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

[1]Kai Chen, [2]Xitao Wen, [3]Xingyu Ma, [2]Yan Chen, [4]Yong Xia,
[5]Chengchen Hu, [6]Qunfeng Dong
[1]HKUST, [2]Northwestern, [3]UCLA, [4]NEC Labs China, [5]XJTU and NTNU, [6]Huawei

***Abstract*—Optical data center networks (DCNs) are becoming increasingly attractive due to their technological strengths compared to traditional electrical networks. However, prior optical DCNs are either hard to scale, vulnerable to single point of failure, or provide limited network bisection bandwidth for many practical DCN 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 (ToR) switches. WaveCube delivers high performance by exploiting multi-pathing and dynamic link bandwidth along the path. Our extensive evaluation results show that WaveCube outperforms previous optical DCNs by up to 400% and delivers network bisection bandwidth that is 70%-85% of an ideal non-blocking network 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 large scale, and its power consumption is 35% of them.

I. Introduction

Nowadays, data centers are being built around the world to support various big data applications and cloud services. As a result, the community has been investigating new DCN structures with the goal to better meet the bandwidth requirement of these applications. The representative DCN designs include pure electrical structures (*e.g.*, BCube [17], DCell [18], Fattree [7], PortLand [22], VL2 [16], CamCube [6], and Jellyfish [25]) and optical/electrical structures (*e.g.*, Helios [15], c-Through [27], OSA [12], and Mordia [23]). However, all these existing DCN designs have important drawbacks.

A. Motivation and Related Work

Electrical DCNs: Initially, people statically provision high capacity between all servers using sophisticated topologies like Fattree [7], BCube [17], and VL2 [16] with pure electrical devices. While this seems to be the only way to prevent any communication bottleneck assuming arbitrary traffic, it suffers from significant wiring challenge and management complexity. Furthermore, full bisection bandwidth at the scale of the entire DCN is not necessary given that not so many applications require uniform high bandwidth at this scale [15]. 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.

Optical DCNs	Scalability (port-count)	Performance	Fault-tolerance
c-Through [27] Helios [15]	Low (∼1000)	Low	No
OSA [12]	Low (∼2000)	High	No
Mordia [23]	Low (∼88)	High	No
WaveCube	High (unlimited)	High	Yes

TABLE I
SUMMARY OF PRIOR OPTICAL DCNS AND COMPARISON TO WAVECUBE.

Optical DCNs: To solve the dilemma, optical networking technologies, due to their ability to dynamically provision high bandwidth resources across the network, have been introduced in recent optical DCNs such as c-Through [27], Helios [15], OSA [12], and Mordia [23]. Most of them leverage MEMS-based optical switches to dynamically set up optical circuits for bandwidth demanding parts of the network. While making significant contributions in pointing out a promising avenue for building DCNs, these existing optical DCN designs suffer from the following issues (summarized in Table I).

- *Scalability.* MEMS is the central switch to connect all ToR switches, the low port density of MEMS limits the scalability[2]. A natural way to scale is to interconnect multiple MEMSes in the form of multi-stage fattree [27]. 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 remains a challenge.
- *Performance*. c-Through/Helios dynamically set up single-hop optical circuits between ToRs on-demand with low fan-in/out, which greatly restricts their performance when hotspots occur in high fan-in/out, a common pattern in production DCN workloads [19, 21]. OSA solves this problem by multi-hop routing on a k-regular topology (each ToR connects k other ToRs). But there is a tradeoff between k and network scale.
- *Fault-tolerance.* c-Through/Helios/OSA have all ToRs connect to a core MEMS, creating a single point of failure[3]. Mordia [23] resides on a ring, any link cut will break the ring and affect the connectivity.

Motivated by this situation, our goal is to design a scalable, fault-tolerant and high performance optical DCN architecture

[1]Micro-Electro-Mechanical-System—one of the most popular optical circuit switches used as the main component by recent optical DCNs [12, 15, 27].

[2]While Mordia does not have MEMS constraint, it is limited by wavelength channel contention and supports 88 ports on a single ring [23].

