Traffic-aware Configuration of All-Optical Data Center Networks based on Hyper-FleX-LION

Hao Yang and Zuqing Zhu, *Fellow, IEEE*

***Abstract*—Due to the advantages of optical circuit switching (OCS), all-optical data center networks (DCNs) have attracted intensive research interests recently. Hyper-FleX-LION is a highly-flexible all-optical DCN architecture that operates with the OCS based on wavelength-division multiplexing (WDM). In this work, we study how to realize traffic-aware configuration of all-optical DCNs in Hyper-FleX-LION. We formulate an integer linear programming (ILP) model for the problem to jointly optimize the configuration of Hyper-FleX-LION and the provisioning schemes of demands in it for minimizing its port usage. To ensure the practicalness of the optimization, we assume that each top-of-rack (ToR) switch can not only receive the traffic targeting to its rack but also forward traffic to other racks as an intermediate node. We also classifier traffic demands as normal and latency-sensitive ones, and set the maximum hop-count for routing latency-sensitive demands. By analyzing the complexity of the problem theoretically, we prove its $\mathcal{APX}$-hardness, *i.e.*, there does not exist a polynomial-time approximation algorithm for it unless $\mathcal{P} = \mathcal{NP}$. Then, we propose a polynomial-time heuristic JTRO based on iterative optimization to solve the problem effectively and time-efficiently. Extensive numerical simulations verify the effectiveness of our proposed algorithm. We also build a small-scale but real all-optical DCN testbed in Hyper-FleX-LION to interconnect four racks, and leverage distributed machine learning (DML) as the network services in it to demonstrate the performance of our proposal experimentally.**

***Index Terms*—Data center networks (DCNs), All-optical DCNs, Traffic-aware Configuration, Distributed machine learning.**

I. Introduction

NOWADAYS, the rapid development of 5G and data-/bandwidth-intensive network services has stimulated people to build data centers (DCs) globally [1]. As DC-related traffic has already contributed the largest portion of Internet traffic [2], the quality-of-service (QoS) demands of network services running in/across DCs put forward increasingly stringent requirements on the performance of DC networks (DCNs) [3]. For instance, large-scale distributed machine learning (DML) cannot run well in a DCN whose inter-rack bandwidth capacity is not sufficient [4], while the emerging network services such as virtual reality cannot be supported without inter-rack communications that satisfy low-latency requirements.

Therefore, the infrastructure of DCNs is facing many challenges, among which the mismatch between QoS demands of network services and capacity/latency of inter-rack networks and the ever-increasing energy consumption are mainly due to the fact that traditional DCNs solely rely on electrical packet switching (EPS) to build inter-rack networks [3]. Note that, optical circuit switching (OCS) provides larger bandwidth capacity, shorter data forwarding latency, and higher energy efficiency than EPS [5–9]. This motivated people to introduce OCS in inter-rack networks for hybrid optical/electrical DCNs (HOE-DCNs) [10, 11] and all-optical DCNs [12–14]. These new DCN architectures can address the inter-rack bottlenecks in EPS-based DCNs, especially when huge amounts of elephant flows can be generated to stress out inter-rack networks (*e.g.*, large-scale DML [4]). Specifically, recent research has shown that EPS-based DCNs built with fat-tree may have difficulty to support certain DML scenarios efficiently due to the inter-rack bottlenecks caused by DML-induced congestions [15].

Compared with HOE-DCNs, all-optical DCNs further improve the performance of inter-rack networks on bandwidth capacity, data forwarding latency, and energy efficiency, but they are also less flexible due to the relatively large switching granularity [7]. Although the issues with switching granularity can be relieved by leveraging network virtualization to groom similar network services to virtual network slices [16–19], the lack of flexibility in all-optical DCNs might still limit their capability of adapting to various inter-rack traffic patterns [14].

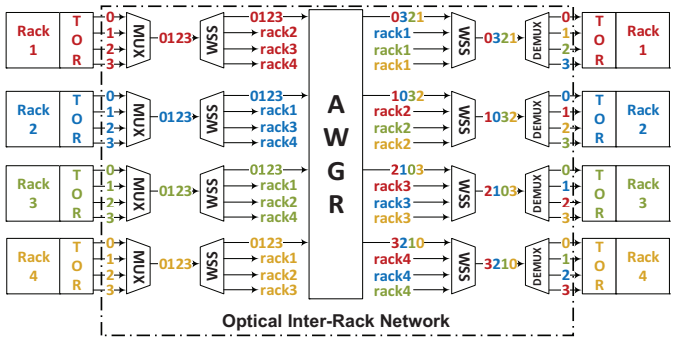
Recently, people have demonstrated that by leveraging the Flex-LIONS switch devices [20, 21], all-optical DCNs can be built with the Hyper-FleX-LION architecture to facilitate reconfigurable all-to-all optical interconnects for supporting various inter-rack traffic patterns efficiently [14]. Fig. 1 shows an illustrative example on all-optical DCNs in Hyper-FleX-LION. Here, the all-optical DCN interconnects four racks, and for the sake of explanation, we illustrate its schematic with discrete optical components in Fig. 1(a). Meanwhile, we hope to point out that the optical inter-rack network can also be built with integrated chips [21] for a more compact and cost/energy-efficient solution. There is an arrayed waveguide grating router (AWGR) working as the core of the optical inter-rack network, and the transmitting/receiving structures of each rack are respectively located at its left/right sides.

