Dynamic Topology Management in Optical Datacenter Networks

Yangming Zhao, Sheng Wang, Shouxi Luo, Hongfang Yu, Shizhong Xu and Xiaoning Zhang
Key Lab of Optical Fiber Sensing and Communication, Education Ministry of China
University of Electronic Science and Technology of China

***Abstract*—In this paper, we study how to manage the topology reconfiguration in OSA-based datacenter networks (DCNs). Though an OSA-based DCN can change its topology to adapt to the traffic matrix and improve the network scalability, it requires too much time (10ms) to reconfigure the topology, which may not only incur a great amount of traffic loss in high throughput low latency DCNs, but also bring much performance degradation to the delay sensitive flows. Therefore, a progressive topology reconfiguration scheme is required to reduce the traffic loss and guarantee the performance of delay sensitive flows. To this end, we first formulate the problem as a mathematical model, and then analyze its feasibility and complexity. Based on these analyses, topology management algorithm (TMA) is proposed to calculate the topology reconfiguration scheme that can maintain the topology connectivity during reconfiguration. By simulation, we find that TMA can reduce the traffic loss during topology reconfiguration by up to 50% in most of the cases and reconfigure topology without traffic loss in some cases.**

I. Introduction

Nowadays, datacenter plays a vital role in our daily life. For example, many online services provided by Amazon, Google, FaceBook, and eBay are supported by datacenters hosting hundreds of thousands of servers. The network interconnection of the datacenter heavily impacts the performance of the datacenter, such as the datacenter scalability, job throughput and response delay. Accordingly, design a topology with high bisection bandwidth and rich connectivity may bring great benefits to the datacenter networks (DCNs).

To provide a high connectivity and rich bisection bandwidth for DCNs, one approach is statically provisioning high bandwidth and rich connectivity to each pair of hosts like Fattree [1], VL2 [2], BCube [3] and DCell [4]. In many real traces, however, we see that such a network may not be fully utilized and the network resource is wasted. To overcome this shortage, wireless communication technologies (e.g., Flyways [5] and 3D beamforming [6]) or optical switching technologies (e.g., OSA [7] , Helios [8] and c-Through [9]) are exploited to design dynamic topologies for DCNs. Using these dynamic topologies, the network interconnection can adapt to the traffic to enhance the network resource utilization. Accordingly, with the explosively increasing of DCN size, dynamic topology is the most promising technology to realize the DCN interconnection.

OSA [7] is the state-of-the-art work to provide the most flexibility. By leveraging the dynamic optical switching module such as Micro-Electro-Mechanical Switch (MEMS) and Wavelength Selective Switch (WSS), it can form any K-regular topology (where K is a predefined number according to the module connection) and adapt both the topology and link capacity to the traffic. In this paper, we will discuss how to manage the topology reconfiguration of OSA. The technology details of OSA will be reviewed in Section II-A.

Though dynamic topology technologies can greatly improve the network resource utilization, an important issue in these technologies is how to online reconfigure the topology. A naive method is to calculate the topology and link capacity for the traffic matrix in each time slot and reconfigure all the links simultaneously. However, it is obviously inefficient since it may break off all the traffic in the network and cause a great amount of traffic loss. Suppose in an OSA-based DCN, there are 80 Top-of-Racks (ToRs), each ToR supports 32 wavelengths and the bandwidth of each wavelength is 10Gbps. Since MEMS needs about 10ms to reconfigure a new topology by mechanically adjusting a microscopic array of mirrors [7], reconfigure the topology once may directly incur 256Gbit traffic loss in the worst case. It should be noted that this calculation does not take the TCP retransmission into account.

Another motivation to study the topology reconfiguration is that most of the flows in the DCNs are mice flows that is delay sensitive. Previous measurement study [10] shows that 80% of the flows in the datacenter are smaller than 10KB, but most of the bytes are in the 10% of large flows. In a high speed and low latency DCN, these mice flows smaller than 10KB are finished in 1 or 2 RTTs which is less than 1ms, so that 10ms is a too large overhead for these mice flows. More importantly, some of these mice flows are delay sensitive and have deadlines. 10ms is a very common application latency target in DCNs [11]. Accordingly, design a progressive topology reconfiguration method is important for guaranteeing the performance of these mice flows and delay sensitive flows.

Based on the discussion above, the first objective we pursue when reconfiguring the DCN topology is to guarantee the network connectivity. Only in this case, the performance of mice flows and deadline-aware flows can be guaranteed. Secondly, we should try to minimize the traffic loss during the

Corresponding author: Sheng Wang (wsh_keylab@uestc.edu.cn)
This work was partially supported by the 973 Program (2013CB329103), NSFC Fund (61271165, 61271171, 61201129 and 61301153), the Fundamental Research Funds for the Central Universities.