[3]We refer to physical structure reliability. Regarding the logical central controller, it can be replicated to multiple physical servers.

(Table I). To the best of our knowledge, prior optical DCNs do not 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 more 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 advanced routing mechanisms which otherwise cannot be easily achieved in a dynamic topology.

Our idea 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 MEMS-free optical DCN architecture that achieves scalability, fault-tolerance, and high-performance simultaneously (Section III). More specifically, WaveCube easily scales to hundreds of thousands of servers. It delivers high performance and fault-tolerance by exploiting multi-pathing and dynamic link bandwidth on each path—it outperforms previous optical DCNs by up to 400% and delivers network bisection bandwidth that is 70%-85% of a non-blocking network on 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 develop a polynomial-time optimal solution to wavelength assignment for dynamic link bandwidth (Section IV).
- We inspect 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 least power of all.

It is worthwhile to mention that WaveCube achieves all the properties without requiring any advanced, expensive optical devices beyond what are used by existing optical DCNs [12, 15, 23, 27]. 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.

Roadmap: Section II introduces the background. Section III presents WaveCube in detail. Section IV introduces wavelength assignment algorithms. Section V evaluates WaveCube. Section VI discusses practical deployment issues and hardware feasibility of WaveCube. Section VII concludes the paper.

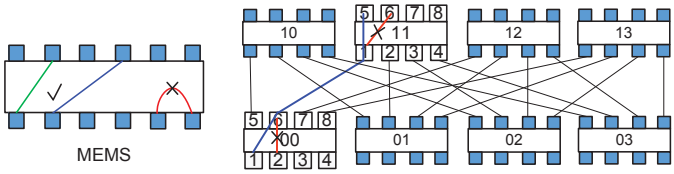

Fig. 1. An example of 2-stage MEMS structure.

II. Background and Problems

A. Optical Networking Technologies in DCNs

Optical networking technologies have been extensively introduced in optical DCNs [12, 15, 23, 27]. 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 ports by mechanically rotating micro-mirrors. The switching time of MEMS is around 10 milliseconds [26].

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-conflicting 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).

B. Problem Analysis with Multi-stage MEMSes

The low port density of MEMS (*i.e.*, 320 in practice or 1000 in Labs) is the bottleneck of scalability. A seemingly natural way to scale is to interconnect multiple MEMSes in the form of multi-stage fattree [27]. However, we show that such a multi-stage MEMS structure has fundamental problems.

First, MEMS switch only allows pairwise, bipartite connection between its ports. This makes MEMS chain less efficient. Figure 1 is an example of 2-stage MEMS structure. Even with such dense connectivity, due to pairwise circuit inside each MEMS, only one connection can be established between a top MEMS and a bottom MEMS at a time. For example, when we set up a circuit between port 1 of MEMS-00 and port 5 of MEMS-11 (blue), no other connection is allowed between MEMS-00 and MEMS-11 any more (red).

Second, multi-stage MEMSes incur significant cost. For instance, each MEMS port costs $\$500$ [15] and every additional 320-port MEMS costs additional $\$160,000$. Further, such a

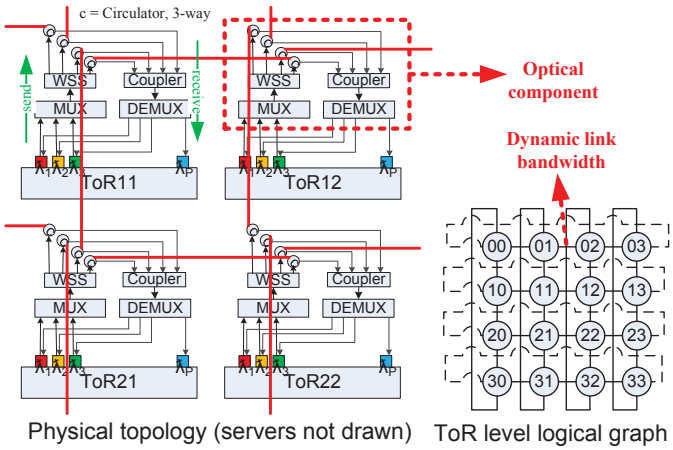

Fig. 2. The WaveCube architecture.