In the Hyper-FleX-LION interconnecting N racks ($N = 4$ in Fig. 1(a)), the top-of-rack (ToR) switch of each rack equips N transceivers (TRXs), each of which transmits in a unique wavelength-division multiplexing (WDM) channel and can receive the optical signal in any of the N WDM channels. Hence, Fig. 1(a) uses different numbers to label the WDM channels used by the TRXs and colors each number to indicate the source rack of each optical connection. For instance, we color the ToR switch of *Rack* 1 as red, and thus the red "0" in

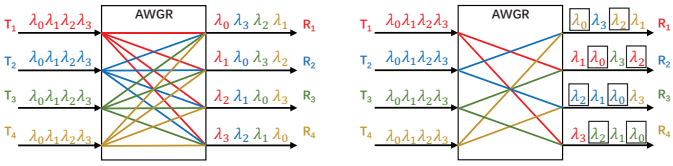
H. Yang and Z. Zhu are with the School of Information Science and Technology, University of Science and Technology of China, Hefei, Anhui 230027, China (Email: zqzhu@ieee.org).

H. Yang is also with the Department of Information Engineering, Southwest University of Science and Technology, Mianyang, Sichuan 621010, China.

Manuscript received on April 30, 2022.



(a) Network architecture



	R1	R2	R3	R4
T1	λ_0	λ_1	λ_2	λ_3
T2	λ_3	λ_0	λ_1	λ_2
T3	λ_2	λ_3	λ_0	λ_1
T4	λ_1	λ_2	λ_3	λ_0

	R1	R2	R3	R4
T1	-	$\lambda_0\lambda_1\lambda_2$	-	λ_3
T2	λ_3	-	$\lambda_0\lambda_1\lambda_2$	-
T3	-	λ_3	-	$\lambda_0\lambda_1\lambda_2$
T4	$\lambda_0\lambda_1\lambda_2$	-	λ_3	-

(b) Configurations for two different traffic matrices

Fig. 1. All-optical DCN in Hyper-FleX-LIONS that interconnects four racks, MUX/DEMUX: wavelength multiplexer/demultiplexer, WSS: wavelength selective switch, AWGR: arrayed waveguide grating router.

Fig. 1(a) represents the optical connection from the first TRX on the ToR switch of *Rack* 1. In the transmitting structure, all the outputs of a ToR switch are multiplexed by a WDM multiplexer (MUX) before being sent to a $1\times N$ wavelength selective switch (WSS). One of the WSS' outputs goes to the AWGR, while the other $N-1$ outputs are directly connected to the $N-1$ WSS' that belong to the receiving structures of other racks, respectively. For example, the second output of the WSS in the transmitting structure of *Rack* 1 goes to the second input of the WSS in the receiving structure of *Rack* 2. Then, in the receiving structure of each rack, the optical signals from the WSS' in the transmitting structures of other racks and the AWGR are distributed to the TRXs of the rack's ToR switch with a WSS and a WDM demultiplexer (DEMUX).

Hence, we can get various optical interconnections among the racks by adjusting the WSS' in the transmitting/receiving structures and leveraging the wavelength switching capability of the AWGR. For instance, Fig. 1(b) shows two configurations of the Hyper-FleX-LION in Fig. 1(a) and the traffic matrices that they support. The left configuration is just the one in Fig. 1(a), where each WSS in the transmitting structures sends all the WDM signals from the TRXs of a ToR switch to the AWGR, which demultiplexes the WDM signals at each of its input ports and switches them to its outputs according to their wavelengths. Specifically, as a passive wavelength switching device, AWGR operates based on a fixed wavelength mapping between its inputs and outputs, *e.g.*, λ_0 and λ_1 entering at *Input* 1 will go to *Outputs* 1 and 2, respectively. Therefore, the left configuration supports the traffic matrix whose elements are all identical, as shown in Fig. 1(b). On the other hand, the right configuration in Fig. 1(b) makes each WSS in the transmitting structures only send two WDM signals from the TRXs of a ToR switch to the AWGR, while the remaining two WDM signals go to one WSS in the receiving structures directly (marked with boxes). For example, λ_0 and λ_2 from *Rack* 1 are sent to *Rack* 2 directly (*i.e.*, bypassing the AWGR).