978-1-4799-3512-3/14/$31.00 ©2014 IEEE

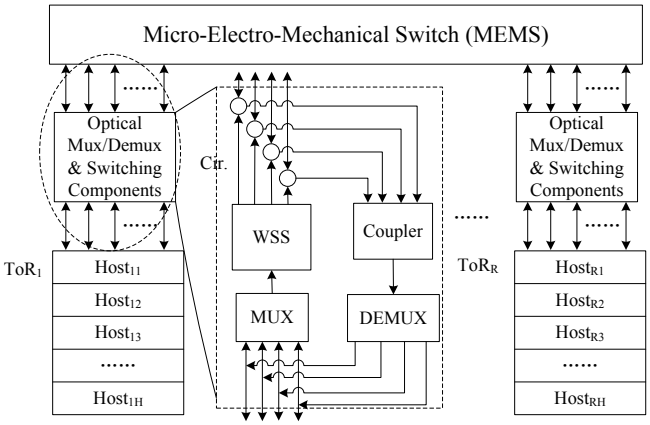


Fig. 1. Optical Switching Architecture (OSA)

intermediate states when reconfiguring the network topology. Last but not least, the reconfiguration duration also need to be minimized with the purpose of maintaining network stability. As far as we know, we are among the first to design intermediate topologies to bridge different topologies in dynamic DCNs, though there are many works on how to adjust the traffic routing without data loss when the traffic matrix changes [12] or some network component failure occurs [13].

The main contributions of our work can be summarized as follows:

- We formulate the problem of managing topology reconfiguration and analyze its feasibility and complexity (Section III).
- Based on these analyses, a connectivity guaranteed topology management algorithm (TMA) is proposed to deal the topology reconfiguration in dynamic DCNs (Section IV).
- By simulation, we find that TMA can reduce the traffic loss during topology reconfiguration by up to 50% in most of the cases and update topology without traffic loss in some cases (Section V).

II. Related Work

In this section, we briefly review the works that are mostly related to our work. In Section II-A, we summarize the main characteristics of the optical switching architecture (OSA) which is the fundamental of our work. After that we discuss the existing works that focus on intermediate plans/schemes to bridge different states in Section II-B.

A. Optical Switching Architecture

The architecture of OSA is shown in Fig. 1. It achieves dynamic topologies via exploiting the reconfigurability of Micro-Electro-Mechanical Switch (MEMS). MEMS is a bipartite $N \times N$ matrix, in which any input port can be connected to any one of the output ports. If we connect each of $\frac{N}{d}$ ToRs to d ports of MEMS (assume N is divisible by d), it means that each ToR is directly connecting with d ToRs. By these connections, the OSA can form all the d-regular topologies for DCNs.

Another advantage of the OSA is that it can flexibly modify the capacity of each link based on Wavelength Selective Switching (WSS). If ToR i is to create a link to ToR j with capacity that is k times of a single wavelength, k wavelengths can be assigned to this link. To this end, OSA first multiplexes all the wavelengths from ToR i into a single fiber connecting a WSS module. Then the WSS module splits the expected k wavelengths to the appropriate port connected with ToR j. In this way, the operator can easily change the capacity of each link by switching selected wavelengths to different links under the constraint that each ToR port is assigned with a special wavelength across its ToR. A constraint is made because of that a fiber cannot carry more than one channels over the same wavelength.

B. Intermediate Plans

Many existing works focus on network intermediate plans (such as routing, topology, link weight etc.) to bridge different network states with the purpose to avoid traffic loss. SWAN [12] reserves some redundant capacity on each link and uses a serial of linear programming (LP) models to calculate a routing modification sequence such that no data loss occurs due to the difficulty in synchronizing routing updates. This work only focuses on the traffic routing changes in the network without considering the topology changes as in our work.

zUpdate [13] focuses on almost the same topic as SWAN and uses LP models to calculate intermediate routing schemes. The difference is that zUpdate supposes that some network component faces failures and the topology may change in some scenarios. However, the final topology is a subgraph of the initial one, which is not the case in our work.

In addition to LP, dynamic programming (DP) is another method to calculate the intermediate plans between the initial and final state of network state migrations. [14] uses DP to study the link weight reassignment scheduling problem and link maintenance scheduling problem, both of which are combinatorial optimization problems. As in zUpdate, it assumes that only a small part of the network topology changes in the intermediate states and does not focus on how to bridge two network states using intermediate network states.