MEMS chain wastes lots of expensive optical ports purely for interconnection purpose.

Third, multi-stage MEMS structure poses challenge for fast coordinated circuit switching and increases signal loss at every stage. It is known that MEMS has ∼10ms switching latency. The synchronization of multiple MEMSes in a dense structure will cause additional switching latency. Hence, ongoing flows, especially those latency-sensitive ones, may get unacceptably delayed during reconfiguration.

Given the above problems with multi-stage MEMSes, we exploit an opposite direction. Instead of adding more MEMSes, we completely remove it. In what follows, we present the design and evaluation of WaveCube, a MEMS-free optical DCN that trades dynamic topology for scalability and fault-tolerance, while still conserving network performance via multi-pathing and dynamic link bandwidth scheduling.

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 2), 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.[4] 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 2, where $n = 2$ and $k_0 = k_1 = 4$, $D_W((1,1),(3,0)) = 2 + 1 = 3$. 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).

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.

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 [11, 12, 15]–[17, 22, 27, 29].

B. Multi-pathing

Node-disjoint paths between two ToRs provide a means of selecting alternate routes and increase fault-tolerance. WaveCube provides high fault-tolerance as it contains $2n$ 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: For space limitation, we omit the details. Note that our proof is inspired by [14] on regular k-ary-n-cube topology, and we applied similar approach to WaveCube with variable radix on each dimension. As a result, the $2n$ node-disjoint paths in WaveCube have different lengths from that in [14]. Interested readers please refer to [14]. ■

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

C. Dynamic Link Bandwidth

Over the $2n$ multi-paths, WaveCube enables dynamic link bandwidth on each path by using WSS (e.g., 1×K). Take ToR11 in Figure 2 for example, as a sender, all wavelengths from ToR are multiplexed onto a single fiber by MUX to feed WSS. Then, WSS divides these wavelengths into K groups, each group goes to another ToR through one of the K outgoing ports. The number of wavelengths in a group amounts to the bandwidth of that ToR-to-ToR 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, K groups of wavelengths from K other ToRs will be transmitted to Coupler through the 3-way Circulators, then the Coupler multiplexes all these wavelengths to a single

[4]Essentially, WaveCube is a generalized k-ary-n-cube [13] with variable radices. CamCube [6] used a k-ary-n-cube, 3D Torus, for its server-centric network topology. However, WaveCube is switch-centric and, more importantly, the link bandwidth of WaveCube can be dynamically adjusted.

fiber to feed DEMUX, and finally the DEMUX de-multiplexes all the wavelengths to their corresponding ports on ToR.

However, wavelength contention requires that the same wavelength cannot be assigned to a ToR link twice simultaneously. *This is because all the wavelengths from/to a ToR will share the same single fiber of MUX-to-WSS/Coupler-to-DEMUX.* 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 prior optical DCN [12]. In Section IV, we will introduce an optimal wavelength assignment algorithm by taking advantage of WaveCube topology properties.

It is worthwhile to note that WaveCube's use of WSS is inspired by OSA [12], but significantly improves by designing the optimal wavelength assignment and optimized wavelength adjustment algorithms (Section IV).

D. Optimization with above two properties

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

Flow scheduling: For an incoming flow, we need to choose, among $2n$ parallel paths, one path to route the flow. There are many methods to use, such as random, round-robin, ECMP, etc. Recent work, such as Hedera [8], also introduced advanced DCN flow scheduling. However, WaveCube does not require advanced (high-overhead) 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 ϕ 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 (1) specifies the goal of maximizing y^{ϕ}, which is equivalent to minimizing the maximum link 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).

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

TABLE II
SOME KEY NOTATIONS USED IN SECTION IV.

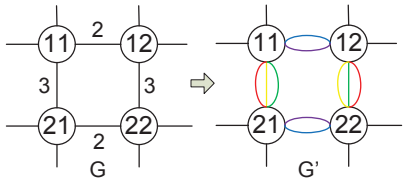

Fig. 3. Example of multigraph construction.