Although existing studies have already addressed the hardware [20, 21], architecture [14], and applications [22] of all-optical DCNs in Hyper-FleX-LION, the problem of how to explore its flexibility to efficiently adapt to various patterns of inter-rack traffic has not been studied yet. This problem is relevant because if we do not leverage a traffic-aware algorithm to configure the inter-rack network of a DCN properly, the link utilization there could be severely unbalanced [23], which not only degrades the QoS of active network services but also makes the DCN's operation less cost-efficient. Meanwhile, due to the unique architecture of Hyper-FleX-LION, the problem is also different from its counterparts that have been studied for HOE-DCNs [24–26] and other all-optical DCNs [27, 28].

In this work, we study how to realize traffic-aware configuration of all-optical DCNs in Hyper-FleX-LION. We first formulate an integer linear programming (ILP) model for the problem, which jointly optimizes the configuration of Hyper-FleX-LION and the provisioning schemes of demands in an arbitrary traffic matrix to minimize the port usage in the all-optical DCN for high cost-efficiency. To ensure the practicalness of the optimization, we assume that each ToR switch can not only receive the traffic targeting to it but also forward traffic to other racks as an intermediate node. Meanwhile, we classifier demands as normal and latency-sensitive ones, where the latter are generated by the network services such as web search [29]. Note that, when multi-hop routing is enabled, additional latency can be caused by the repeated optical-to-electrical-to-optical (O/E/O) conversions and packet processing in ToR switches. Hence, we restrict the maximum number of hops that latency-sensitive demands can be routed.

Then, we analyze the time complexity of the problem theoretically, and prove its $\mathcal{APX}$-hardness, *i.e.*, there does not exist a polynomial-time approximation algorithm for it unless $\mathcal{P}=\mathcal{NP}$. Hence, we propose a polynomial-time heuristic (namely, JTRO) to solve the problem effectively and time-efficiently. Extensive numerical simulations confirm the performance of our algorithm and show that it can outperform existing benchmarks. Finally, we build a small-scale but real all-optical DCN testbed in Hyper-FleX-LION to interconnect four racks, and leverage DML as the network services in it to demonstrate the effectiveness of our proposal experimentally.

The rest of the paper is organized as follows. We first conduct a brief survey on the related work in Section II. Then, in Section III, we provide the problem description, formulate the ILP model to solve it exactly, and analyze the problem's complexity. Section IV discusses our algorithm design for the heuristic. The numerical simulations and experimental demonstrations are presented in Sections V and VI, respectively. Finally, Section VII summarizes the paper.

II. Related Work

The idea of introducing OCS in inter-rack networks of DCNs for HOE-DCNs and all-optical DCNs has already been studied for more than a decade. In the early days, Helios [10] and c-Through [11] were proposed as HOE-DCNs and OSA [27] was developed for all-optical DCNs. Since then, many new architectures have been proposed [3, 12–14]. Based on these architectures, people have considered how to explore their reconfigurability to adapt to various patterns of inter-rack traffic, and designed a number of algorithms [24–28].

In [27], the authors developed traffic-aware configuration algorithms for all-optical DCNs in OSA. They formulated an ILP model and proposed a heuristic that solves the problem by computing the inter-rack topology, routing of demands, and wavelength assignments in three sequential steps. However, as the steps are handled independently in the heuristic, they might not fully explore the flexibility of OSA, and moreover, the authors did not consider latency-sensitive traffic in the network model. Zhao *et al.* [25] leveraged a hierarchical approach to schedule the inter-rack traffic in an HOE-DCN where OCS is realized with an optical cross-connect (OXC). Nevertheless, it is known that OXC can usually provide 1-to-1 connectivity, which will greatly restrict the flexibility of OCS for traffic routing [30]. The studies in [26, 28] considered multi-hop traffic routing in all-optical DCNs and HOE-DCNs, respectively, and proposed heuristics for the problems. Similar as the approach used in [27], the heuristics also planned the multi-hop traffic routing in independent steps sequentially. In [24], we proposed an adaptive algorithm based on deep reinforcement learning (DRL) to schedule the traffic demands in an HOE-DCN, where the OCS part was based on an OXC.

