Toward A Scalable, Fault-Tolerant, High-Performance Optical Data Center Architecture

Kai Chen, Xitao Wen, Xingyu Ma, Yan Chen, *Fellow, IEEE*, Yong Xia, *Senior Member, IEEE*, Chengchen Hu, Qunfeng Dong, and Yongqiang Liu

***Abstract*— Optical data center networks (DCNs) are becoming increasingly attractive due to their technological strengths compared with the traditional electrical networks. However, existing optical DCNs are either hard to scale, vulnerable to single point of failure, or provide limited network bisection bandwidth for many practical data center workloads. To this end, we present WaveCube, a scalable, fault-tolerant, high-performance optical DCN architecture. To scale, WaveCube removes MEMS,[1] a potential bottleneck, from its design. WaveCube is fault-tolerant, since it does not have single point of failure and there are multiple node-disjoint parallel paths between any pair of top-of-rack switches. WaveCube delivers high performance by exploiting multi-pathing and dynamic link bandwidth along the path. For example, our evaluation results show that, in terms of network bisection bandwidth, WaveCube outperforms prior optical DCNs by up to 400% and is 70%–85% of the ideal non-blocking network (i.e., theoretical upper bound) under both realistic and synthetic traffic patterns. WaveCube's performance degrades gracefully under failures—it drops 20% even with 20% links cut. WaveCube also holds promise in practice—its wiring complexity is orders of magnitude lower than Fattree, BCube, and c-Through at scale, and its power consumption is 35% of them.**

***Index Terms*— Data center networks, network structure, optical networking.**

Manuscript received March 25, 2015; revised February 6, 2016 and January 28, 2017; accepted March 7, 2017; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor S. Sengupta. Date of publication April 14, 2017; date of current version August 16, 2017. This work was supported by the Hong Kong RGC under Grant GRF-16203715, Grant ECS-26200014, Grant GRF-613113, and Grant CRF-C703615G. *(Corresponding author: Kai Chen.)*

K. Chen is with The Hong Kong University of Science and Technology, Hong Kong (e-mail: kaichen@cse.ust.hk).

X. Wen is with Google, Mountain View, CA 94043 USA (e-mail: xitao.wen@gmail.com).

X. Ma is with the University of California, Los Angeles, CA 90095, USA (e-mail: mxy020426@gmail.com).

Y. Chen is with Northwestern University, Evanston, IL 60208 USA (e-mail: ychen@northwestern.edu).

Y. Xia is with Microsoft, Redmond, WA 98052 USA (e-mail: xy12180@gmail.com).

C. Hu is with Xi'an Jiaotong University, Xi'an 710048, China (e-mail: huc@ieee.org).

Q. Dong is with DataBox Ltd., Suzhou 215123, China (e-mail: dong.qunfeng@gmail.com).

Y. Liu is with NEC Labs China, Beijing 100084, China (e-mail: liuyq7809@gmail.com).

Digital Object Identifier 10.1109/TNET.2017.2688376

[1]Micro-Electro-Mechanical-System–one of the most popular optical circuit switches used as the main component by many recently-proposed optical DCNs [15], [18], [39].

I. INTRODUCTION

A. Motivation

Nowadays, data centers are being built around the world to support various Internet applications and cloud services such as web search, online social networks, scientific computing, and data analysis (*e.g.*, GFS [20], Mapreduce [17] and Dryad [26]). As a result, people have been investigating new data center structures with the goal to better meet the bandwidth requirement of these applications. The current representative designs include pure electrical structures (*e.g.*, BCube [22], DCell [23], Fattree [10], PortLand [32], VL2 [21], CamCube [9], and Jellyfish [37]) and optical/electrical structures (*e.g.*, Helios [18], c-Through [39], OSA [15], and Mordia [34]). However, these existing DCN designs fall short.

Electrical DCNs: Initially, people statically provision uniformly high capacity between all servers using sophisticated designs like Fattree [10], BCube [22], and VL2 [21] with pure electrical devices. While this approach seems to be the only way to prevent any communication bottleneck assuming arbitrary traffic demands, it suffers from significant wiring challenge and management complexity. Furthermore, full bisection bandwidth at the scale of the entire data center is not necessary given not so many real applications require such high bandwidth at this scale [18]. This leads to a dilemma: the network must be fully provisioned against any localized congestion despite the fact that, at any time, certain parts of network are rarely used or even sit idle.

In addition, the industry recently has an increasing trend towards deploying 10 GigE NICs at end hosts. As an evidence, Google has already deployed 10 GigE and is pushing for 40/100 GigE [29], [31], [35]. However, copper cables would be unsustainable for 10 GigE over 10 meters due to the intrinsic tradeoff between the length, power budget, and high electrical loss at higher data rate [18]. This results in another dilemma for high-speed electrical DCN designs.

Optical DCNs: To solve these dilemmas, optical network technologies, due to their ability to dynamically provision bandwidth resources across the network and support high bit-rate (*e.g.*, 64 Tb/s) over long distance, have been introduced in recent optical DCNs such as Helios [18], c-Through [39], OSA [15], and Mordia [34]. Most of these

TABLE I
SUMMARY OF PRIOR OPTICAL DCNS AND COMPARISON TO WAVECUBE

Optical DCNs	Scalability (port count)	Performance	Fault-tolerance
c-Through [39] Helios [18]	Low (∼1000)	Low	No
OSA [15]	Low (∼2000)	High	No
Mordia [34]	Low (∼88)	High	No
WaveCube	High (unlimited)	High	Yes

optical designs explore the reconfigurability of MEMS-based[2] optical switches to set up optical circuits for bandwidth demanding parts of the network. They have made significant contributions in pointing out a promising direction for building *thin* dynamic structures instead of the *fat* static ones as above. However, we find that the following important issues have not been fully attended in these existing optical DCNs.

- **Scalability:** MEMS is the central switch to connect all ToR switches, thus the low port density of MEMS limits the scalability. For example, today's largest MEMS has 320 ports and only supports 2560 servers in OSA [15]. A natural way to scale is to interconnect multiple MEMSes in the form of multi-stage fattree [39]. Unfortunately, as we will show in Section II, the properties of MEMS make it (both technically and economically) hard to scale optical DCNs in this way. Practical and economic scaling of optical DCNs still remains a challenge.
- **Performance:** c-Through and Helios set up one-hop optical circuits between ToRs on-demand but with low fan-in/out, which greatly restricts their performance when hotspots occur with high fan-in/out – a common pattern in real DCN workloads [25], [28]. For example, people see real cases where even the top 5 ToR neighbors cumulatively add up to a small fraction of load on hotspots [25]. In such scenarios, c-Through circuits can only offload less than 20% of the traffic since a ToR can only connect one other ToR at a time. OSA [15] solves this problem via multi-hop routing on a k-regular topology (each ToR connects k other ToRs). However, there is a tradeoff between k and network scale.
- **Fault-tolerance:** Both c-Through and OSA have all ToRs connect to a central MEMS, creating a single point of failure. Mordia [34] resides on a ring, any link cut will break the ring and affect the connectivity. Helios can have multiple MEMSes to connect all Pods to increase fault-tolerance, however, one additional MEMS will impose $160,000 cost, which is expensive.

Considering these, our goal is to design a scalable, fault-tolerant and high performance optical DCN architecture (Table I). To the best of our knowledge, none of existing optical DCNs can achieve all these properties simultaneously.

B. Our Approach and Contributions

Given that MEMS is the bottleneck for scalability and it is hard to interconnect multiple MEMSes to scale, we take the contrary approach: instead of adding multiple MEMSes, we completely remove it from our design. Without MEMS, the network can easily scale. As a side-effect, however, we lose the dynamic topology. But this gives us a chance to develop more advanced routing mechanisms which otherwise cannot be easily achieved in a dynamic topology.

Our general design principle in WaveCube is to use multi-path routing and dynamic link bandwidth scheduling on each path to compensate the loss of not having dynamic topology, while achieving scalability. Furthermore, after removing MEMS, fault-tolerance can be obtained since we eliminate the single point of failure.

This paper makes the following contributions:

- We design WaveCube, a new MEMS-free optical DCN architecture that achieves scalability, fault-tolerance, and high-performance simultaneously (Section III). Specifically, WaveCube easily scale up to hundreds of thousands of servers. It delivers high performance and fault-tolerance by exploiting multi-pathing and dynamic link bandwidth on each path—in terms of network bisection bandwidth, WaveCube outperforms prior optical DCNs by up to 400% and is 70%-85% of the ideal non-blocking network with both realistic and synthetic traffic patterns; its performance degrades gracefully in case of failures—a 20% drop even with 20% links cut.
- By exploiting WaveCube topology properties, we introduce a polynomial-time optimal solution to wavelength assignment for dynamic link bandwidth (Section IV), which is a challenging problem remains unsolved in previous optical DCN [15].
- We inspect the practical deployment issues of WaveCube, and show that it holds promise in practice (Section VI). For example, using a practical model, we find that WaveCube is easy to build—its wiring complexity is 2-3 orders of magnitude simpler than Fattree/BCube and 1 order simpler than c-Through at large scale. Furthermore, it incurs low cost and consumes minimal power of all.

It is also worthwhile to note that WaveCube achieves all its design goals without requiring any advanced, expensive optical devices beyond what are used by existing optical DCNs [15], [18], [34], [39]. Our strategy is to better orchestrate them to realize our aims. We have presented a hardware feasibility analysis for implementing WaveCube, however, building a non-trivial, fully functional WaveCube prototype is our next step effort and is beyond the scope of this paper. Our hope is that the design, analysis, and extensive simulations conducted in this paper will pave the way for the next step of prototyping.

II. BACKGROUND

Optical network technologies have been extensively introduced in optical DCNs [15], [18], [39]. We overview the main devices and their properties. For more details, please refer to those papers.

MEMS-based Optical Switch: MEMS works on the physical layer. It is a bipartite $N \times N$ circuit switching matrix, which allows any input port to be connected to any one of the output

[2]While Helios/c-Through/OSA use 3D MEMS based switches, Mordia [34] uses Nistica wavelength selective switches which contain Texas Instruments DLP 2D MEMS-based switches. Note that such 2D technology has different optical properties, *e.g.*, smaller port count, faster switching speed, and greater insertion loss, *etc*.

ports by mechanically rotating micro-mirrors. The switching time of MEMS is around 10 milliseconds [38].