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 on the link, which is a positive integer. Constraint (4) relaxes an integer to be a real number, which we will round back to an integer later. As shown in Table III, this linear program can be efficiently solved with 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 2), each ToR up-port is bound to a fixed unique wavelength, and the wavelengths for all ToRs are the same. Furthermore, 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-conflicting wavelengths to satisfy ϕ.

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-conflicting 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 3 (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-conflicting wavelengths is equivalent to an edge-coloring solution [1] on the multigraph, where each color represents a distinct wavelength and no two adjacent edges share the same color.

However, the edge coloring problem on a general multigraph G' is NP-complete [1], and the minimal number of colors needed in an edge coloring $\chi(G') \in [\Delta(G'), \Delta(G')+$

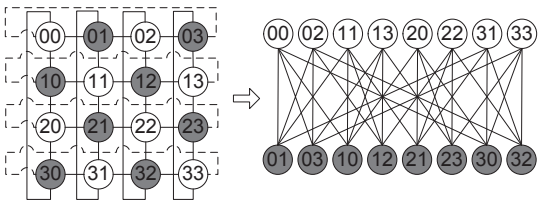

Fig. 4. An example of topology transformation.

$\mu(G')]$, 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 prior work [12]. WaveCube solves the problem by designing a polynomial-time optimal wavelength assignment algorithm that takes advantage of the WaveCube topology properties.

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

It is clear that at least $\Delta(G')$ wavelengths are needed to provision ϕ. Theorem 2 guarantees that we can always use this minimum number of $\Delta(G')$ wavelengths to provision ϕ. We 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 [1, 12]. However, WaveCube is designed in such a way that its topology is guaranteed to be bipartite, for which an elegant polynomial-time algorithm can be found for optimal wavelength assignment.

Proof: To show its bipartite nature, we randomly select a node and mark it "black", and then starting from this black node, we mark all its neighbors "white". Then, for each white (or black) node, we mark its neighbors black (or white) iteratively until all the nodes are covered. Since in WaveCube, each 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 4 illustrates an example of topology transformation.

We next theoretically prove this. 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 comprises 3 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 5); 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. ■

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

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" [24]. 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.

B. Optimized Wavelength Adjustment

Traffic may change and link bandwidth needs adjustment to better fit traffic. Once bandwidth demand ϕ changes, we need to re-assign wavelengths to satisfy new ϕ accordingly. A naive approach is to assign wavelengths from scratch without considering the old assignment. However, given that shifting a wavelength from one WSS port to another incurs $\sim$10ms latency, wavelength re-assignment should shift minimal wavelengths. This minimizes the disruption of ongoing traffic especially those latency-sensitive flows.

Problem 2. Minimal Wavelength Adjustment (MWA): Given a 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 formulated MWA problem as a 0-1 integer linear program, and proved it is NP-hard. We then design a heuristic algorithm in Figure 6. The basic idea is to use the old wavelength distribution $\lambda_o = \{m_1, m_2, \cdots, m_\Delta\}$ to assist the decomposition of new multigraph G'_n into Δ matchings

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 5 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 [2] algorithm;
22 return X_{ij}; /* $x_{ij}=1$ means assigning c_i (color of m_i) to m'_j */

Fig. 6. A heuristic algorithm of MWA problem.

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 [2]).

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 5 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-D.

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. Evaluation 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 matrices from a production DCN with ~400 servers with 1G ports. The data center runs Map-reduce style applications. To replay the traffic over 8640 servers with 10G ports, we shrink the transmission time and replicate traffic spatially.
- *Microsoft-based.* We synthesize traffic patterns based on measurement results from recent works [9, 19, 21] 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 [19], and most of the traffic (80%) are within the rack [9]. 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.

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

Simulator: We implement our own simulator as 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, sources and destinations. The simulation runs in discrete time ticks with the granularity of ms. On each tick, the rate of each flow is updated by running on all active flows the progressive filling algorithm [4], 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. We run on a Linux server with Dual Xeon X5560 @ 2.8GHz CPU and 32G memory.