III. Problem Formulation and Analysis

As discussed in Section I, we should first guarantee the network connectivity, and then minimize the traffic loss during topology recofiguration. In this section, we formulate this problem as an optimization model in Section III-A, and analyze its feasibility and complexity in Section III-B.

A. Problem Formulation

Notation List:

$\boldsymbol{G}_i$, $i = 1, 2 \ldots, n-1$: The i^{th} intermediate topology during the topology reconfiguration. Especially, we use $\boldsymbol{G}_0$ and $\boldsymbol{G}_n$ to denote the initial and final topologies, respectively.

P^i_{sd}, $i = 1, 2 \ldots, n$: The set of edges used by the path from s to d in $\boldsymbol{G}_{i-1} \cap \boldsymbol{G}_i$.

$\boldsymbol{W}^i_{uv}$: The set of wavelengths used by edge (u, v) in $\boldsymbol{G}_{i-1} \cap \boldsymbol{G}_i$.

C: Constant. The bandwidth of a wavelength in the system.

f_{sd}: Constant. The required traffic rate from ToR s to ToR d.

r^i_{sd}: The real traffic rate from ToR s to ToR d in $\boldsymbol{G}_{i-1} \cap \boldsymbol{G}_i$.
l^i_{uv}: The traffic rate on edge (u, v) in $\boldsymbol{G}_{i-1} \cap \boldsymbol{G}_i$.
Optimization Model:

$$Objective: \quad \text{minimize} \sum_{s,d} \sum_{i=1}^{n} (f_{sd} - r^i_{sd}) \tag{1}$$

$$Subject\ to: \quad \text{All the constraints due to the OSA} \tag{2}$$

$$\forall 0 \le i \le n-1, \boldsymbol{G}_i \cap \boldsymbol{G}_{i+1} \text{ is connected} \tag{3}$$

$$\forall 1 \le i \le n-1, l^i_{u,v} = \sum_{(s,d):(u,v)\in \boldsymbol{P}^i_{sd}} r^i_{sd} \tag{4}$$

$$\forall (u,v) \text{ and } 1 \le i \le n-1, l^i_{uv} \le C|W^i_{uv}| \tag{5}$$

$$\forall (u,v) \text{ and } 1 \le i \le n-1, r^i_{sd}, \le f_{sd} \tag{6}$$

$$\forall (s,d) \text{ Flow conservation} \tag{7}$$

Considering that MEMS spends about 10ms to reconfigure the port interconnection [7], minimize the total traffic loss is equivalent to minimize the sum of overflow traffic rate in each intermediate topology. It should be noted that n is a variable in this problem, so that reduce the number of intermediate topologies is also a way to pursue the objective.

On the other hand, constraint (3) guarantees that there is a connected common subgraph between two adjacent intermediate topologies. In this case, we can first route all the traffic through the connected common subgraph such that we can at least serve the delay sensitive flows without traffic loss.

The remaining constraints are common constraints in general network programming problems. Equation (2) is the OSA-based system constraints, such as k-regular topology, avoid wavelength contention, etc. (see detail in Section II-A). Constraint (4) is used to calculate the traffic rate on each edge and Constraint (5) identifies that the traffic rate on each edge cannot exceed the total bandwidth that can be provided by all the wavelengths on that edge. Constraint (6) says that the real traffic rate between a pair of ToRs would not exceed their demands while the last constraint is the flow conservation that must be satisfied in all the network traffic problems.

B. Feasibility and Complexity Analysis

Though we have formulated the problem as an optimization model in last section, it cannot be solved directly, since it is not a convex programming problem and difficult to solve. Accordingly, we study the feasibility and complexity of this problem which may provide some hints to solve it. The only constraint that may result in a infeasible problem is (3). In other words, if there is always a graph sequence satisfying (3), our problem is always feasible. Consequentially, we first propose following theorem:

Theorem 1. *If $\boldsymbol{G}_0$ and $\boldsymbol{G}_n$ are two connected k-regular graphs ($k \ge 3$) with the same node set, there must be a graph sequence $\boldsymbol{G}_1, \boldsymbol{G}_2, \ldots, \boldsymbol{G}_{n-1}$, such that*
1) $\boldsymbol{G}_i \cap \boldsymbol{G}_{i+1}$ for $i = 0, 1, \ldots, n-1$ are connected graphs;
2) $\Delta(\boldsymbol{G}_i) \le k$ for $i = 1, 2, \ldots, n$.

To prove this theorem, we first introduce several lemmas.