Previous studies also addressed the provisioning of specific network services in HOE-DCNs and all-optical DCNs [22, 31, 32]. We designed a knowledge-defined orchestration scheme based on DRL in [31] to optimize the provisioning of Hadoop jobs in an HOE-DCN, for reducing job completion time. Wang *et al.* [32] showed that the DML jobs in an HOE-DCN could be accelerated when its OCS part was reconfigured adaptively. In [22], we experimentally demonstrated that an all-optical DCN in Hyper-FleX-LION could accelerate DML jobs better than an HOE-DCN based on OXC.

Finally, to the best of our knowledge, all the existing studies in this area did not consider the algorithm design for Hyper-FleX-LION, which makes the algorithms proposed in them not suitable for the problem considered in this work. Although our work in [22] did consider the service provisioning in Hyper-FleX-LION, it was mainly for proof-of-concept demonstration. We did not address algorithm design in it either, and multi-hop routing in inter-rack networks was not considered.

III. Problem Description

In this section, we first explain the network model of traffic-aware configuration for Hyper-FleX-LION, and then formulate an ILP model to describe the optimization for it.

A. Network Model

In this work, we consider an all-optical DCN in Hyper-FleX-LION that interconnects N racks and operates with the principle shown in Fig. 1(a). Hence, each ToR switch equips N TRXs, which transmit in N adjacent WDM channels (denoted as $\{\lambda_i,\ i \in [1, N]\}$). The capacity of the i-th TRX on a ToR switch is denoted as C. Similar as previous studies in [25–28], we model the traffic among racks as an $N \times N$ traffic matrix $\mathbf{D}$, where each element represents the bandwidth demand that needs to be satisfied between a rack pair. To ensure that the network model is generic, we do not specify the unit of C and traffic amounts in $\mathbf{D}$ here. As the traffic matrix $\mathbf{D}$ contains the traffic from various network services, we assume that it can be further divided into two traffic matrices as $\mathbf{D} = \mathbf{D}_{lt} + \mathbf{D}_{ls}$, where $\mathbf{D}_{lt}$ denotes the matrix of latency-tolerant traffic (*e.g.*, for Hadoop file streams [33]) while $\mathbf{D}_{ls}$ is the latency-sensitive traffic matrix (*e.g.*, for virtual reality and online games [34]).

To fully utilize the capacity of each TRX, we assume that multi-hop routing is allowed in the inter-rack network, *i.e.*, each ToR switch can not only receive the traffic targeting to it but also forward traffic to other racks as an intermediate node. Note that, as the multi-hop routing makes a flow go through repeated O/E/O conversions and electrical processing in ToR switches, additional latency can be induced to prolong its flow completion time (FCT). Hence, we restrict the hop-count of latency-sensitive traffic in $\mathbf{D}_{ls}$ to not exceed H.

The operation principle of Hyper-FleX-LION in Fig. 1(a) determines that any WDM channel from a source ToR switch can potentially reach any destination ToR switch (including itself). However, there are also a few constraints. First, one TRX on a source ToR switch can only connect to one TRX on a destination ToR switch. Second, the TRXs connecting to a same destination ToR switch should use different WDM channels. In other words, we can model the inter-rack network of Hyper-FleX-LION as a graph $G(V, E)$, where each vertex v in set V represents a ToR switch, and each directed edge e in set E denotes a used unidirectional transmission from the optical transmitter (TX) in one TRX to the optical receiver (RX) in another TRX and is a partially determined element. Specifically, the source and color (*i.e.*, the WDM channel) of each directed edge e is predetermined by the corresponding TRX, while its destination should be optimized in the traffic-aware configuration. Hence, the traffic-aware configuration can be viewed as a problem of finalizing edges in E to obtain a proper graph $G(V, E)$ and routing traffic in $G(V, E)$ to satisfy all the demands in $\mathbf{D}$, such that the total number of used ports is minimized[1]. Here, each "used port" refers to a unidirectional transmission that carries traffic between a TX/RX pair. We select such an optimization objective for two reasons. First, the number of used ports in a DCN is an important metric for evaluating its operational complexity and energy consumption (*i.e.*, the operational expenses (OPEX)) [3, 23]. Second, if an algorithm that can serve the same amount of traffic demands with a lower port usage enables a DCN operator to provision more network services with the same capital expenses (CAPEX), *i.e.*, bringing in more revenue.

[1]We assume that each element $d_{v,v} \in \mathbf{D}$ for the loop-back traffic of ToR switch v is 0. This is because the loop-back traffic can be forwarded internally by ToR switch v without going through the Hyper-FleX-LION.