B. Achieved Network Bisection Bandwidth

Figure 7 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 and a hypothetical non-blocking network, which serves as the upper-bound of performance for any DCN.

From the figure, we find that WaveCube outperforms c-Through by 300%-400% and delivers network bisection bandwidth that is 70%-85% of non-blocking under all traffic patterns. This is not a surprising result. Because c-Through assumes one-hop pairwise circuits in its optical part, such

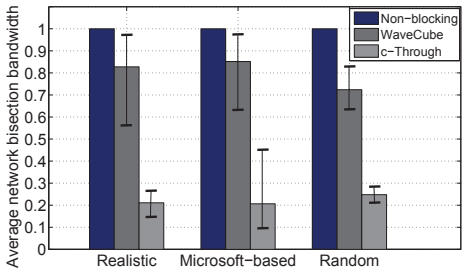

Fig. 7. Network bisection bandwidth (*Note: Helios performs similarly as c-Through, OSA/Mordia perform better but they cannot support 8640 servers*).

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 fixed 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. 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. 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 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 8. 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.

D. Quality of Wavelength Adjustment

We evaluate the quality of our wavelength adjustment algorithm in Figure 6 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.

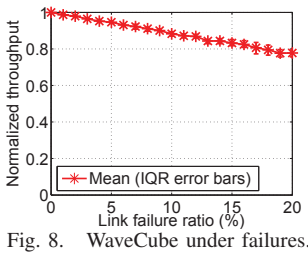

Fig. 8. WaveCube under failures.

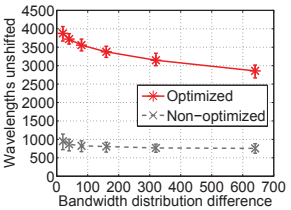

Fig. 9. Wavelength adjustment.

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

TABLE III
TIME COST OF LBS AND WAO.

Results in Figure 9 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.

E. Overhead of the Central Controller

The central controller handles most of intelligence 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 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 measure time cost of two main algorithms executed by the controller: link bandwidth scheduling (LBS, Section III-D) and wavelength adjustment optimization (WAO, Section IV-B). Table III shows the results. 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.

F. 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 [28]. Another

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

TABLE IV
WIRING COMPLEXITY OF DCNS[5].

study [10] found that 60% of ToR-pairs see less than 20% change in demand for seconds. Further, recent work [19] 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, there do exist dynamic workloads [20], and we note that if the traffic is highly dynamic, WaveCube's performance would be unpredictable. We will further evaluate WaveCube under highly dynamic traffic and design counter-measures in future work.

VI. PRACTICAL DEPLOYMENT ANALYSIS

We discuss practical deployment issues like wiring, cost and power of WaveCube, and compare with prior DCNs. Given a large body of recent designs, we select Fattree [22] (switch-centric), BCube [17] (server-centric) and c-Through [27] (optical) as representatives. Then, we assess the hardware feasibility of WaveCube by comparing it with OSA [12].

A. Wiring Complexity

We observe that many decent DCNs [16]–[18, 22], while providing good performance, are hard to construct in practice due to dense topologies and strict wiring rules. For example, many wires must be connected between specific devices or specific ports. How to build a data center is a practical question, especially when the DCN is large. Comparing to recently proposed DCNs, WaveCube is perhaps among the easiest-to-build ones in terms of wiring.

Directly counting the number of wires as [17] did is not a good way to quantify the wiring complexity. Because not all the wires have the same difficulty to set up in practice. To this end, we classify wires into *rule-based* and *rule-free*. The rule-free wire refers to the wire that if one end is fixed, the other end can be freely selected from a set of devices or ports. Usually, wires of servers to ToR are rule-free and very easy to connect. In contrast, a rule-based wire requires that once one end is fixed, the other end must be connected to certain device or port. Such rule-based wiring is usually error-prone and needs special care. Hence, the complexity of wiring mainly comes from the rule-based wires.