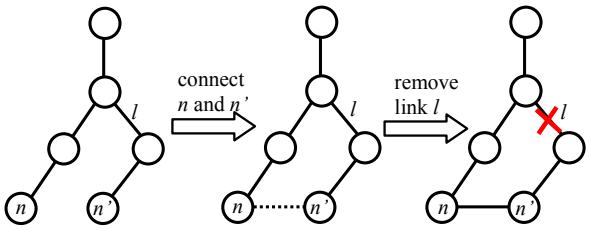


Fig. 2. Example for the "while" loop in Algorithm 1

Lemma 1. *For any connected k-regular graphs ($k \ge 3$) $\boldsymbol{G}_0$, there is a graph sequence $\boldsymbol{G}_1, \boldsymbol{G}_2, \ldots, \boldsymbol{G}_m$, such that*
1) $\boldsymbol{G}_i \cap \boldsymbol{G}_{i+1}$ for $i = 0, 1, \ldots, m-1$ are connected graph;
2) $\boldsymbol{G}_m$ is a cycle;
3) $\Delta(\boldsymbol{G}_i) \le k$ for $i = 1, 2, \ldots, m$.

Proof: We prove this lemma by constructing a graph sequence satisfying all the three conditions. The Algorithm 1 is designed to construct such graph sequence. Next, we prove that the graph sequence generated by Algorithm 1 is satisfying all the three conditions in Lemma 1.
Condition 1: It is obvious that $\boldsymbol{G}_0 \cap \boldsymbol{G}_1$ is connected since $\boldsymbol{G}_1$ is the spanning tree of $\boldsymbol{G}_0$. When i is odd, $\boldsymbol{G}_i \cap \boldsymbol{G}_{i+1} = \boldsymbol{G}_i$, while $\boldsymbol{G}_i \cap \boldsymbol{G}_{i+1} = \boldsymbol{G}_{i+1}$ when i is even. Accordingly, we should prove $\boldsymbol{G}_i$ is a connected graph for all i. For odd i, $\boldsymbol{G}_{i+1}$ is obvious connected if $\boldsymbol{G}_i$ is connected, since it is derived by adding an edge on a connected graph. For even i, we should note that there is a cycle in $\boldsymbol{G}_i$ since we derive $\boldsymbol{G}_i$ by connecting two leaves in $\boldsymbol{G}_{i-1}$. Start a travesing from one of the leaves in $\boldsymbol{G}_{i-1}$ that are selected to connect, if l is the first edge connecting a node with more than 2 nodal degree, l must be on the cycle. Hereby, remove link l from $\boldsymbol{G}_i$ does not divide it into multiple disconnected parts. It is worth noting that $\boldsymbol{G}_i$ is always a tree since there is only $N-1$ edges in the connected graph.
Condition 2: Fig.2 shows how the "while" loops in Algorithm 1 works. In each loop, two leaves in the graph are eliminated and at most 1 leaf is generated. After $N-2$ loops, there is at most 2 leaves in the graph and it must be a line. In this case, a cycle can be formed by connecting the two leaves.
Condition 3: Since $\boldsymbol{G}_0$ is a k-regular graph and $\boldsymbol{G}_1$ is the spanning tree of $\boldsymbol{G}_0$, there must be $\Delta(\boldsymbol{G}_1) \le k$. For odd i, $\boldsymbol{G}_{i+1}$ is derived from connecting to leaf nodes which can only generate two nodes with nodal degree $2 < k$. For even i, $\boldsymbol{G}_{i+1}$ is yielded by deleting an edge from $\boldsymbol{G}_i$, which cannot increase the nodal degree. ■

Lemma 2. *For two cycles, $\boldsymbol{G}$ and $\boldsymbol{G}^*$, that have the same node set and there are at least 4 nodes in each cycle , there must be a graph sequence, $\boldsymbol{G}_1, \boldsymbol{G}_2, \ldots, \boldsymbol{G}_m$, such that*
1) $\boldsymbol{G} \cap \boldsymbol{G}_1$, $\boldsymbol{G}_i \cap \boldsymbol{G}_{i+1}$ for $i = 1, 2, \ldots, m-1$ and $\boldsymbol{G}_m \cap \boldsymbol{G}^*$ are connected graphs;
2) $\Delta(\boldsymbol{G}_i) \le 3$ for $i = 1, 2, \ldots, m$.

Proof: To prove this lemma, we can only prove that there is a graph sequence that satisfies the two conditions in the lemma to swap any two nodes on the cycle. As shown in Fig.3, we want to swap the location of node B and node E.