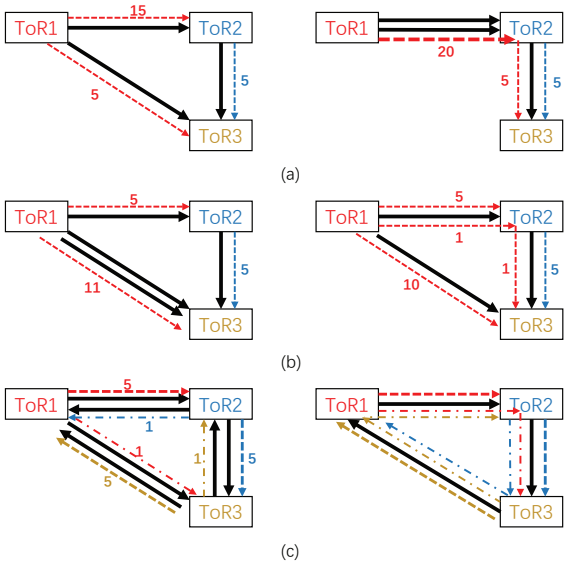


Fig. 2. Examples on correlation between topology design and traffic routing.

B. Topology Design and Traffic Routing

The traffic-aware configuration basically includes two subproblems, *i.e.*, the topology design and traffic routing. Here, the topology design refers to setting up/tearing down connections among the racks to adapt to the traffic matrix, by configuring the optical components in Hyper-FleX-LION. The traffic routing is to plan forwarding paths for each demand by labeling packets and configuring the flow tables in ToR switches. Note that, although Hyper-FleX-LION uses OCS, each ToR switch operates based on EPS. Therefore, a ToR switch can split the traffic of a demand to multiple routing paths by labeling the demand's packets accordingly and forwarding them to the output ports corresponding to the paths [35, 36], achieving traffic splitting on a per-packet basis. Hence, our network model considers not only the multi-hop routing through intermediate rack(s) but also multi-path routing.

We hope to point out that due to the flexibility of Hyper-FleX-LION, the subproblems are more correlated than those considered in existing studies [25–28]. Hence, they should be tackled jointly, which can also be justified with the examples in Fig. 2. The example in Fig. 2(a) explains how an improper topology design can make traffic routing infeasible. Here, there are three demands among the ToR switches, as 1→2, 1→3, and 2→3 for capacities of 15, 5, and 5, respectively (marked as dashed arrows). The capacity of each TRX is 10, and each black arrow denotes a unidirectional transmission[2] between a TX/RX pair, which delivers a capacity of 10. Then, with the left configuration in Fig. 2(a), the demand of 1→2 cannot be satisfied, while all the demands can be served in the right configuration in Fig. 2(a). The two configurations send traffic though the same number of unidirectional transmissions between TRXs (*i.e.*, their port usages are both 3), but their supports to the demands are different.

[2]Note that, real-world TRXs do not support unidirectional transmissions. In Fig. 2, we omit irrelevant unidirectional transmissions for clearer illustration, but they are actually not turned off.

Fig. 2(b) explains why proper traffic routing can help to optimize the topology design. This time, the demands are still for 1→2, 1→3, and 2→3, but their capacities become 5, 11, and 5, respectively. Then, to support the traffic routing in the left configuration, we need to use three TXs on *ToR Switch* 1, while the traffic routing in the right configuration only requires two TXs. Hence, the proper traffic routing in the right configuration helps to save one TX/RX pair over that in the left one. Fig. 2(c) explains why multi-hop routing should be considered. Here, we assume that all the racks are involved in a DML whose architecture is the *Ring-AllReduce* [37]. The traffic demands form a ring, *i.e.*, each ToR switch sends/receives data to/from its next/previous ToR switch with capacities of 5 and 1, respectively. If multi-hop routing is not allowed, the port usage of the left configuration needs to be 6 (the scheme in [22]), while with multi-hop routing, the right configuration only needs 3 TX/RX pairs.

C. Integer Linear Programming Model

In order to solve the traffic-aware configuration for Hyper-FleX-LION exactly, we formulate an ILP model. Table I lists the parameters and variables of the ILP.

TABLE I
PARAMETERS AND VARIABLES OF ILP

Parameters	
N	Number of racks interconnected by Hyper-FleX-LION.
V	Set of ToR switches ($\|V\| = N$).
$\mathbf{D}_{ls}$	Matrix of latency-sensitive traffic ($N \times N$).
$\mathbf{D}_{lt}$	Matrix of latency-tolerant traffic ($N \times N$).
$d_{v,u}$	Traffic demand from v to u ($v, u \in V$).
H	Maximum hop-count for latency-sensitive traffic.
C	Bandwidth capacity of each TRX on a ToR switch.
M	A large positive constant.
Variables	
$L^i_{(u,v)}$	Binary variable that equals 1 if ToR switch u uses the i-th TRX to set up an connection to ToR switch v.
$f^{(u,v)}_{s,t}$	Non-negative integer variable that represents the amount of traffic, which is for $s \to t$ ($\{s, t \in V, s \neq t\}$) and gets routed over an optical connection (u, v) ($u, v \in V$).
$x^{(u,v)}_{s,t}$	Binary variable that equals 1 if the demand for $s \to t$ does not use an optical connection (u, v).
$T^v_{s,t}$	Non-negative integer variable that denotes the maximum hop-count for routing the demand for $s \to t$ to reach ToR switch v ($v \in V$).