To quantify the complexity, we further understand placement of devices in DCNs. For maintenance and management, devices are arranged in racks, and racks are organized in rows and columns [5]. Thus, we assign each rack a coordinate (i, j). 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 [3]) to estimate the wiring length between them.

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	-	-	-

TABLE V
COST AND POWER FOR DIFFERENT DEVICES (†PER PORT VALUE), SOME VALUES ARE REFERRED FROM [15].

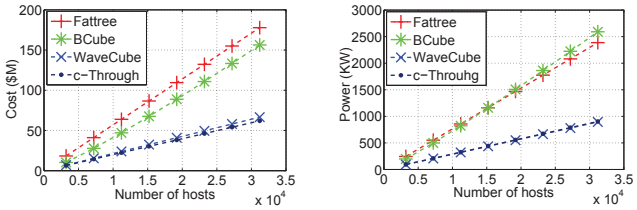

Fig. 10. Cost and power for different DCNs.

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:

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

With (5), we compare WaveCube with Fattree, BCube and c-Through. In order to make the comparison more accurate, we compute the complexity of each DCN according to its own structure characteristics. We put racks in rows and columns, and place servers and switches in a way that the length of rule-based wires is optimized.

The results are shown in Table IV[5]. It can be seen that WaveCube is 2-3 orders of magnitude simpler than Fattree/BCube and 1 order simpler than c-Through at large scale. When the size grows, the complexity of WaveCube grows much slower than the others. The advantages of WaveCube come from 2 main reasons: 1) A single optical fiber can aggregate high data volume which otherwise need to be carried by many copper cables; 2) 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. 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 [15]. So long-distance, cross-rack links in Fattree or BCube must be optical.

Figure 10 shows the results. It is evident that, to host the same number of servers, WaveCube is significantly cheaper than either Fattree (∼35%) or BCube (∼40%) and consumes much less power (∼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

[5]Many other DCNs like VL2 [16], DCell [18], Helios [15], OSA [12], etc., are not listed here. Basically, VL2 is similar to Fattree. DCell is server-centric as BCube, they have extremely complex wiring rules making them hard to build. Helios and OSA are similar to c-Through.

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.

C. WaveCube Hardware Feasibility

The crux of showing the feasibility of WaveCube is to demonstrate the feasibility of optical component in Figure 2. 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 OSA testbed to discuss the hardware feasibility of WaveCube. (Interested readers please refer to [12] for details of OSA testbed.)

In the OSA testbed, instead of direct connection, one Polatis series-1000 OSM/MEMS with 32 ports (16×16) 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 simulations, this seemingly simple architecture re-design has translated to significant benefits in scalability, fault-tolerance, as well as the optimal wavelength assignment.

From the implementation perspective, WaveCube can be built by fixating the MEMS circuits and simply treating them as dumb fibers. Therefore, the extensive feasibility study in OSA paper [12], as a side effect, 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 a scalable optical DCN. Thus, we have focused on evaluating WaveCube in large-scale simulated settings in last section, leaving the implementation of a non-trivial WaveCube prototype as future work.

VII. 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 since there is no single point of failure and multiple node-disjoint paths exist between any pair of ToRs. WaveCube delivers high performance by exploiting multi-pathing and dynamic link bandwidth. Our evaluation shows that WaveCube achieves all its design goals and our practical deployment analysis shows that it holds promise in practice.

Acknowledgements This work was supported by Hong Kong RGC-ECS 26200014, China 973 Program No.2014CB340303, NSF NeTS 1219116, NSFC 61272459, and Program for New Century Excellent Talents in University. We thank Yongqiang Liu for his help in this project.