Wavelength Selective Switch (WSS): A WSS unit is a $1\times N$ optical device for wavelength de-multiplexing. It has one common incoming port and N outgoing ports. It can divide all the wavelengths from the common incoming port into N groups, with each group going via an outgoing port. The WSS is run-time reconfigurable (around 10 milliseconds).

Wavelength Division Multiplexing (WDM): WDM encodes multiple non-conflict wavelengths onto a single fiber. Depending on the channel spacing, up to 100 wavelengths can be carried on a fiber in the conventional or C-band. In optical DCNs, a wavelength is usually rate-limited by the port of the electrical switch it is connected to, *e.g.*, 10Gbps.

Others: There are other optical devices such as circulator, transceiver, coupler, etc. Circulator enables bidirectional transmission over a fiber so that MEMS ports can be used efficiently. Transceiver converts between electrical and optical signals on ToR switches. Coupler multiplexes multiple wavelengths onto a fiber (similar but simpler than multiplexer).

III. The WaveCube Architecture

In this section, we present the WaveCube architecture. We first introduce its topology, multi-pathing and dynamic link bandwidth. Then, we show how to use multi-pathing and dynamic link bandwidth for network performance.

A. WaveCube Topology

In WaveCube (Figure 1), servers are connected to ToRs, and ToRs are directly connected to each other via optical components which provide dynamic link bandwidth between ToRs (Section III-C). There is no aggregate or core layers. At ToR level, it is a n-dimensional cube where the ith dimension has k_i ToRs in a loop, *i.e.*, a $(k_{n-1}, k_{n-2}, \cdots, k_0)$-radix topology.[3] In our design, we assume every k_i is even.

Each ToR has an address array $A = (a_{n-1}, a_{n-2}, \cdots, a_0)$, where $a_i \in [0, k_i - 1]$.

The distance between two ToRs A and B, which we call *WaveCube distance*, is $D_W(A,B) = \sum_{i=0}^{n-1} \omega(a_i - b_i)$, where $\omega(a_i - b_i) = min\{|a_i - b_i|, k_i - |a_i - b_i|\}$. For example, in Figure 1, where $n = 2$ and $k_0 = k_1 = 4$, $D_W((1,1),(3,4)) = 2 + 1 = 3$. Two ToRs A and B are neighbors in WaveCube if and only if $D_W(A,B) = 1$. In other words, their address arrays only differ in one dimension, and only differ by 1 (mod k_i).[4]

Lemma 1: A WaveCube network is composed of $\prod_{i=0}^{n-1} k_i$ ToRs and $n\prod_{i=0}^{n-1} k_i$ ToR links (bidirectional).

Lemma 2: The diameter of a WaveCube network (i.e., the longest shortest path between all ToR pairs) is $\sum_{i=0}^{n-1} \frac{k_i}{2}$.

Scalable Topology: Lemma 1 and Lemma 2 indicate that a k-ary-n-cube WaveCube contains $N = k^n$ ToRs and its diameter is $\frac{nk}{2} = \frac{n\sqrt[n]{N}}{2} = \frac{k\log_k^N}{2}$, which shows that WaveCube diameter scales nicely with the number of ToRs. The total number of optical links scales linearly with and is always n times the number of ToRs. For example, using a 4-ary-8-dimensional WaveCube, 65,536 ToRs can be connected into a DCN whose diameter is 16, accommodating 2,097,152 servers (assuming 32 servers per ToR).

Centralized Control: Note that WaveCube employs a central controller to manage the network, such as fault-tolerant routing, bandwidth scheduling, *etc*. This is inspired by many other DCN designs [15], [18], [21], [22], [32], [39].

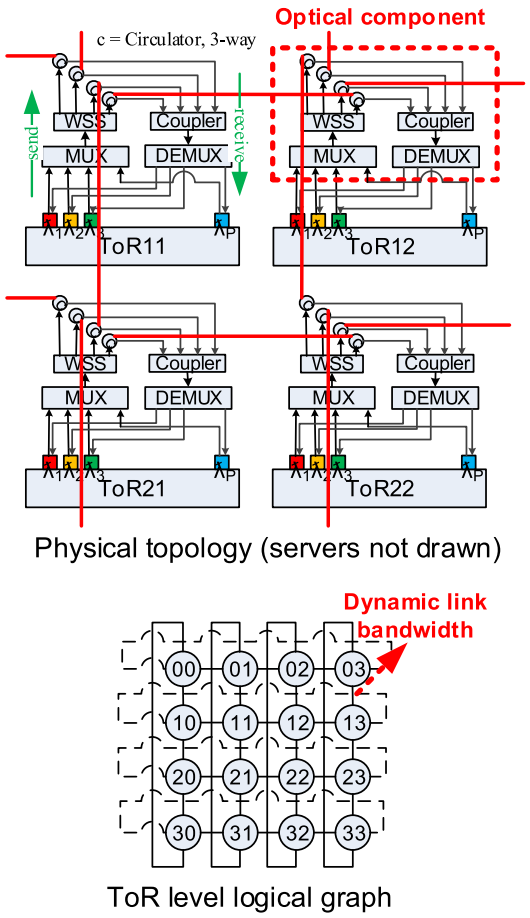


Fig. 1. The WaveCube architecture.

B. Multi-pathing: Routing and Fault-tolerance

Node-disjoint paths between two ToRs provide a means of selecting alternate routes and increase fault-tolerance. WaveCube provides $2n$ node-disjoint paths between every pair of ToRs, which is maximum since every ToR is only connected to $2n$ neighbor ToRs.

Theorem 1: WaveCube contains $2n$ node-disjoint paths between every pair of ToRs.

Proof: Suppose $D_W(A,B) = l$ and $D_H(A,B) = h$ (Hamming distance, *i.e.*, the number of digits on which A and B differ), we can find $2n$ node-disjoint paths as follows. (Without loss of generality, assume A and B differ on the lowest h digits.)

[3]Essentially, WaveCube is a generalized k-ary-n-cube [16] with variable radices. CamCube [9] also used a k-ary-n-cube, 3D Torus, for its server-centric network topology. WaveCube differs from CamCube in that it is switch-centric and, more importantly, the link bandwidth of WaveCube can be dynamically adjusted.

[4]WaveCube distance is similar to Lee distance [4], but with different radix values in certain dimensions.

Class 1 (h paths of length l): For each i $(0 \le i \le h-1)$, starting from A, we first "correct" the ith digit in $|a_i - b_i|$ steps along the shortest path on the ith dimension, adding 1 to a_i or subtracting 1 from a_i at each step; then, we repeat the same correction process on the $(i+1)$th digit, $(i+2)$th digit, ..., $(h-1)$th digit, 0th digit, 1st digit, ..., $(i-1)$th digit, respectively; a l-hop path from A to B is thus formed. By finding such a l-hop path for every i $(0 \le i \le h-1)$, h node-disjoint paths of length l can be found. For example, between two ToRs $(0,0)$ and $(0,2)$ in Figure 1, we can find one such path — $(0,0)(0,1)(0,2)$.

Class 2 (h Paths of Length $l + k_i - 2d_i$): where $d_i = min(|a_i - b_i|, k_i - |a_i - b_i|)$, $(0 \le i \le n-1)$. For each i $(0 \le i \le h-1)$, we move one step along the longest path from a_i to b_i on the ith dimension, say from ToR A to ToR $A' = (a_{n-1}, \cdots, a_{i+1}, a_i - 1, a_{i-1}, \cdots, a_0)$; then, we find a Class 1 path of length l between A' and $B' = (b_{n-1}, \cdots, b_{i+1}, a_i - 1, b_{i-1}, \cdots, b_0)$; finally, we move on from B' to B, along the longest path from a_i to b_i on the ith dimension; a $(l + k_i - 2d_i)$-hop path from A to B is thus formed. This gives us h node-disjoint paths of length $l + k_i - 2d_i$. For example, between two ToRs $(0,0)$ and $(0,2)$ in Figure 1, we can find one such path — $(0,0)(0,3)(0,2)$.

Class 3 ($2(n-h)$ Paths of Length $l+2$): For each i $(h \le i \le n-1)$, we first move one step from A to B on the ith dimension, from ToR A to ToR $A' = (a_{n-1}, \cdots, a_{i+1}, a_i + 1, a_{i-1}, \cdots, a_0)$; then, we can find a Class 1 path of length l between A' and $B' = (b_{n-1}, \cdots, b_{i+1}, b_i + 1, b_{i-1}, \cdots, b_0)$; finally, we move one step along the ith dimension, from B' back to B; a (l+2)-hop path from A to B is thus formed. Similarly, we can also find such a (l+2)-hop path via $A'' = (a_{n-1}, \cdots, a_{i+1}, a_i - 1, a_{i-1}, \cdots, a_0)$ and $B'' = (b_{n-1}, \cdots, b_{i+1}, b_i - 1, b_{i-1}, \cdots, b_0)$. By finding such a pair of $(l+2)$-hop paths for every i $(h \le i \le n-1)$, $2(n-h)$ node-disjoint paths of length $l+2$ are found. For example, between two ToRs $(0,0)$ and $(0,2)$ in Figure 1, we can find two such paths — $(0,0)(1,0)(1,1)(1,2)(0,2)$ and $(0,0)(3,0)(3,1)(3,2)(0,2)$.

In total, we have found $h+h+2(n-h) = 2n$ paths between A and B, and it is clear from the above path finding process that the paths are all node-disjoint. ∎

WaveCube's $2n$ node-disjoint paths are efficient for high-performance routing, load-balancing, and fault-tolerance.

C. Dynamic Link Bandwidth

In addition to multi-pathing, WaveCube enables dynamic link bandwidth on each path using optical components (mainly WSS).