Algorithm 1: Transform a graph to a cycle

Input: A connected graph $\boldsymbol{G}$

Output: A graph sequence $\{\boldsymbol{G}_i\}$ statisfying Lemma 1

$i \leftarrow 1$
Calculate a spanning tree of $\boldsymbol{G}$ and set it to be $\boldsymbol{G}_i$, $i \leftarrow i+1$
while $\boldsymbol{G}_i$ has more than 2 leaves **do**
  Select 2 leaves in $\boldsymbol{G}_i$, say n and n'.
  $\boldsymbol{G}_{i+1} \leftarrow \boldsymbol{G}_i + (n, n')$, $i \leftarrow i+1$
  Start at n' to find out the first edge connecting a node with more than 2 nodal degre, say it is l.
  $\boldsymbol{G}_{i+1} \leftarrow \boldsymbol{G}_i - l$, $i \leftarrow i+1$
end while
Say n and n' is the last two leaves, $\boldsymbol{G}_{i+1} \leftarrow \boldsymbol{G}_i + (n, n')$
return $\{\boldsymbol{G}_i\}$

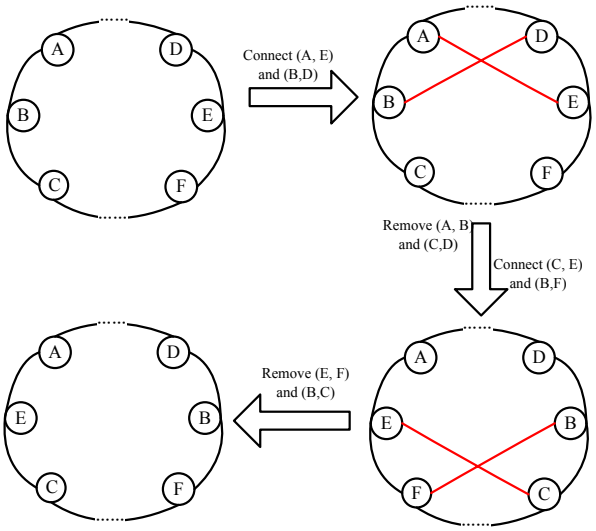


Fig. 3. How to swap the location of two nodes on the cycle

We first connect (A, E) and (B, D). After that, remove edge (A, B) and (D, E) will not impact the graph connectivity. Till now, the node order on the upper side of the cycle is fixed. The same procedure should be executed on the lower side of the cycle as in Fig.3. Especially, if there are less than 2 nodes between the two nodes to be swapped, only 1 intermediate topology is required since we only need to fix the order on one side of the cycle. ■

Based on the proof of Lemma 2, we can simply design Algorithm 2 to convert one cycle to another. It is worth noting that all the transformation procedures in both Algorithm 1 and Algorithm 2 are reversible, i.e. if the graph sequence $\{\boldsymbol{G}_i\}$ can transform $\boldsymbol{G}_0$ to a cycle $\boldsymbol{G}_m$, the inverse sequence of $\{\boldsymbol{G}_i\}$ can transform the cycle $\boldsymbol{G}_m$ to graph $\boldsymbol{G}_0$, so is the procedure to convert one cycle to another. Accordingly, the Theorem 1 can be proved by constructing such graph sequence. Hereafter, we used $\{\boldsymbol{G}_i\}^{-1}$ to denote the inverse sequence of $\{\boldsymbol{G}_i\}$.

Proof: Assume graph sequence $\{\boldsymbol{A}_i\}$ can transform $\boldsymbol{G}_0$ to a cycle $\boldsymbol{C}$ and $\{\boldsymbol{B}_i\}$ can transform $\boldsymbol{G}_n$ to a cycle $\boldsymbol{C}^*$. Both sequence can be obtained by Algorithm 1. $\{\boldsymbol{M}_i\}$ can convert $\boldsymbol{C}$ to $\boldsymbol{C}^*$, which can be derived by Algorithm 2. In this case, the graph sequence $\{\{\boldsymbol{A}_i\}, \{\boldsymbol{M}_i\}, \{\boldsymbol{B}_i\}^{-1}\}$ which satisfies all the conditions in Theorem 1 can transform $\boldsymbol{G}_0$ to $\boldsymbol{G}_n$. ■

Algorithm 2: Convert one cycle to another

Input: Initial cycle $\boldsymbol{G}$, final cycle $\boldsymbol{G}^*$

Output: a graph sequence $\{\boldsymbol{G}_i\}$ statisfying Lemma 2

Label the location of nodes with the same method (e.g. label each node with $1, 2, \ldots$ clockwise) in both $\boldsymbol{G}$ and $\boldsymbol{G}^*$
$i \leftarrow 1$
while $\boldsymbol{G}_i$ is not the same as $\boldsymbol{G}^*$ **do**
  if $\boldsymbol{G}(i) \neq \boldsymbol{G}^*(i)$ **then**
    //$\boldsymbol{G}(i)$ denotes the nodes at the location i in $\boldsymbol{G}$
    Swap $\boldsymbol{G}(i)$ and $\boldsymbol{G}^*(i)$ in $\boldsymbol{G}$ and get all the intermediate graphs $\boldsymbol{G}_i$
  end if
  $i \leftarrow i+1$
end while
return $\{\boldsymbol{G}_i\}$

In Lemma 2, we restrict the maximal nodal degree to be 3. If we relax this constraint a little, we will get a much easier mean to fix the cycle.