Objective:

The objective is to minimize the port usage (*i.e.*, the number of used TX/RX pairs) in the Hyper-FleX-LION.

$$\text{Minimize} \quad \sum_{u \in V} \sum_{v \in V} \sum_{i=1}^{N} L^i_{(u,v)}. \tag{1}$$

Constraints:

1) Constraints for Topology Design:

$$\sum_{v \in V} L^i_{(u,v)} \leq 1, \quad \forall u \in V, \ \forall i \in [1, N]. \tag{2}$$

Eq. (2) ensures that the optical signal from each TRX on a ToR switch can only be switched by the WSS' and AWGR in the Hyper-FleX-LION to at most one ToR switch.

$$\sum_{u\in V} L^i_{(u,v)} \le 1, \quad \forall v \in V,\ \forall i \in [1,N]. \tag{3}$$

Eq. (3) ensures that all the optical signals forwarded by the WSS' and AWGR in the Hyper-FleX-LION to one ToR switch can only use different WDM channels.

2) Constraints for Traffic Routing:

$$\sum_{v\in V\setminus u} f^{(u,v)}_{s,t} - \sum_{v'\in V\setminus u} f^{(v',u)}_{s,t} = \begin{cases} d_{s,t}, & u = s, \\ -d_{s,t}, & u = t, \\ 0, & \text{otherwise}, \end{cases} \quad \forall u \in V,\ \{d_{s,t} \in \mathbf{D} = \mathbf{D}_{ls} + \mathbf{D}_{lt}, s \ne t\}. \tag{4}$$

Eq. (4) is the flow conservation constraint for multi-hop traffic routing, to ensure that all the demands in $\mathbf{D}$ are served correctly, and it supports multi-path routing for each demand.

$$\sum_{\{s,t\in V, s\ne t\}} f^{(u,v)}_{s,t} \le \sum_{i=1}^{N} L^i_{(u,v)} \cdot C, \quad \forall u,v \in V. \tag{5}$$

Eq. (5) ensures that the total amount of traffic, which is routed over the optical connection (u,v), cannot exceed the available capacity provided by the Hyper-FleX-LION for (u,v).

$$\begin{cases} M \cdot \left(1 - x^{(u,v)}_{s,t}\right) \ge f^{(u,v)}_{s,t} \ge 1 - x^{(u,v)}_{s,t}, & \forall u,v \in V, \\ T^v_{s,t} \ge T^u_{s,t} - M \cdot x^{(u,v)}_{s,t} + 1, & \forall u,v \in V, \\ T^v_{s,t} = 0, & v = s, \\ T^v_{s,t} \le H, & v = t, \\ & \forall \{d_{s,t} \in \mathbf{D}_{ls}, s \ne t\}. \end{cases} \tag{6}$$

Eq. 6 ensures that latency-sensitive demands will not be routed for more than H hops.

D. Complexity Analysis

We then analyze the complexity of the traffic-aware configuration for Hyper-FleX-LION. First of all, we set $H = +\infty$ to relax the constraint in Eq. (6) and obtain a special instance of the original problem. Hence, if we can prove that the special instance is $\mathcal{APX}$-hard, *i.e.*, there does not exist a polynomial-time approximation algorithm for it unless $\mathcal{P} = \mathcal{NP}$, the original problem is also $\mathcal{APX}$-hard [38]. Next, to describe our analysis clearly, we define the following two questions:

Definition 1. *For a traffic matrix* $\mathbf{D}$ *and a Hyper-FleX-LION interconnecting* N *racks, the* ***Accommodation Problem*** *is defined as: Whether a topology design can be found to allow the Hyper-FleX-LION to accommodate all the demands in* $\mathbf{D}$*?*

Definition 2. *For a traffic matrix* D *and a Hyper-FleX-LION with* N *racks,* ***Traffic-aware Configuration Problem*** *is defined as the optimization in the ILP where Eq.* (6) *has been relaxed,* i.e.*, a special instance of the original problem.*

We can verify that the **Accommodation Problem** is the decision problem of the **Traffic-aware Configuration Problem**.