Take ToR11 in Figure 1 for example, as a sender, the WSS receives all wavelengths from the ToR, which are multiplexed onto a single fiber by the MUX. Then, the WSS divides these wavelengths into K groups, each group of wavelengths going to another ToR through one of the K outgoing links/ports of the WSS. The number of wavelengths assigned to a ToR link amounts to the bandwidth of that link. For example, if the WSS incoming port receives 40 wavelengths, it can route wavelengths 1–5 to outgoing port 1, 11–20 to port 2, 22–30 to port 3 etc. Then, links 1, 2, 3 are assigned 5, 10, 9 units of bandwidth (*i.e.*, 50Gbps, 100Gbps, 60Gbps if each ToR port is 10Gbps), respectively.

As a receiver, the coupler multiplexes K groups of wavelengths received from K other ToRs to a single fiber, and the DEMUX de-multiplexes all the wavelengths to their respective ports on ToR.

However, wavelength contention requires that the same wavelength cannot be assigned to a ToR link twice simultaneously. This poses a challenge to fully use the property of dynamic link bandwidth, since non-contention wavelength assignment is NP-hard and has not been solved in existing optical DCN [15]. As shown later in Section IV, we make a new contribution in designing an optimal wavelength assignment algorithm by taking advantage of the WaveCube topology properties.

It is worthwhile to note that WaveCube's use of WSS is inspired by OSA [15], but WaveCube significantly outperforms OSA by designing the optimal wavelength assignment and optimized wavelength adjustment algorithms (Section IV).

D. Optimization with The Above Two Properties

To optimize network performance using multi-pathing and dynamic link bandwidth, we schedule flows over the multiple paths and then dynamically provision link bandwidth to fit the resulted traffic.

Per-Flow Scheduling: For each incoming flow, the central controller decides, among $2n$ parallel paths, which one to route the flow. There are many known scheduling methods to use, such as random, round-robin, ECMP, etc. Recent work Hedera [11] also introduced an advanced DCN flow scheduling method. However, WaveCube does not require sophisticated (possibly high-overhead) per-flow scheduling, since it has dynamic link bandwidth. We just distribute traffic among multiple paths randomly, and then dynamically allocate link bandwidth to handle possible congestion resulted from unbalanced flow scheduling. Our evaluation results show that this simple method works well.

Bandwidth Scheduling: The goal of link bandwidth scheduling is to find a link bandwidth assignment such that link utilization is optimized. Here, link utilization is defined as $\frac{\tau(u,v)}{c^{\phi}(u,v)}$, where $\tau(u,v)$ is the traffic volume on link (u,v), and $c^{\phi}(u,v)$ is the bandwidth assigned by ϕ to link (u,v). Given a traffic matrix T, we define an optimal bandwidth assignment as an assignment ϕ that minimizes the maximum link utilization. For that, we define a variable y^{ϕ}, which represents the reciprocal of the maximum link utilization, given by $\min_{(u,v)\in E}\{\frac{c^{\phi}(u,v)}{\tau(u,v)}\}$. The bandwidth scheduling problem is formulated as the following linear program.

$$Objective: \quad \max_{\phi} y^{\phi} \tag{1}$$

$$Subject\ to: \quad y^{\phi} \le \frac{c^{\phi}(u,v)}{\tau(u,v)}, \quad \forall \phi,\ \forall (u,v) \in E \tag{2}$$

$$\sum_{v \in V} c^{\phi}(u,v) \le C, \quad \forall u \in V \tag{3}$$

$$c^{\phi}(u,v) \in R^{+}, \quad \forall (u,v) \in E \tag{4}$$

The objective function (1) specifies the goal of maximizing y^{ϕ}, which is equivalent to minimizing the maximum link

TABLE II
SOME KEY NOTATIONS USED IN SECTION IV

Notation	Meaning
$G = (V, E)$	WaveCube ToR level topology graph
ϕ	bandwidth demand on E, computed in Section III-D
$G' = (V, E')$	multigraph representation of $G = (V, E, \phi)$
λ	wavelength assignment on E', that satisfies bandwidth demand ϕ
$G^r = (V, E^r)$	regular multigraph extended from G', E^r=E' + dummy edges

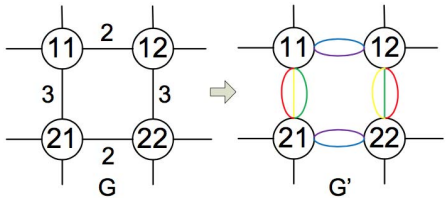


Fig. 2. Example of multigraph construction.

utilization. Constraint (2) states the correctness requirement that, in any feasible assignment ϕ, y^ϕ should be less than or equal to the reciprocal of the link utilization $\frac{\tau(u,v)}{c^\phi(u,v)}$ of any link (u, v). Constraint (3) shows that the total bandwidth assigned to the links incident to a node cannot exceed the node's capacity. Here, link bandwidth is denoted by the number of wavelengths carried on the link, which is a positive integer. Constraint (4) relaxes the integer to be a real number, which we will round back to an integer later.

While the traditional Internet TE has a similar optimization goal but through changing flow routing [19], WaveCube optimizes by adjusting link bandwidth. This linear program can be solved efficiently with tools such as Cplex and Glpk.

IV. WAVELENGTH ASSIGNMENT

After computing a bandwidth assignment ϕ, we need physically assign wavelengths to each link that is equal to the desired bandwidth of that link. In this section, we study two key problems for wavelength assignment:

- *Wavelength assignment:* What is the minimal number of wavelengths to implement ϕ?
- *Wavelength adjustment:* How to optimize wavelength adjustment during bandwidth re-assignment?

A. Optimal Wavelength Assignment

In WaveCube (Figure 1), each ToR up-port is bound to a fixed wavelength. The total wavelengths for each ToR are the same and equal to the number of up-ports. Further, due to wavelength contention introduced above, the same wavelength cannot be assigned to a ToR link twice simultaneously. Given a ϕ, we have to assign non-conflict wavelengths to satisfy ϕ. Notations are explained in Table II.

Problem 1 (Optimal Wavelength Assignment (OWA)): Given a WaveCube graph $G = (V, E, \phi)$ where ϕ is a link bandwidth assignment on E, find a non-conflict wavelength assignment λ on E to satisfy ϕ, such that the number of wavelengths used is minimized.

In $G = (V, E, \phi)$, each node in V is a ToR, each edge in E is a ToR link, and ϕ specifies the bandwidth demand on each link. Figure 2 (left) is an example of G with bandwidth demand specified. We translate it to a multigraph $G' = (V, E')$ (right), so that the number of edges between two ToRs in G' is equal to the bandwidth demand between them in G. Then, satisfying ϕ with the minimal non-conflict wavelengths is equivalent to an edge-coloring solution [2] on the multigraph, where each color represents a distinct wavelength and no two adjacent edges share the same color.

We note that the traditional Routing and Wavelength Assignment (RWA) [33] in optical networks is to select a suitable lightpath for each connection so that no two lightpaths share a link assigned the same wavelength. Moreover, in such networks without wavelength converters, the same wavelength must be used on all links along the lightpath (known as wavelength continuity constraint). The RWA problem has mostly been formulated as an integer programming problem that does not lend itself to efficient solution. In comparison, WaveCube leverages multi-hop optical links (with Optical-Electrical-Optical conversion) to build an end-to-end path, and each hop can use a different wavelength. As a result, OWA in WaveCube can be model as the edge coloring problem.

However, the edge coloring problem on a general multigraph G' is NP-complete [2], and the minimal number of colors needed in an edge coloring $\chi(G') \in [\Delta(G'), \Delta(G') + \mu(G')]$, where $\Delta(G')$ is the maximum node degree of G' and $\mu(G')$ is the multiplicity (*i.e.*, the maximum number of edges in any bundle of parallel edges).

This poses a challenge as $\Delta(G')$ equals to the total number of wavelengths available, while by theory it is possible to require as many as $\Delta(G') + \mu(G') = 2\Delta(G')$ in order to fully satisfy ϕ. This problem has not been solved in previous work [15]. However, WaveCube introduces a polynomial-time optimal wavelength assignment solution by taking advantage of the WaveCube topology properties.

Theorem 2: For any WaveCube graph $G = (V, E, \phi)$, we can always provision ϕ (without wavelength contention) using $\Delta(G')$ wavelengths.

First of all, it is clear that at least $\Delta(G')$ wavelengths are needed to provision ϕ. Otherwise, there are not enough wavelengths to avoid wavelength conflict at the ToR with maximal degree in G'. Theorem 2 guarantees that we can always use this minimum number of $\Delta(G')$ wavelengths to provision ϕ.

We will prove Theorem 2 by finding an edge-coloring solution on G' with $\Delta(G')$ colors (*i.e.*, wavelengths). This is a daunting goal since it is NP-hard on general topologies [2], [15]. Our novelty is that WaveCube is designed in such a way that its topology is guaranteed to be bipartite in nature, for which an elegant polynomial-time algorithm can be found for the optimal wavelength assignment. This innovative architectural property makes WaveCube the very first optical DCN of its kind. In the following, we first show the bipartite nature of WaveCube, and then briefly describe the polynomial-time algorithm for the optimal wavelength assignment.

Proof: To show its bipartite nature, we randomly select a node in a WaveCube topology and mark it "black", and then starting from this black node, we mark all its neighbors "white". In the following steps, for each white (or black) node, we mark its neighbors black (or white)

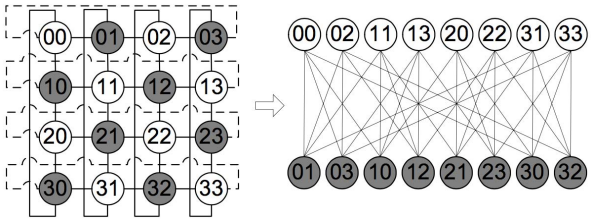


Fig. 3. An example of topology transformation.

iteratively until all the nodes are covered. Since in WaveCube, every radix k_i $(0 \le i \le n-1)$ is even, this procedure will converge. We then put all black nodes in one group and white nodes in another. This apparently is a bipartite graph since all the edges are between these two groups. Figure 3 illustrates an example of topology transformation.

We next theoretically prove the correctness. For that, we show how nodes in G (and G') can be partitioned into two sets, say V_1 and V_2, so that every pair of neighboring nodes in WaveCube must go to different sets and hence edges in WaveCube exist across V_1 and V_2 only. To prove that, for each node $u = (a_{n-1}, a_{n-2}, \cdots, a_0)$, if $\sum_{i=0}^{n-1} a_i$ is even, we put u in V_1 and otherwise V_2.