Proposition 1. *For two cycle graphs, $\boldsymbol{G}$ and $\boldsymbol{G}^*$, that have the same node set and at least 4 nodes, there must be a graph $\boldsymbol{G}'$, such that*
1) $\boldsymbol{G} \cap \boldsymbol{G}'$ and $\boldsymbol{G}' \cap \boldsymbol{G}^*$ are connected graphs;
2) $\Delta(\boldsymbol{G}') \leq 4$.

Proof: The equivalent proposition of Proposition 1 is that for a cycle (N nodes) with labels $(1, 2, \ldots, N)$ on nodes, there must be a graph $\boldsymbol{G}'$ satisfying the conditions in Proposition 1, such that the node labels in $\boldsymbol{G}^*$ is ordered. $\boldsymbol{G}'$ can be constructed by connecting node m with node $(m+1) \mod N$ and node $(m+N-1) \mod N$ if they are not connected in $\boldsymbol{G}$. Obviously, there are a cycle in $\boldsymbol{G}'$ with ordered node labels and the nodal degree of each node is at most 4. ■

This proposition greatly reduce the number of intermediate topologies than that in Lemma 2.

Though we can guarantee the feasibility of the model formulated in Section III-A, to find the optimal solution of our problem is still difficult.

Theorem 2. *The problem formulated by the **Optimization Model** in Section III-A is NP-hard.*

Proof: It is obvious that there is a edge colouring problem in the OSA-based system [7] which is an NP-hard problem, let alone the entire problem. ■

IV. Algorithm Design

As shown in last section, the problem in our work always has a feasible solution but it is difficult to find the optimal solution. Therefore, our algorithm should at least guarantee that we can always derive a feasible solution and then pursue

a better algorithm performance. In this section, we first propose topology management algorithm (TMA) to manage the topology reconfiguration in Section IV-A and briefly discuss how to implement TMA in Section IV-B.

A. Topology Management Algorithm

Based on the discussion in last section, we can easily design an algorithm to find out the intermediate topologies to manage the topology reconfiguration as the proof of Theorem 1. However, it requires too many intermediate topologies and some of the intermediate topologies can be merged:

- In Algorithm 1, some of the "while" loops can be merged. In each loop, we can use all the leaves to make pairs and connect them. When removing edges, we can determine which edge to remove for each pair of leaves one by one, in order to pursue that the most number of nodes with 2 nodal degree are generated. In real systems, these edge removings can be executed in one configuration operation.
- In Algorithm 2, some of the node pairs can also be swapped at the same time. To check if two node pairs can be swapped in the same topology, only the nodal degree constraint should be checked when edges are added.

On the other hand, we should further reduce the overflow during the topology reconfiguration:

- When the spanning tree is calculated, the edge carrying more one-hop traffic volume should be maintained to minimize the total traffic in the network.
- In intermediate topologies, wavelengths should also be adjusted according to the topology and traffic routing to transmit as much traffic as possible.

Combining all above discussions, we design a Topology Management Algorithm (TMA) to manage the topology configuration in OSA-based DCNs. This algorithm is shown in Algorithm 3. The key idea of TMA is to find the intermediate topologies by the same method of proving Theorem 1, while taking the above discussions into account to further optimize the algorithm performance.

When we transform the spanning tree to a cycle, each "while" loop in Algorithm 3 will eliminate $\lfloor \frac{L}{2} \rfloor$ leaves on the graph, where L is the number of leaves on the graph $\boldsymbol{G}_i$. Accordingly, at most $\log(N)$ intermediate topologies are required to convert a spanning tree to a cycle, where N is the number of nodes (ToRs) in the network. When we transform a cycle to another, at most $N-2$ intermediate topologies are required. Accordingly, $2\log(N)+N-2$ intermediate topologies are required in the worst case.

B. Discussion

TMA can only guarantee that there is a connected topology during each step of the topology reconfiguration but cannot guarantee that it is overflow-free. However, it is enough to serve the delay sensitive flows since most of them are mice flows. In real systems, we can enable mice flows to preempt the network resource by assigning them a high priority. As to the elephant flows, a temporary overflow will not bring too much overhead to them.

