Qazi, H. Gupta, V. Sekar, S. R. Das, J. P. Longtin, H. Shah, and A. Tanwer, "Firefly: A reconfigurable wireless data center fabric using free-space optics," in *Proc. ACM SIGCOMM*, 2014, pp. 319–330.

[39] C. Raiciu, S. Barre, C. Pluntke, A. Greenhalgh, D. Wischik, and M. Handley, "Improving datacenter performance and robustness with multipath TCP," in *Proc. ACM SIGCOMM*, 2011, pp. 266–277.

Ye Yu received the BSc degree from Beihang University and is currently working toward the PhD degree at the Department of Computer Science, University of Kentucky. His research interests including data center networks and software defined networking. He is a student member of the IEEE.

Chen Qian (M'08) received the BSc degree from Nanjing University in 2006, the MPhil degree from the Hong Kong University of Science and Technology in 2008, and the PhD degree from the University of Texas at Austin in 2013, all in computer science. He is an assistant professor at the Department of Computer Science, University of Kentucky. His research interests include computer networking and distributed systems. He has published more than 30 research papers in highly competitive conferences and journals including ACM SIGMETRICS, ACM CoNEXT, IEEE ICNP, IEEE ICDCS, IEEE INFOCOM, IEEE PerCom, *IEEE/ACM Transactions on Networking*, and *IEEE Transactions on Parallel and Distributed Systems*. He is a member of the IEEE and ACM.