In WaveCube, two neighbors $u = (a_{n-1}, a_{n-2}, \cdots, a_0)$ and $v = (b_{n-1}, b_{n-2}, \cdots, b_0)$ differ on exactly one dimension, say dimension t, and we can assume without loss of generality that $b_t = (a_t + 1) \bmod k_t$. Since k_t is even, if a_t is odd, b_t must be even; if a_t is even, b_t must be odd. Therefore, if $\sum_{i=0}^{n-1} a_i$ is odd, $\sum_{i=0}^{n-1} b_i$ must be even; if $\sum_{i=0}^{n-1} a_i$ is even, $\sum_{i=0}^{n-1} b_i$ must be odd. This proves that u and v must go to different sets. Hence, G (as well as G') is a bipartite graph.

Given G' is a bipartite graph, a polynomial-time algorithm for coloring G' with $\Delta(G')$ colors consists of three steps: (1) We augment bipartite graph G' into a $\Delta(G')$-regular bipartite graph G^r by adding dummy edges $(\Delta(G^r) = \Delta(G'))$. A $\Delta(G')$-regular bipartite graph is a bipartite graph where the degree of every node is $\Delta(G')$. (2) Partition the edges in $\Delta(G^r)$ into $\Delta(G')$ perfect matchings via *Decomposition()* (described in Figure 4); (3) Assign a distinct color to each perfect matching, and G' is therein colored by $\Delta(G')$ colors (without wavelength conflict). Proof of Theorem 2 is completed. ■

In the 3 steps of coloring G', *Decomposition()* is critical. It finds $\Delta(G^r)$ perfect matchings in G^r in a divide-and-conquer manner. Its correctness is guaranteed by the fact that "any k-regular bipartite graph has a perfect matching" [36]. Given this, we can extract one perfect matching from the original graph, the residual graph is $(\Delta(G^r)-1)$-regular; then we extract the second perfect matching, *etc*, until ending up with $\Delta(G^r)$ perfect matchings. *Find_Perfect_Matching()* is a procedure to find a perfect matching in a regular bipartite graph we learned from previous work. We show its correctness in Appendix-A.

B. Optimized Wavelength Adjustment

In operation, network state may change and link bandwidth needs adjustment to better fit traffic. Once bandwidth demand ϕ changes, we need to re-assign wavelengths to satisfy the new ϕ accordingly. A naive approach is to assign the wavelengths from scratch without considering the old distribution. However, given that shifting a wavelength from one WSS port to another would incur ∼10ms latency, wavelength re-assignment should shift minimal wavelengths. This minimizes the disruption of ongoing traffic and is specially important to latency-sensitive flows.

Problem 2 (Minimal Wavelength Adjustment (MWA)): Given the WaveCube topology $G = (V, E)$, the old bandwidth distribution ϕ_o, the old wavelength distribution λ_o satisfying ϕ_o, and the new bandwidth demand ϕ_n, find a wavelength assignment λ_n satisfying ϕ_n such that, from $\lambda_o \rightarrow \lambda_n$, the shifting of wavelengths is minimal.

We formulate MWA problem as a 0-1 integer linear program, and prove it is NP-hard. We then design a heuristic algorithm in Figure 5. The basic idea is to use the old wavelength distribution $\lambda_o = \{m_1, m_2, \cdots, m_\Delta\}$ to assist the

Decomposition(G^r): /* decompose G^r into $\Delta(G^r)$ perfect matchings */
1 if ($d = 1$)
2 return $M = G^r$; /* G^r itself is a perfect matching */
3 else
4 if (d is odd)
5 M = *Find_Perfect_Matching*(G^r);
6 $G^r = G^r \setminus M$;
7 find all Euler cycles in G^r, and pick every other edge on each cycle to
form two $\lfloor d/2 \rfloor$-regular graphs: G_1 and G_2;
8 return $M \cup$ *Decomposition*(G_1) $\cup$ *Decomposition*(G_2);

Find_Perfect_Matching(G^r): /* find a perfect matching M in G^r */
9 Initialization: $\forall e \in E^r$, let $w(e) = 1$; $M = E^r$; /* w is edge weight */
10 while (M contains a cycle C)
11 pick every other edge in C to form 2 matchings M_1, M_2 such that $w(M_1) \ge w(M_2)$;
12 $\forall e \in M_1, w(e) + +$; $\forall e \in M_2, w(e) - -$;
13 $M = \{e | e \in E^r \;\&\&\; w(e) > 0\}$;
14 return M;

Fig. 4. Find $\Delta(G^r)$ perfect matchings that form G^r.

Wavelength_Adjustment($G'_o = (V, E'_o), G'_n = (V, E'_n)$):
/* G'_o is multigraph representation of $G = (V, E, \phi_o)$, and G'_n is multigraph
representation of $G = (V, E, \phi_n)$ */
1 let $\lambda_o = \{m_1, m_2, \cdots, m_\Delta\}$; /* m_i (color c_i) is a matching in G'_o */
2 let $\lambda_n = \{\}$; $d = \Delta$; $V_{max} = \{v \mid \text{degree}(v) = d\}$;
3 foreach $m_i \in \lambda_o$: /* use λ_o to assist the decomposition of G'_n */
4 $m = \{e \mid e \in m_i \;\&\&\; e \in E'_n\}$;
5 if (m covers all nodes in V_{max})
6 $\lambda_n = \lambda_n \cup \{m\}$; $E'_n = E'_n \setminus m$; $d - -$;
7 $d = d - 1$; $V_{max} = \{v \mid \text{degree}(v) = d\}$;
8 else
9 $V' = \{v \mid v \in V_{max} \;\&\&\; v$ is not associated with $m\}$;
10 if ($\exists$ a matching $m' \subseteq E'_n$ covers V' && $m \cap m' = \emptyset$)
11 $m = m \cup m'$; /* a matching in G'_n that covers V_{max}; */
12 $\lambda_n = \lambda_n \cup \{m\}$; $E'_n = E'_n \setminus m$;
13 $d = d - 1$; $V_{max} = \{v \mid \text{degree}(v) = d\}$;
14 if ($d > 0$) /* not all Δ matchings are found from above */
15 find the rest d matchings using Figure 4 algorithm and put them into λ_n; /* suppose now $\lambda_n = \{m'_1, m'_2, \cdots, m'_\Delta\}$ */
16 return *Color_Assignment*(λ_o, λ_n);

Color_Assignment(λ_o, λ_n): /* given λ_o, find a color assignment to λ_n to maximize the common colors on common edges between λ_o and λ_n */
17 $\lambda_o = \{m_1, m_2, \cdots, m_\Delta\}$; /* m_i has color c_i */
18 $\lambda_n = \{m'_1, m'_2, \cdots, m'_\Delta\}$;
19 build cost matrix $C_{ij} = \{c_{ij} | c_{ij} = |m_i \cap m'_j|\}$;
20 let $X_{ij} = \{x_{ij} | x_{ij} = \{0, 1\}, \sum_i x_{ij} = 1, \sum_j x_{ij} = 1\}$;
21 maximize CX using Hungarian [3] algorithm;
22 return X_{ij}; /* $x_{ij} = 1$ means assigning c_i (color of m_i) to m'_j */

Fig. 5. A heuristic algorithm of MWA problem.

decomposition of new multigraph G'_n into Δ matchings λ_n=$\{m'_1, m'_2, \cdots, m'_\Delta\}$, and then assign colors to λ_n to maximize overlap between λ_n and λ_o (Hungarian [3]).

Specifically, in lines 3-13, using each of the old matchings m_i as a reference, we try to find a new matching m in G'_n that has as many overlap edges with m_i as possible. It is worthwhile to note that in lines 5 and 11, we require that the new matching found must cover all the maximum degree nodes in the current graph. This is a sufficient condition to guarantee that G'_n can be decomposed into Δ matchings finally. Because with this requirement, after successfully finding i matchings, the residual graph has maximum node degree $(\Delta - i)$ and thus can be decomposed into $(\Delta - i)$ matchings. If lines 3-13 cannot find all the Δ matchings of G'_n, in 14-15, we proceed to use ordinary method in Figure 4 to find the remaining matchings. Finally, in lines 16-22, we use Hungarian algorithm to assign colors to λ_n with the goal to maximize the color overlap between λ_n and λ_o. We note that our algorithm is not optimal and there is room to improve. However, it runs quickly and provides impressive gains as shown in Section V-E.

V. Performance Evaluation

In this section, we evaluate WaveCube via large-scale simulations. We first introduce the evaluation methodology, and then present the results.

A. Methodology

Topology: Our simulation is mainly based on a $(6, 6, 6)$-radix WaveCube topology. It has 3 dimensions and each dimension has 6 ToRs. We assume each ToR has 80 10G ports: half of them connect to 40 hosts with 10G NICs and the other half connect to 6 other ToRs via optics. This topology has a total number of 8640 hosts. Further, we assume each port of ToR that connects to the optics is equipped with an optical transceiver with a unique wavelength that carries 10G bandwidth. The number of wavelengths on a specific ToR link varies from 1 to 40, suggesting a variance from 10G to 400G.

Traffic Patterns: We use the following traffic patterns.

- **Realistic:** We collect real traffic matrix (TM) from a production data center[5] with ∼400 servers with 1G ports. The data center runs Map-reduce style applications, with widespread communication patterns. For example, the average server fan-in/out degrees are $45/43$ and the maximal are $169/164$. To replay the traffic over 8640 servers with 10G ports, we proportionally reduce the transmission time by 10X (from 1G to 10G) and replicate TMs spatially (assuming more concurrent Map-reduce applications). Specifically, we partition the entire servers into over 20 groups and replay the TM in each group.
- **Microsoft-based:** We synthesize traffic patterns based on measurement results from recent works [12], [25], [28] by Microsoft. These papers describe the traffic characteristics in real data centers. For example, they found that hotspots are often associated with a high fan-in (fan-out) manner [25], and most of the traffic (80%) are within the rack [12]. We capture the hotspot characteristics and assume all traffic exit the rack to create intensive communications.
- **Random:** We assume each server in a ToR talks to servers in up to 15 randomly selected ToRs. In this pattern, many ToRs can simultaneously talk to one ToR, creating hotspots and communication bottlenecks.