Algorithm 3: Topology Management Algorithm

Input: Initial topology $\boldsymbol{G}$, final topology $\boldsymbol{G}^*$, traffic matrix $\boldsymbol{M}$

Output: Intermediate network management schemes $\{(\boldsymbol{G}_i, \boldsymbol{R}_i, \boldsymbol{W}_i)\}$

1: Initialize $i \leftarrow 1$
2: Set weight to every edge (i,j) on $\boldsymbol{G}$ as $\boldsymbol{M}_{ij}$
3: Calculate maximal spanning tree of $\boldsymbol{G}$ which is $\boldsymbol{G}_i$
4: **while** $\boldsymbol{G}_i$ has more than 2 leaves **do**
5: Make leaf pairs to connect and get intermediate topology $\boldsymbol{G}_i$
6: Calculate routing $\boldsymbol{R}_i$ (by shortest path on the graph before adding edges) and wavelength assignment $\boldsymbol{W}_i$ (by edge coloring), $i \leftarrow i+1$
7: Remove edges as in Algorithm 1
8: **end while**
9: Connect remaining two leaves, calculate routing and wavelength assignment
10: Repeat line 4 to 9 on $\boldsymbol{G}^*$ and network management schemes are $(\boldsymbol{G}_i^*, \boldsymbol{R}_i^*, \boldsymbol{W}_i^*)$
11: Convert the cycle derived from $\boldsymbol{G}$ to the one derived from $\boldsymbol{G}^*$, say the intermediate management schemes are $(\boldsymbol{G}_i', \boldsymbol{R}_i', \boldsymbol{W}_i')$
12: **return** $\{(\boldsymbol{G}_i + \boldsymbol{G}_i' + \{\boldsymbol{G}^*\}^{-1}, \boldsymbol{R}_i + \boldsymbol{R}_i' + \{\boldsymbol{R}^*\}^{-1}, \boldsymbol{W}_i + \boldsymbol{W}_i' + \{\boldsymbol{W}^*\}^{-1})\}$

It should be noting that most of the intermediate topologies are used to synchronize two cycles in our algorithm. To reduce the number of intermediate topologies and maintain the network stability, we can construct an OSA-based network whose nodal degree constraint is at least 4, which is very realistic in DCNs. In this case, we need at most $2\log(N)+1$ intermediate topologies by using the topology management method in the proof of Proposition 1.

V. Simulation

In this section, we evaluate TMA by simulation. In Section V-A, we study the performance of TMA on reducing the traffic loss during the topology reconfiguration. As an online algorithm, the algorithm running time is an important issue. This is studied in Section V-B. All the traffic in our simulation was collected from a production datacenter hosted by IBM global services. However, the dataset only provides the incoming and outgoing traffic rates of each VM, whereas not only we need traffic matrix to calculate final topology, but also TMA requires traffic matrix as its input. To this end, we applied the Gravity model [15] to estimate the traffic matrix. All the finial topologies in each time slot is calculated by the algorithm introduced in [7].

A. TMA performance

Fig.4 shows the performance of TMA on reducing the traffic loss incurred by topology reconfiguration in a 25-hour duration. In the production datacenter, we collect the traffic

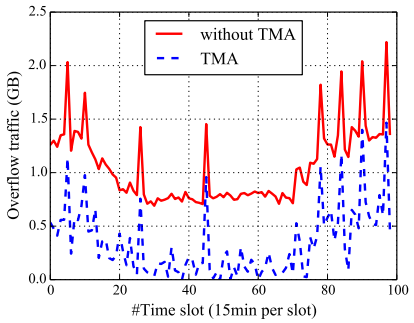
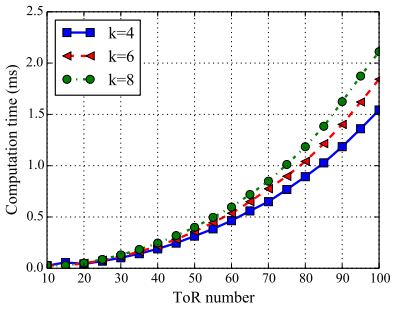
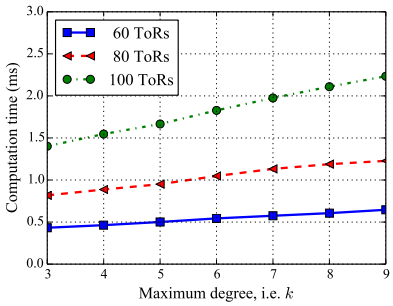