References

[1] Edge coloring, http://en.wikipedia.org/wiki/edge_coloring.
[2] Hungarian algorithm, http://en.wikipedia.org/wiki/hungarian_algorithm.
[3] Manhattan distance, http://en.wiktionary.org/wiki/manhattan_distance.
[4] Progressive filling, http://en.wikipedia.org/wiki/max-min_fairness.
[5] D. Abts and J. Kim. High performance datacenter networks: Architectures, algorithms, and opportunity. *Synthesis Lectures on Computer Architecture*, 2011.
[6] H. Abu-Libdeh, P. Costa, A. Rowstron, G. O'Shea, and A. Donnelly. Symbiotic routing in future data centers. In *SIGCOMM*, 2010.
[7] M. Al-Fares, A. Loukissas, and A. Vahdat. A scalable, commodity data center network architecture. In *ACM SIGCOMM*, 2008.
[8] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat. Hedera: Dynamic flow scheduling for data center networks. In *NSDI'10*.
[9] T. Benson, A. Akella, and D. Maltz. Network traffic characteristics of data centers in the wild. In *ACM IMC*, 2010.
[10] T. Benson, A. Anand, A. Akella, and M. Zhang. The case for fine-grained traffic engineering in data-centers. In *INM/WREN*, 2010.
[11] K. Chen, C. Guo, H. Wu, J. Yuan, Z. Feng, Y. Chen, S. Lu, and W. Wu. Generic and automatic address configuration for data centers. In *SIGCOMM*, 2010.
[12] K. Chen, A. Singla, A. Singh, K. Ramachandran, L. Xu, Y. Zhang, X. Wen, and Y. Chen. Osa: An optical switching architecture for data center networks with unprecedented flexibility. In *NSDI*, 2012.
[13] W. J. Dally. Performance analysis of k-ary n-cube interconnection networks. *IEEE Transaction on Computer*, 1990.
[14] K. Day and A. E. Al-Ayyoub. Fault diameter of k-ary n-cube networks. In *IEEE TPDS*, 1997.
[15] N. Farrington, G. Porter, S. Radhakrishnan, H. H. Bazzaz, V. Subramanya, Y. Fainman, G. Papen, and A. Vahdat. Helios: A hybrid electrical/optical switch architecture for modular data centers. In *SIGCOMM*, 2010.
[16] 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 *ACM SIGCOMM*, 2009.
[17] 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. In *SIGCOMM*, 2009.
[18] 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 *SIGCOMM*, 2008.
[19] D. Halperin, S. Kandula, J. Padhye, P. Bahl, and D. Wetherall. Augmenting data center networks with multi-gigabit wireless links. In *ACM SIGCOMM*, 2011.
[20] S. Kandula, J. Padhye, and P. Bahl. Flyways to de-congest data center networks. In *HotNets*, 2009.
[21] S. Kandula, S. Sengupta, A. Greenberg, P. Patel, and R. Chaiken. The nature of datacenter traffic: Measurements and analysis. In *IMC*, 2009.
[22] 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. In *SIGCOMM*, 2009.
[23] G. Porter, R. Strong, N. Farrington, A. Forencich, P.-C. Sun, T. Rosing, Y. Fainman, G. Papen, and A. Vahdat. Integrating microsecond circuit switching into the data center. In *SIGCOMM*, 2013.
[24] A. Schrijver. Bipartite edge-colouring in o(δm) time. *SIAM Journal on Computing*, 1998.
[25] A. Singla, C.-Y. Hong, L. Popa, and P. B. Godfrey. Jellyfish: Networking data centers randomly. In *NSDI*, 2012.
[26] T. Truex, A. A. Bent, and N. W. Hagood. Beam steering optical switch fabric utilizing piezoelectric actuation technology. In *NFOEC*, 2003.
[27] G. Wang, D. Andersen, M. Kaminsky, K. Papagiannaki, T. Ng, M. Kozuch, and M. Ryan. c-Through: Part-time optics in data centers. In *SIGCOMM*, 2010.
[28] X. Wen, K. Chen, Y. Chen, Y. Liu, Y. Xia, and C. Hu. Virtualknotter: Online virtual machine shuffling for congestion resolving in virtualized datacenter. In *ICDCS*, 2012.
[29] Y. Zhang, C. Guo, D. Li, R. Chu, H. Wu, and Y. Xiong. Cubicring: Enabling one-hop failure detection and recovery for distributed in-memory storage. In *NSDI*, 2015.