[5]The name of the production data center is anonymized for privacy.

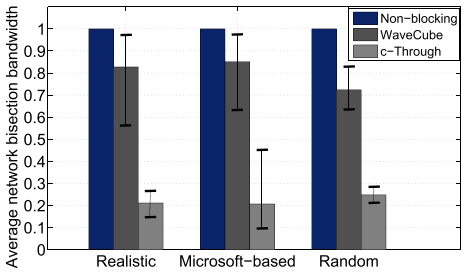


Fig. 6. Network bisection bandwidth.

Evaluation Metrics: We extensively evaluate WaveCube from the following aspects. First, we measure the network bisection bandwidth of WaveCube under the above traffic patterns. Second, we quantify the benefit of dynamic link bandwidth in improving network performance. Third, we check the fault-tolerance. Fourth, we quantify the effect of wavelength adjustment optimization in avoiding unnecessary wavelength shifting. Fifth, we analyze the control overhead of WaveCube. Finally, we discuss the effect of traffic stability on WaveCube.

Simulator: We implement our own simulator because there is no standard one for our purpose. The simulator we developed models WaveCube as a directed graph with alterable edge weights. It takes as input the flows with sizes, start time, source and destination hosts. The simulation runs in discrete time ticks with the granularity of millisecond. On each tick, the rate of each flow is updated by running on all active flows the progressive filling algorithm [6], which produces a bandwidth allocation satisfying max-min fairness, and is known as a good estimation of TCP behaviors. The sent bytes are subtracted after each tick and completed flows are removed. The simulator calls the bandwidth scheduler to reschedule link bandwidth periodically.

B. Achieved Network Bisection Bandwidth

Figure 6 shows the average (max/min) network bisection bandwidth achieved by WaveCube when running 40 instances of each of the above traffic patterns on the simulated WaveCube with 8640 hosts. The results are specifically compared against c-Through (Helios performs similarly as c-Through, OSA/Mordia perform better but they are not scalable.)[6] and a hypothetical non-blocking network, which serves as the upper-bound of performance for any DCN.

[6]We note that Helios has the same idea as c-Through as it uses both optical and electrical switches for interconnection, except that it is designed for inter-modular-DCNs with one or more MEMSes for inter-Pods. Helios will perform similar as c-Through when it employs a MEMS for inter-ToRs inside a single DCN. We expect OSA to perform better than c-Through/Helios, however it is designed for container-size DCN with 2560 servers, and cannot scale to 8640 servers at its current design. Mordia has even more severe scalability issue than OSA. Due to these concerns, we do not compare with Helios, OSA, or Mordia directly in this paper.

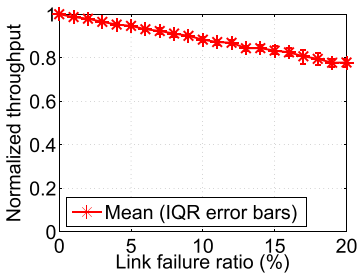


Fig. 7. WaveCube under failures.

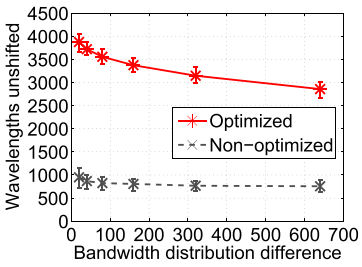


Fig. 8. Quality of wavelength adjustment (mean, IQR).

From the figure, we find that, in terms of network bisection bandwidth, WaveCube outperforms c-Through by 300%-400% and is 70%-85% of non-blocking under all traffic patterns. This is not a surprising result. Because c-Though assumes an one-hop pairwise circuits in its optical part, such interconnect is of marginal use to offload the traffic when hotspots are associated with high fan-in (fan-out). In contrast, our $(6, 6, 6)$-radix WaveCube uses multi-hop routing in a fixed 6-regular topology, and any pair of ToRs has 6 node-disjoint parallel paths. Despite a fix topology, WaveCube demonstrates competitive performance via its rich path diversity. Furthermore, WaveCube dynamically adjusts its link bandwidth to fit the underlying traffic, further improving its performance (see Section V-C). In summary, our results suggest that multi-pathing and dynamic link bandwidth are effective to offload the hotspots, and deliver high bisection bandwidth for both realistic and synthetic traffic patterns.

C. Benefit of Dynamic Link Bandwidth

To evaluate the benefit brought by our link bandwidth optimization, we assume a *static* network where the 40 wavelengths of each node in the $(6, 6, 6)$-radix topology are evenly and statically distributed along all dimensions. Figure 9 shows the network bisection bandwidth versus time for WaveCube against the static network under different traffic patterns.

From the three figures, we find that, with dynamic link bandwidth optimization, the performance can be increased by 10%-40%. This indicates that while the path diversity in WaveCube balances the traffic loads among the network, the uncoordinated flow scheduling for different source-destination ToR pairs causes congestion on some links. WaveCube can adaptively change link bandwidth to handle such congestion, leading to improved performance.

D. Performance under Failures

To show fault-tolerance, we check the aggregate throughput of WaveCube under failures. In our experiment, we generate the node/link failures in the network randomly, and we regard a node failure as a combination of link failures incident to this node. We run with the realistic traffic pattern and show the result in Figure 7. The throughput is normalized by the non-failure case.

In the figure, we see a graceful performance degradation with increased failures. For example, with as many as 20% links down, the network aggregate throughput is decreased by 20%. This result is expected because WaveCube structure is fault-tolerant. It has $2n$ node-disjoint parallel paths between any pair of ToRs. Once failures happen, the traffic can be easily routed away from the failed parts using other parallel paths. Furthermore, WaveCube has flexible link bandwidth. In case a link fails, the associated nodes can reschedule the bandwidth of the failed link to other links so that the resources can be potentially reused elsewhere.

E. Quality of Wavelength Adjustment

We evaluate the quality of our wavelength adjustment algorithm in Figure 5 on the $(6, 6, 6)$-radix WaveCube with each ToR having 40 wavelengths. We first select a base network state (S_1) with initial bandwidth demand (ϕ_1) and wavelength distribution (λ_1). Then, we generate another network state (S_2) with a new bandwidth demand (ϕ_2) by randomly rearranging bandwidth requirement on its links. We run *Wavelength_Adjustment()* to get the new wavelength distribution (λ_2) and check how our algorithm can keep its original distribution unmodified (*i.e.*, $|\lambda_2 \cap \lambda_1|$) during the state transition ($S_1 \rightarrow S_2$). We compare our algorithm with the one without optimization. We repeat 100 times for each experiment and compute the mean and IQR, *i.e.*, $25th$-$75th$ percentiles.

Results in Figure 8 suggest that our algorithm effectively avoids unnecessary wavelength shifting. For example, when the bandwidth demand difference (*i.e.*, $|\phi_1 - \phi_2|$) is 20, our algorithm maintains $3876/4320$=90% wavelengths unshifted, while a non-optimized method only keeps $949/4320$=22% wavelengths unchanged. There is a decreasing trend on the curves. This is because larger bandwidth demand change is likely to cause bigger wavelength shifting. We have not been able to compare our algorithm with the optimal solution due to its complexity. But we are able to make an estimation. For example, when $|\phi_1 - \phi_2| = 640$, the optimal solution can at most keep $4320 - 640 = 3680$ wavelengths unshifted (should be less than 3680). As a comparison, our algorithm can keep 3000 wavelengths unshifted, over 80% of the optimal.

F. Overhead of the Central Controller

The central controller handles most of intelligences in control plane. It needs to maintain the connectivity, utilization and wavelength distribution information for each ToR link. The connectivity is used for detecting failures, utilization for link bandwidth optimization, and wavelength distribution for wavelength adjustment optimization, respectively. Required states

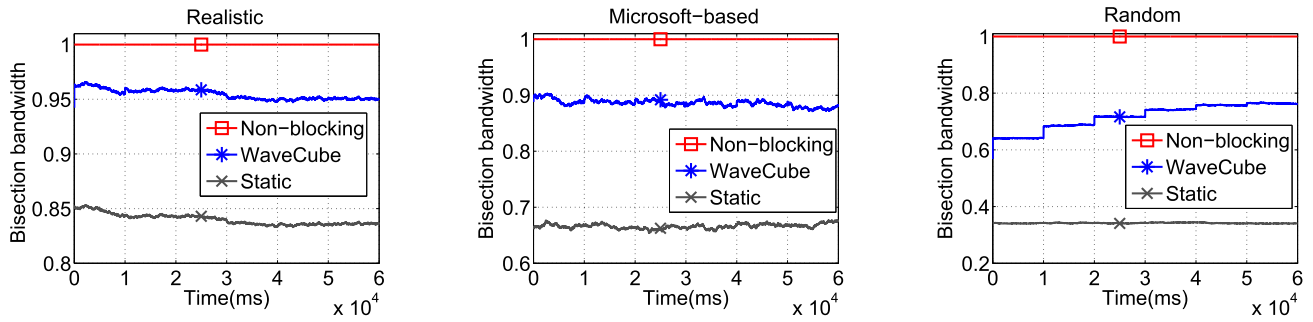


Fig. 9. Benefit of dynamic link bandwidth optimization in network performance improvement.

TABLE III
TIME COST OF LINK BANDWIDTH SCHEDULING (LBS) AND WAVELENGTH ADJUSTMENT OPTIMIZATION (WAO)

WaveCube	#Hosts	LBS Time(ms)	WAO Time(ms)
$(6,6,6)$-radix	8,640	1.9	2
$(8,8,8)$-radix	20,480	4.5	7
$(10,10,10)$-radix	40,000	9.3	18
$(12,12,12)$-radix	69,120	16.6	48

for these information is $O(m)$ where m is the number of ToR links in the network. This is modest considering Lemma 1, which is $O(10^5)$ even for mega data centers.