Fig. 4. Performance to reduce traffic loss



(a) Network size vs. algorithm running time



(b) Maximal node degree vs. running time

Fig. 5. Algorithm Running Time

information every 15 minutes and then update the network topology. Therefore, there are 99 time slots in this figure. The OSA-based DCN has 10778 hosts and 80 ToRs. Each ToR can connect to other 4 ToRs simultaneously and has 32 10G-wavelengths to use. In Fig.4, we can see that though the traffic loss has the same trend before and after TMA is adopted, TMA can greatly reduce the traffic loss in every topology reconfiguration. For most of the cases, TMA reduces up to 50% of the traffic loss and reconfigure the topology without traffic loss in some time slots.

B. Time complexity

Fig.5(a) and 5(b) show how the algorithm running time changes with the network size and maximal nodal degree of each node, respectively. Though the running time of TMA will increase with the network size, the computation time is still relatively small comparing with the time to reconfigure MEMS. When there are 100 ToRs in the network, TMA requires only 1.5ms to calculate all the intermediate topologies. On the other hand, it is not necessary to start topology reconfiguration till all the intermediate topologies are obtained. When the first one is obtained, we can start the reconfiguration and all the computation would have finished before the first reconfiguration completes.

The algorithm running time is also increasing with the maximal nodal degree of each node, but the increasing is very slow. As shown in Fig.5(b), the maximal nodal degree changes from 3 to 9 will ask for only 42.35% more time to calculate the intermediate topologies though there are $3\times$ edges in the network.

VI. Conclusion

We studied how to manage the topology reconfiguration in OSA-based DCNs (datacenter networks). To keep the network connectivity (guarantee the performance of delay sensitive flows) and minimize the traffic loss during the reconfiguration, we leverage intermediate topologies to realize the reconfiguration progressively. To get these intermediate topologies, we not only formulate this problem to be a programming model, but also analyze the problem feasibility and complexity. Since the problem is always feasible but NP-hard, we propose a heuristic TMA (Topology Management Algorithm) to find out intermediate topology sequence. By simulation, we find that TMA can greatly reduce the traffic loss during the topology reconfiguration by up to 50% in most of the cases with very little algorithm running time.

References

[1] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 4, pp. 63–74, Aug. 2008.

[2] 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," *SIGCOMM Comput. Commun. Rev.*, vol. 39, no. 4, pp. 51–62, Aug. 2009.

[3] 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 *Proceedings of the ACM SIGCOMM 2009*. New York, NY, USA: ACM, pp. 63–74.

[4] C. Guo, H. Wu, K. Tan, L. Shi, Y. Zhang, and S. Lu, "Dcell: A scalable and fault-tolerant network structure for data centers."

[5] J. P. Srikanth Kandula and P. Bahl, "Flyways to de-congest data center networks," in *8th ACM Workshop on Hot Topics in Networks*, 2009.

[6] W. Z. etc., "3d beamforming for wireless data centers," in *10th ACM Workshop on Hot Topics in Networks*, 2011.

[7] 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*, 2012.

[8] 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*, 2010.

[9] G. Wang, D. G. Andersen, M. Kaminsky, K. Papagiannaki, T. S. E. Ng, M. Kozuch, and M. Ryan, "c-through: Part-time optics in data centers," in *SIGCOMM 2010*, 2010.

[10] T. Benson, A. Akella, and D. A. Maltz, "Network traffic characteristics of data centers in the wild," in *Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement*, ser. IMC '10. New York, NY, USA: ACM, 2010, pp. 267–280.

[11] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan, "Data center tcp (dctcp)," in *ACM SIGCOMM '10*. New York, NY, USA: ACM, 2010, pp. 63–74.

[12] C.-Y. Hong, S. Kandula, R. Mahajan, M. Zhang, V. Gill, M. Nanduri, and R. Wattenhofer, "Achieving high utilization with software-driven wan," in *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*. New York, NY, USA: ACM, 2013, pp. 15–26.

[13] H. H. Liu, X. Wu, M. Zhang, L. Yuan, R. Wattenhofer, and D. Maltz, "zupdate: Updating data center networks with zero loss," in *ACM SIGCOMM '13*. New York, NY, USA: ACM, 2013, pp. 411–422.

[14] S. Raza, Y. Zhu, and C.-N. Chuah, "Graceful network state migrations," *IEEE/ACM Trans. Netw.*, vol. 19, no. 4, pp. 1097–1110, Aug. 2011.

[15] Y. Zhang, M. Roughan, N. Duffield, and A. Greeberg, "Fast accurate computation of large-scale ip traffic matrices from link loads," 2003.