Theorem 1. *The* ***Accommodation Problem*** *is* $\mathcal{NP}$*-complete.*

Proof: We prove the $\mathcal{NP}$-completeness of the **Accommodation Problem** by the restriction method [38], *i.e.*, constructing a special instance for it and prove that the special instance is a general case of a known $\mathcal{NP}$-complete problem. Specifically, we build the special instance by first considering a special traffic matrix $\mathbf{D}$, which is generated as follows.

- Assume that there are only two source ToR switches in $\mathbf{D}$, denoted as s_1 and s_2.
- Add $(N-1)$ random positive demands in $\mathbf{D}$ for the traffic from s_1 to each ToR switch $v \in V \setminus s_1$.
- Add $(N-1)$ random positive demands in $\mathbf{D}$ for the traffic from s_2 to each ToR switch $v \in V \setminus s_2$.
- Randomly select a ToR switch t_1 from $V \setminus s_1$, and increase the demand for $s_1 \to t_1$ to be greater than the capacity of a TRX (*i.e.*, C) if it is not.
- Randomly select a ToR switch t_2 from $V \setminus s_2$, and increase the demand for $s_2 \to t_2$ to be greater than the capacity of a TRX (*i.e.*, C) if it is not.

Now, we obtain a traffic matrix $\mathbf{D}$ with $2\cdot(N-1)$ demands, and can verify that no matter how the Hyper-FleX-LION is configured, at least two demands (*e.g.*, d_{s_1,t_1} and d_{s_2,t_2}) cannot be served with single-hop routing. Then, we further restrict the special instance by assuming that the topology has been designed as $G = (V,E)$ and all the demands in $\mathbf{D}$ except for d_{s_1,t_1} and d_{s_2,t_2} have been served in it. Hence, the special instance of the **Accommodation Problem** becomes: For a directed graph $G = (V,E)$, is it possible to find suitable routing paths for two traffic demands d_{s_1,t_1} and d_{s_2,t_2}? This is equivalent to the problem of two-commodity integral flow in directed graphs (D2CIF), which is known to be $\mathcal{NP}$-complete [39]. Therefore, we prove that the **Accommodation Problem** is also $\mathcal{NP}$-complete. ■

Then, based on *Theorem* 1, we can get *Theorem* 2 below.

Theorem 2. *The* ***Traffic-aware Configuration Problem*** *is* $\mathcal{APX}$*-hard.*

Proof: We prove that the **Traffic-aware Configuration Problem** is $\mathcal{APX}$-hard with contradiction. We first assume that there exists a polynomial-time approximation algorithm $\mathbf{A}$, which can find a k-approximation solution for the **Traffic-aware Configuration Problem**. In other words, $\mathbf{A}$ can find a edge set E^A to make $G = (V, E^A)$ accommodate the traffic matrix $\mathbf{D}$ in polynomial time, which guarantees that $\frac{|E^A|}{|E^*|} \le k$, where $|E^A|$ is the number of edges in E^A, $|E^*|$ is the number of edges in the optimal solution E^*, and $k \ge 1$.

Then, we consider a general instance of the **Traffic-aware Configuration Problem**, and denote it as I. The analysis above suggests that if the optimal solution E^* exists for I, we can use $\mathbf{A}$ to find an approximation solution E^A for I, which is also a feasible solution. Otherwise, if a feasible solution does not exists for I, we can also use $\mathbf{A}$ to get the right conclusion as $\mathbf{A}$ will not provide a feasible solution in this case either. Therefore, we can use $\mathbf{A}$ to check whether there exists a feasible solution for I (*i.e.*, solving the decision problem of the **Traffic-aware Configuration Problem**) in polynomial time, because $\mathbf{A}$ runs in polynomial time. However, this contradicts to *Theorem* 1 (*i.e.*, the **Accommodation Problem** is $\mathcal{NP}$-

complete) unless $\mathcal{P} = \mathcal{NP}$. Hence, there does not exist a polynomial-time approximation algorithm **A** for the **Traffic-aware Configuration Problem**, and it is $\mathcal{APX}$-hard. ■

Finally, as *Theorem* 2 confirms that a special case of the original problem is $\mathcal{APX}$-hard, we prove that the original problem (*i.e.*, the ILP in Section III-C) is also $\mathcal{APX}$-hard.

IV. Algorithm Design

As Section III-D suggests that the traffic-aware configuration for Hyper-FleX-LION is $\mathcal{APX}$-hard, we do not try to design a polynomial-time approximation algorithm but will propose an effective heuristic for it in this section, which is named as joint topology design and traffic routing optimization (JTRO). Different from the ILP, which cannot obtain a feasible solution if the traffic load in $\mathbf{D}$ is too high, JTRO is a heuristic and thus can serve part of the demands in $\mathbf{D}$ in such a case.