We next evaluate time cost of two main algorithms executed by the central controller: link bandwidth scheduling (Section III-D) and wavelength adjustment optimization (Section IV-B). Table III shows the results for both optimization algorithms versus network sizes. The result for LBS suggests that the optimization can be finished quickly. For example, it takes 16.6ms for a large WaveCube with 69,120 hosts. Further, we find that the runtime increases with the network size incrementally. The result for WAO suggests that our algorithm is time-efficient as it just spends tens of milliseconds for the 69,120-host WaveCube. The fast algorithms are essential to make WaveCube react to new traffic patterns promptly.

G. Effect of Traffic Stability

WaveCube performs well due to its multi-pathing and dynamic link bandwidth. Among these two, the gain of dynamic link bandwidth should assume certain traffic stability. The analysis on our real traffic matrices shows over 60% traffic stablity at minutes or even hourly timescale [40]. Another study [13] found that 60% of ToR-pairs see less than 20% change in demand for seconds. Further, recent work [25] used 300s to compute the demands from their traces and found that the present traffic demand can be well predicted from the past ones. All of these studies give us confidence that WaveCube's dynamic link bandwidth can take effect on a variety of practical workloads.

However, we envision that if the traffic is highly dynamic [27], WaveCube's performance can be significantly affected due to its 10ms switching delay. To quantify this, we vary the traffic stability period t from 1ms to 1000ms for two synthetic traffic patterns used in Helios [18], *i.e.*, Node (or Rack) Stride and Host Stride. Basically, denote

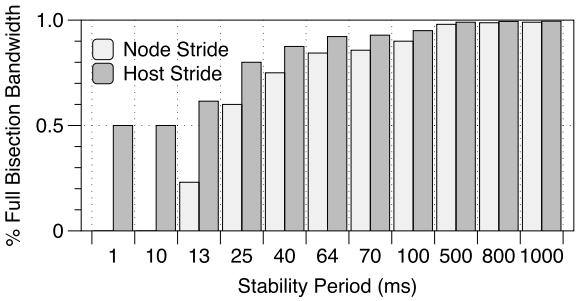


Fig. 10. Throughput vs traffic stability (Node=Rack, Host=Server).

n as the number of racks in the entire topology and k the number of hosts per rack. The traffic patterns are described as follows.

- **Node Stride:** Numbering the racks from 0 to $n-1$. For the i-th node, its j-th host initiates a TCP flow to the j-th host in the $(i+l \mod n)$-th node, where l rotates from 1 to n every t ms (t is the traffic stability period). This pattern tests the response to abrupt demand changes between nodes, as the traffic from one rack completely shifts to another rack in a new period.
- **Host Stride:** Numbering the hosts from 0 to $n \times k-1$, the i-th host sends a TCP flow to the $(i+k+l \mod (n \times k))$-th host, where l rotates from 1 to $\lceil k/2 \rceil$ every t ms. This pattern showcases the gradual demand shift between nodes.

Figure 10 shows the average throughput vs traffic stability. We make two observations: 1) If stability period is less than 10ms (*i.e.*, the reconfiguration delay), Node Stride achieves almost 0 throughput while Host Stride achieves 50% of the full bisection bandwidth. This is because for the Node Stride case, all the wavelengths need to be reconfigured each time; while for the Host Stride case, 50% of the wavelengths stay unchanged; 2) Both patterns see the throughput increasing to full bisection bandwidth with longer stability period.

From the above result, we find it is detrimental to apply WaveCube for dynamic traffic whose stability period is less than the reconfiguration delay. One alternative way to handle such case is to disable dynamic link bandwidth. Then, the performance of WaveCube is degraded to be a static network with multi-pathing. Actually, the bottom curves in Figure 9 show the performance without dynamic link bandwidth. Though there is performance degradation, it would not completely disable WaveCube.

VI. Practical Deployment Analysis

We discuss the practical deployment issues like construction and expansion of WaveCube, and compare it with previous DCNs. Given a large body of recent designs, we select Fattree [32] (switch-centric), BCube [22] (server-centric) and c-Through [39] (optical) as representatives. Then, we analyze the cost and power. Finally, we evaluate the hardware feasibility of WaveCube by comparing it with OSA [15].

A. Wiring Complexity

We observe that some DCN designs [21]–[23], [32], while providing good performance, are hard to construct in practice due to their dense connections and strict wiring rules. For example, many wires must be connected between specific devices or via specific ports. How to build a data center is a practical question, especially when the data center is large. Comparing to all recently proposed DCNs, WaveCube is perhaps the easiest-to-build one in terms of wiring.

Directly counting the number of wires as previous work did [22] is not a very sound way to quantify the wiring complexity. Not all the wires have the same difficulty to set up in practice. To analyze and compare the complexity of wiring in DCNs, we develop the following metrics. First, we categorize wires into *rule-based* ones and *rule-free* ones. The rule-free wire refers to the wire that if one end is fixed, the other end can be selected from a set of devices or ports unconditionally. Usually, wires from servers to ToR on the same rack are rule-free and very easy to connect. In contrast, a rule-based wire requires that once one end is fixed, the other end should be connected to a specific device/port or one of several devices/ports conditionally. Such rule-based wiring is usually error-prone and needs special care. Hence, the real complexity of wiring mainly comes from the rule-based wires.

To quantify the complexity, we first need to understand the placement of devices in data centers. Usually, devices are arranged in racks, and racks are organized in rows and columns [8]. Thus, we assign each rack a coordinate (i, j), i is the index of column and j is the index of row. Suppose two devices a and b are in different racks (i_a, j_a) and (i_b, j_b), we use $d(a, b) = |i_a - i_b| + |j_a - j_b|$ (Manhattan distance [5]) to estimate the wiring length between these two devices. This is because a practical and structured approach is to place wires in rows or columns along the racks, which facilitates cable identification, trouble-shooting and planning for future changes [1].

We assume two devices in the same rack have length $d(a, b) = 1$. We use $r(a, b)$ to denote whether a wire is rule-free (*i.e.*, $r(a, b) = 0$) or rule-based (*i.e.*, $r(a, b) = 1$). Then, we summarize the wiring complexity of a data center as:

$$C\_index = \sum_{\forall a,b \in V} r(a, b) \times d(a, b) \tag{5}$$

With the above metric, we compare WaveCube with Fattree, BCube and c-Through. We target at both container-size (3000-5000 servers) and large-scale (25000-40000 servers). Especially, in order to make the comparison more accurate, we compute the complexity of each structure according to its specific structure characteristics individually. We put racks in rows and columns, and place servers and switches in a way that the length of rule-based wires is optimized.

TABLE IV
Wiring Complexity of DCNs[4]

Scale	DCN	#Hosts	C_index
Container-size	Fattree	3,456	25,728
	BCube	4,096	48,672
	c-Through	4,000	1,475
	WaveCube	5,760	1,104
Large-scale	Fattree	27,648	427,160
	BCube	32,768	1,228,064
	c-Through	36,000	35,775
	WaveCube	36,864	6,272

The results are shown in Table IV.[7] It can be seen that WaveCube is 2-3 orders of magnitude simpler than Fattree/BCube and 1 order of magnitude simpler than c-Through at scale. When the size grows, the complexity of WaveCube grows much slower than the others. The advantages of WaveCube come from two main reasons: a) A single optical fiber can aggregate high data volume which otherwise need to be carried by many copper cables; b) Most of wires in WaveCube are local except the loop-back ones, while many wires in Fattree and BCube are (and have to be) between remote racks. In c-Through, all ToRs connect to the central MEMS, introducing remote wiring.

B. Data Center Expansion

With the growth of applications and storage, the scale of a DCN will not remain the same for long [14]. Thus, it is desirable that a structure should support expansion. In WaveCube, the expansion can be easily achieved via adding ToR switches in one dimension. For example, a $(6, 6, 6)$-radix WaveCube can be extended to $(6, 6, 8)$-radix WaveCube via adding 72 ToRs and 2840 servers. In addition, since most WaveCube wires are local, it is relatively easy to "insert" new ToRs in one dimension locally without causing re-wiring elsewhere. Similarly, expanding c-Through can be done through adding more ToRs and connecting them to the MEMS with possibly long wires, and thus is not hard either.

On the contrary, expansion in BCube or Fattree (or related DCNs) is not easy. Regardless of many long wires, the primary reason is that the number of servers in the networks is decided by the switch port density. For example, Fattree-1 hosting 16000 servers can be built with 40-port switches, and Fattree-2 hosting 27648 servers can be built with 48-port switches. To expand from Fattree-1 to Fattree-2, two possible ways may exist: 1. Initially build Fattree-1 with 48-port switches and reserve 27648-16000=11648 ports for future expansion, or 2. Initially build Fattree-1 with 40-port switches and replace them with 48-port switches when expansion is required. Apparently, both approaches are inflexible.

[7]Note that there are many other DCNs like VL2 [21], DCell [23], Ficonn [30], BCN [24], Helios [18], OSA [15], etc, not specifically listed here. Basically, VL2 is similar as Fattree and both are Clos network. DCell, Ficonn and BCN are all server-centric as BCube, they have extremely complex wiring rules making them hard to build. Helios and OSA are similar, if not harder than c-Through since they introduce denser optical interconnections.

TABLE V

COST AND POWER FOR DIFFERENT DEVICES (†PER PORT VALUE), SOME VALUES ARE REFERRED FROM HELIOS [18]

Device	Cost($)	Power(W)	Device	Cost($)	Power(W)
ToR (10G)	500†	12.5†	(DE)MUX	3000	0
MEMS	500†	0.24†	Coupler	100	0
WSS	1000†	1†	Circulator	200	0
Transceiver	800	3.5	-	-	-

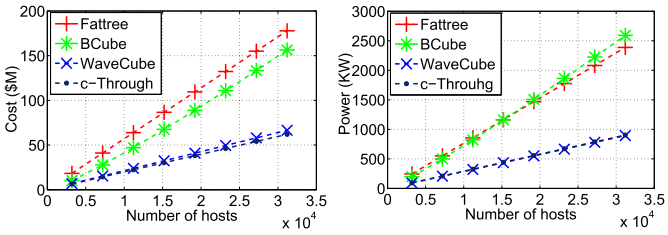


Fig. 11. Cost and power for different DCNs.

C. Cost and Power Consumption

We estimate the cost and power consumption for different DCNs based on the values listed in Table V. Note that in 10G electrical networks, optical transceivers are required for over 10m links [18]. So long-distance, cross-rack links in Fattree or BCube must be optical. For example, Google's DCN uses optical fibers over fattree interconnect fabric [7].

Figure 11 shows the results. It is evident that, to host the same number of servers, WaveCube is significantly cheaper than either Fattree ($\sim$35%) or BCube ($\sim$40%) and consumes much less power ($\sim$35% for both). This is counter-intuitive since it is common-sense that optical devices such as WSS and MEMS are expensive. However, a detailed check reveals that the real dominance is optical transceivers. Both Fattree and BCube have much higher switch port density, and so their cost is higher. For example, Fattree uses $5k^3/4$ ports to connect $k^3/4$ servers (where k is the number of ports on a Fattree switch), while BCube uses $(l+1)k^{l+1}$ ports to connect k^{l+1} servers (where k is the number of ports on a BCube switch, and l is the level of BCube).

In contrast, WaveCube only has ToR switches, and c-Through has a few more electrical switches in addition to ToRs. Both have lower switch port density, leading to lower cost. Due to the same reason, the power cost of WaveCube and c-Through is lower than Fattree and BCube. The same trend applies to other related DCNs: electrical ones are more expensive and consume more power than optical ones at the era of 10G. Finally, we observe that WaveCube is slightly costly than c-Through. This is because WaveCube employs a few more optical devices such as circulator and coupler, and WSS is more expensive than MEMS.

D. Usage of Optical Devices

In Table VI, we compare WaveCube with other optical structures in terms of the main optical devices (e.g., WSS, MEMS optical switch, MUX/DEMUX, and transceiver) used at the same scale of 8640 hosts (= 216 racks $\times$ 40 hosts/rack). We just follow the construction rule in each paper. As shown in the table, Helios/c-Through do not need WSS but a 216-port MEMS to interconnect 216 racks. OSA almost has the same devices as WaveCube, except that it requires one additional 1296-port MEMS to connect all 216 racks (each rack needs 6 MEMS ports). By default, Mordia only supports limited port count, e.g., 88 [34], on a Mordia ring (due to wavelength contention). We scale it out to 8640 by stacking 98 rings together using a 98$\times$98 MEMS optical switch as suggested by [34]. Furthermore, Mordia only needs MUX but not DEMUX. Finally, all the structures require the same number of optical transceivers.

TABLE VI

COMPARISON OF OPTICAL DEVICES AT THE SAME SCALE (8640 HOSTS)

Devices	Helios/ c-Through	OSA	Mordia	Wavecube
WSS	0	216 (1×6)	2160 (1×4)	216 (1×6)
MEMS	1 (216 ports)	1 (1296 ports)	1 (196 ports)	0
MUX/DEMUX	216/216	216/216	2160/0	216/216
Transceivers	8640	8640	8640	8640

However, it is important to note that, while at the same scale, different structures may deliver different network bisection bandwidth. For example, Helios/c-Through only support rack-to-rack pairwise connection at a time [18], [39]. Mordia (stacked rings) is blocking, meaning that not any two ports across different rings are connected [34]. In contrast, WaveCube/OSA provide all-to-all connectivity at any time via multi-hop routing. Furthermore, the WSSes used in different structures are also different, e.g., the WSS used in Mordia is microsecond level (11.5μs, expensive) switching [34], whereas in WaveCube the WSS is millisecond level ($\sim$10ms, cheaper) switching.

E. WaveCube Hardware Feasibility

The crux of showing the feasibility of WaveCube is to demonstrate the feasibility of the optical component in Figure 1. This part is similar to that of OSA without introducing any new advanced optical devices. To this end, instead of building a dedicated small WaveCube testbed, we leverage the OSA testbed to discuss the hardware feasibility of WaveCube. (Readers can refer to [15] for details of the OSA testbed.)

In the OSA testbed, instead of direct connection, one Polatis series-1000 OSM/MEMS with 32 ports (16$\times$16) was used to connect 8 PC-emulated ToRs through WSS units. The key difference between WaveCube and OSA is that WaveCube removes MEMS and directly connects ToRs in a k-ary-n-cube topology. As we have shown through analysis and extensive simulations, this seemingly simple architecture re-design has translated to significant benefits in scalability, fault-tolerance, as well as the optimal wavelength assignment.

WaveCube can be built by fixating the MEMS circuits and simply treating them as dumb fibers. Therefore, the extensive feasibility study in OSA paper [15], as a reference, has also demonstrated the feasibility of the optical component of a small-scale WaveCube. However, we note that even with such a small-scale testbed, it is far from sufficient to conduct performance evaluation for WaveCube, whose target is scalable optical DCN. Thus, we have focused on evaluating WaveCube

in large-scale simulated settings as elaborated in last section, and leave the implementation of a larger non-trivial WaveCube prototype as our next step.

VII. Related Work

There are a large spectrum of works in data center networks and optical networks that are related to WaveCube. Due to space limitation, we only focus on the closely related ones.

Helios [18], c-Through [39], OSA [15], and Mordia [34] are the most related works to WaveCube since they all explore optical technologies in DCNs. However, WaveCube differs from them in both design goals and methodology. Generally speaking, WaveCube seeks a scalable, fault-tolerant, high-performance optical DCN architecture, while none of these three proposals achieves the three goals simultaneously.

c-Through and Helios present hybrid optical/electrical DCNs. Their basic idea is to use MEMS-based optical switch to provide high-capacity one-hop, pairwise connections between ToRs, and use traditional electrical interconnects (possibly oversubscribed) for overall connections among ToRs for bursty traffic. Helios targets on inter-Pod connections where each Pod is a modular DCN with thousands of servers, while c-Through focuses on inter-ToR connections inside a single DCN. A common feature of c-Through and Helios is that their optical connection is of low fan-in (fan-out) and non-transitive. This leads to reduced performance when the hotspots occur with a high fan-in (fan-out) manner which are prevalent in practical data center workloads [25].

In an earlier paper [15], OSA introduces a topology malleable all-optical DCN. It allows multi-hop single-path routing via optical links on a reconfigurable K-regular topology with MEMS and also enables link capacities to be changed on-demand. The highlight of OSA is its unprecedented flexibility. It overcomes the deficiency of single-hop non-transitive connection of c-Through or Helios. However, OSA is intended for container-size DCNs and is hard to scale. Furthermore, the wavelength assignment problem remains unsolved. Mordia [34] also cannot scale due to wavelength contention.

VIII. Conclusion

We have presented WaveCube, a scalable, fault-tolerant, high-performance optical DCN architecture. WaveCube removes MEMS from its design, thus achieving scalability. It is fault-tolerant as there is no single point of failure and node-disjoint parallel paths exist between any pair of ToRs. WaveCube delivers high performance by exploiting multi-pathing and dynamic link bandwidth. Our results show that WaveCube well outperforms previous optical DCNs and delivers network bisection bandwidth that is 70%-85% of non-blocking under both realistic and synthetic traffic patterns. And its performance degrades gracefully in case of failures—a 20% drop even with 20% links cut.

Appendix

A. Correctness of Find_Perfect_Matching()

Find_Perfect_Matching() is what we learned and summarized from previous work and is not our contribution. We prove it for reader's better understanding of the algorithm.

Theorem 3: The procedure Find_Perfect_Matching() returns a perfect matching M of G^r in $O(\Delta(G^r)|E^r|)$ time.

Proof: We first show M is a perfect matching. The proof is based on the fact that the sum of weight w of all edges incident to each node is $\Delta(G^r)$ in the initialization, and is never changed during the whole algorithm execution. This is guaranteed by line 12 of Figure 4. When the algorithm terminates, the subgraph of G^r whose edge set is M has no cycle and is thus a forest. Suppose u is a leaf node and $e = (u, v)$, then $w(e) = \Delta(G^r)$, and due to the above fact, v must have no other incident edge in M except e, which indicates that M is a matching. Furthermore, since M covers all the nodes in G^r, M is a perfect matching of G^r.

Regarding the time complexity, the value $\sum_{e\in E^r} w(e)^2$ is $|E^r|$ in the initialization, and is $\frac{|V|}{2}\Delta(G^r)^2 = \Delta(G^r)|E^r|$ when algorithm terminates. In each iteration (lines 10-13), the value $\sum_{e\in E^r} w(e)^2$ is increased by:

$$\begin{aligned}
&\sum_{e\in M_1}((w(e)+1)^2 - w(e)^2) \\
&\qquad + \sum_{e\in M_2}((w(e)-1)^2 - w(e)^2) \\
&= \ 2w(M_1) + |M_1| - 2w(M_2) + |M_2| \\
&\geq \ |M_1| + |M_2| = |C| \qquad (6)
\end{aligned}$$

The above inequation is due to $w(M_1) \geq w(M_2)$. Furthermore, we need $O(|C|)$ time to find a cycle C. Thus, the entire algorithm runs in $O(\Delta(G^r)|E^r|)$ time. ∎

References

[1] *Best Practices Guide: Cabling the Data Center*. [Online]. Available: http://www.brocade.com/downloads/documents/white_papers/cabling_best_practices_ga-bp-036-02.pdf

[2] *Edge Coloring*. [Online]. Available: http://en.wikipedia.org/wiki/edge_coloring

[3] *Hungarian Algorithm*. [Online]. Available: http://en.wikipedia.org/wiki/hungarian_algorithm

[4] *Lee Distance*. [Online]. Available: http://en.wikipedia.org/wiki/lee_distance

[5] *Manhattan Distance*. [Online]. Available: http://en.wiktionary.org/wiki/manhattan_distance

[6] *Progressive Filling*. [Online]. Available: http://en.wikipedia.org/wiki/max-min_fairness

[7] *Scaling Challenges for Warehouse Scale Computers*. [Online]. Available: http://iee.ucsb.edu/content/2011-summit-video-presentation-bikash-koley

[8] D. Abts and J. Kim, "High performance datacenter networks: Architectures, algorithms, and opportunity," *Synthesis Lectures Comput. Archit.*, vol. 6, no. 1, p. 115, Mar. 2011.

[9] H. Abu-Libdeh, P. Costa, A. Rowstron, G. O'Shea, and A. Donnelly, "Symbiotic routing in future data centers," in *Proc. SIGCOMM*, Aug. 2010, pp. 51–62.

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

[11] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat, "Hedera: Dynamic flow scheduling for data center networks," in *Proc. USENIX Symp. Netw. Syst. Design Implement. (NSDI)*, Apr. 2010, p. 19.

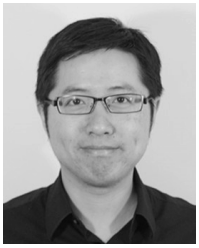
[12] T. Benson, A. Akella, and D. Maltz, "Network traffic characteristics of data centers in the wild," in *Proc. ACM IMC*, Nov. 2010, pp. 267–280.

[13] T. Benson, A. Anand, A. Akella, and M. Zhang, "The case for fine-grained traffic engineering in data centers," in *Proc. USENIX INM/WREN*, Apr. 2010.

[14] K. Chen *et al.*, "Generic and automatic address configuration for data centers," in *Proc. SIGCOMM*, Aug. 2010, pp. 39–50.

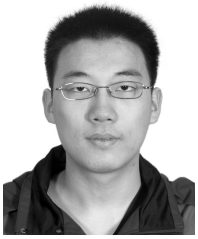
[15] K. Chen *et al.*, "Osa: An optical switching architecture for data center networks with unprecedented flexibility," in *Proc. USENIX Symp. Netw. Syst. Design Implement. (NSDI)*, 2012, pp. 239–252.

[16] W. J. Dally, "Performance analysis of k-ary n-cube interconnection networks," *IEEE Trans. Comput.*, vol. 39, no. 6, pp. 775–785, Jun. 1990.
[17] J. Dean and S. Ghemawat, "Mapreduce: Simplified data processing on large clusters," in *Proc. USENIX Symp. Oper. Syst. Design Implement. (OSDI)*, 2004, pp. 137–150.
[18] N. Farrington *et al.*, "Helios: A hybrid electrical/optical switch architecture for modular data centers," in *Proc. SIGCOMM*, Aug. 2010, pp. 339–350.
[19] B. Fortz and M. Thorup, "Internet traffic engineering by optimizing ospf weights," in *Proc. INFOCOM*, Mar. 2000, pp. 519–528.
[20] S. Ghemawat, H. Gobioff, and S. T. Leung, "The Google file system," in *Proc. SIGOPS*, Oct. 2003, pp. 29–43.
[21] A. Greenberg *et al.*, "VL2: A scalable and flexible data center network," in *Proc. ACM SIGCOMM*, Aug. 2009, pp. 51–62.
[22] C. Guo *et al.*, "BCube: A high performance, server-centric network architecture for modular data centers," in *Proc. SIGCOMM*, Aug. 2009, pp. 63–74.
[23] C. Guo *et al.*, "DCell: A scalable and fault-tolerant network structure for data centers," in *Proc. SIGCOMM*, Aug. 2008, pp. 75–86.
[24] D. Guo *et al.*, "Bcn: Expansible network structures for data centers using hierarchical compound graphs," in *Proc. INFOCOM*, Apr. 2011, pp. 61–65.
[25] D. Halperin, S. Kandula, J. Padhye, P. Bahl, and D. Wetherall, "Augmenting data center networks with multi-gigabit wireless links," in *Proc. ACM SIGCOMM*, Aug. 2011, pp. 38–49.
[26] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly, "Dryad: Distributed data-parallel programs from sequential building blocks," in *Proc. Eurosys*, Mar. 2007, pp. 59–72.
[27] S. Kandula, J. Padhye, and P. Bahl, "Flyways to de-congest data center networks," in *Proc. HotNets*, Aug. 2009.
[28] S. Kandula, S. Sengupta, A. Greenberg, P. Patel, and R. Chaiken, "The nature of datacenter traffic: Measurements and analysis," in *Proc. ACM IMC*, Nov. 2009, pp. 202–208.
[29] C. F. Lam *et al.*, "Fiber optic communication technologies: What's needed for datacenter network operations," *IEEE Commun. Mag.*, vol. 48, no. 7, pp. 32–39, Jul. 2010.
[30] D. Li *et al.*, "FiConn: Using backup port for server interconnection in data centers," in *Proc. INFOCOM*, Apr. 2009, pp. 2276–2285.
[31] H. Liu, C. F. Lam, and C. Johnson, "Scaling optical interconnects in datacenter networks opportunities and challenges for wdm," in *Proc. HOTI*, Aug. 2010, pp. 113–116.
[32] R. N. Mysore *et al.*, "Portland: A scalable fault-tolerant layer 2 data center network fabric," in *Proc. SIGCOMM*, Aug. 2009, pp. 39–50.
[33] A. E. Ozdaglar and D. P. Bertseka, "Routing and wavelength assignment in optical networks," *IEEE/ACM Trans. Netw.*, vol. 11, no. 2, pp. 259–272. Apr. 2003.
[34] G. Porter *et al.*, "Integrating microsecond circuit switching into the data center," in *Proc. SIGCOMM*, Sep. 2013, pp. 447–458.
[35] J. Rath. (May 24, 2010). *Google Eyes 'Optical Express' for Its Network*. [Online]. Available: http://www.datacenterknowledge.com/archives
[36] A. Schrijver, "Bipartite edge-colouring in $o(\delta m)$ time," *SIAM J. Comput.*, vol. 28, no. 6, pp. 841–846, 1998.
[37] A. Singla, C.-Y. Hong, L. Popa, and P. B. Godfrey, "Jellyfish: Networking data centers randomly," in *Proc. USENIX Symp. Netw. Syst. Design Implement. (NSDI)*, Apr. 2012, p. 17.
[38] T. Truex, A. A. Bent, and N. W. Hagood, "Beam steering optical switch fabric utilizing piezoelectric actuation technology," in *Proc. NFOEC*, 2003.
[39] G. Wang *et al.*, "c-Through: Part-time optics in data centers," in *Proc. SIGCOMM*, Aug. 2010, pp. 327–338.
[40] X. Wen *et al.*, "Virtualknotter: Online virtual machine shuffling for congestion resolving in virtualized datacenter," in *Proc. ICDCS*, Jun. 2012, pp. 12–21.



Kai Chen received the Ph.D. degree in computer science from Northwestern University, Evanston, IL, USA, in 2012. He is currently an Assistant Professor with the Department of Computer Science and Engineering, The Hong Kong University of Science and Technology, Hong Kong. His research interest includes networked systems design and implementation, data center networks, and cloud computing.

Xitao Wen received the B.S. degree in computer science from Peking University, Beijing, China, in 2010, and the Ph.D. degree in computer science from Northwestern University, Evanston, IL, USA, in 2016. His research interests include networking and security in networked systems, with a current focus on software-defined network security and datacenter networks.



Xingyu Ma received the B.E. degree in computer science from Tsinghua University, China, in 2012, and the M.S. degree in computer science from the University of California at Los Angeles, Los Angeles, CA, USA, in 2014. His interests include datacenter networking and mobile systems.

Yan Chen (F'17) received the Ph.D. degree in computer science from the University of California at Berkeley, Berkeley, CA, USA, in 2003. He is currently a Professor with the Department of Electrical Engineering and Computer Science, Northwestern University, Evanston, IL, USA. His research interests include network security and measurement and diagnosis for large-scale networks and distributed systems. He received the Department of Energy Early CAREER Award in 2005, the Department of Defense Young Investigator Award in 2007, and the Best Paper nomination in the ACM SIGCOMM 2010. Based on Google Scholar, his papers have been cited over 10 000 times and his h-index is 42.

Yong Xia (M'04–SM'12) received the B.E. degree from the Huazhong University of Science and Technology, Wuhan, China, in 1994, the M.E. degree from the Institute of Automation, Chinese Academy of Sciences, Beijing, in 1998, and the Ph.D. degree from Rensselaer Polytechnic Institute, Troy, NY, USA, in 2004. His research interests are in computer networking, mobile networking, and scalable data processing systems.

Chengchen Hu received the Ph.D. degree from Tsinghua University, Beijing, China, in 2008. From 2008 to 2010, he was an Assistant Research Professor with Tsinghua University. He became an Associate Professor with the Department of Computer Science and Technology, Xi'an Jiaotong University (XJTU), Xi'an, China. He visited Northwestern University, USA, for three months in 2007 and the Norwegian University of Science and Technology Norway, for one year in 2014. He is currently a Professor and the Head of the Department of Computer Science and Technology with XJTU.

He has authored over 70 papers in high competitive venues, including INFOCOM, CoNext, ICDCS, the IEEE/ACM TRANSACTIONS ON NETWORKING, the IEEE TRANSACTIONS ON COMMUNICATIONS, and the IEEE JOURNAL ON SELECTED AREAS IN COMMUNICATIONS. His main research interests include network measurement, cloud datacenter networking, and software defined networking. He served in the Organization Committee and Technical Program Committee of several conferences, including INFOCOM 2012–2017, IWQoS 2010, GLOBECOM 2010–2017, ICC 2011–2016, ANCS2 014–2015/2017, and Networking 2014–2017. He is a recipient of a Fellowship from the European Research Consortium for Informatics and Mathematics, Microsoft Star-Track Young Faculty Program, New Century Excellent Talents in University awarded by Ministry of Education, China.

Qunfeng Dong received the Ph.D. degree in computer science from UW-Madison. Since 1998, he has been devoting his career to research in technological fields centered on networking, computing, and data. He joined the University of Science and Technology of China as a Computer Science Professor, where he published the very first NSDI paper that has ever been accomplished by native universities of the Greater China area. In 2013, he joined Huawei as the Chief Scientist on computing architecture and the CTO on hardware acceleration, where he also served as a Founding Member of Huawei's Algorithm Committee. He is currently the Chairman and the CEO of DataBox Ltd., a leading vendor of chips and systems for data center and cloud computing, which he founded in 2015.

Yongqiang Liu received the B.E. degree in computer science from the Harbin Institute of Technology Harbin, China, in 2001, and the Ph.D. degree in computer networking from Peking University in 2006. He was a Researcher and the Research Manager with NEC Laboratories China from 2011 to 2012. He is currently a Senior Research Scientist with Hewlett-Packard Laboratories China. His research interests include wireless ad hoc network and wireless mesh network, data center networking, parallel computing, and android networking.