A. Algorithm based on Heuristic Iterative Optimization

To solve the traffic-aware configuration for Hyper-FleX-LION, the exact way is to enumerate all the feasible inter-rack topologies, find the traffic routing scheme in each topology for serving $\mathbf{D}$ such that the port usage is minimized, and select the topology and the routing scheme in it, which provides the least port usage among all the cases. Although the exhaustive search runs in exponential time, it provides the idea for our algorithm design. We first introduce a theorem.

Theorem 3. *For any topology G that can satisfy $\mathbf{D}$ but does not use all the TRXs in Hyper-FleX-LION, we can find another topology G', which uses all the TRXs and still satisfies $\mathbf{D}$, such that we can remove unnecessary edges in G' to get a topology whose size*[3] *is the same as or smaller than that of G.*

Proof: For an arbitrary topology G that can satisfy $\mathbf{D}$ but does not use all the TRXs in a Hyper-FleX-LION, we arbitrarily add a new edge e in it to get a larger topology G_1. Then, we can easily verify that the best traffic routing scheme in G_1 will not use more TRXs than those used by the best traffic routing scheme in G. In other words, the new edge e might make the best traffic routing of $\mathbf{D}$ uses less TRXs, but in the worst case, the numbers of TRXs used by the best traffic routing schemes in G and G_1 should be the same. Therefore, we can keep adding new edges in G until obtaining another topology G' that uses all the TRXs in the Hyper-FleX-LION, and use the analysis above repeatedly to prove *Theorem* 3. ■

Theorem 3 tells us that to solve the traffic-aware configuration for Hyper-FleX-LION, we only need to enumerate all the largest topologies, which uses all the TRXs (*i.e.*, $|E| = N^2$). Although this greatly reduces the number of inter-rack topologies that we need to enumerate, the exhaustive search still cannot be done in polynomial time. Hence, we propose a heuristic based on iterative optimization.

Algorithm 1 describes the overall procedure of our proposed heuristic. *Lines* 1-2 are for the initialization, where we assign a unique index i to denote each ToR switch in V as v_i and initialize the available numbers of input/output ports on each $v_i \in V$ as N. Here, N_i^-/N_i^+ denote the available numbers of input/output ports on ToR switch v_i, respectively. Then, in *Lines* 3-18, we greedily construct a topology $G(V, E)$ that uses all the TRXs (*i.e.*, $|E| = N^2$) and can satisfy the traffic matrix $\mathbf{D}$. In each iteration of the for-loop covering *Lines* 3-13, we determine the edges that go out of a ToR switch. For each ToR switch v_i, we first sort the demands from it in descending order of their bandwidth (*Line* 4), and then set up a directed edge for each demand in turn until all the output ports of the ToR

[3]For simplicity, we define the size of an inter-rack topology $G(V, E)$ as the number of directed edges in it (*i.e.*, $|E|$ or the used TX/RX pairs).

Algorithm 1: Overall Procedure

Input: N, $\mathbf{D}_{ls}$, $\mathbf{D}_{lt}$, C, K, γ.
Output: $G = (V, E)$, W_E.

1 $V = \{v_i,\ i \in [1, N]\}$, $E = \emptyset$, $\mathbf{D} = \mathbf{D}_{ls} + \mathbf{D}_{lt}$;
2 $\{N_i^+ = N,\ N_i^- = N,\ \forall i \in [1, N]\}$;
3 **for** *each* $i \in [1, N]$ **do**
4 sort all demands from v_i ($d_{v_i,t} \in \mathbf{D}$) in descending order, and store their destinations in set V';
5 **while** *($N_i^+ \neq 0$) AND (there is $d_{v_i,v_j} > 0$)* **do**
6 **for** *each* $v_j \in V'$ **do**
7 **if** *($d_{v_i,v_j} > 0$) AND ($N_j^- > 0$)* **then**
8 $N_i^+ = N_i^+ - 1$, $N_j^- = N_j^- - 1$;
9 $d_{v_i,v_j} = d_{v_i,v_j} - C$, $(v_i, v_j) \rightarrow E$;
10 **end**
11 **end**
12 **end**
13 **end**
14 **for** *each* $i \in [1, N]$ **do**
15 **if** $N_i^+ \neq 0$ **then**
16 add feasible edges randomly until $N_i^+ = 0$, and insert the newly-added edges in E;
17 **end**
18 **end**
19 $\tilde{g} = \sum_{i=1}^{N} \left\lceil \frac{\sum_{j=1}^{N} d_{v_i,v_j}}{C} \right\rceil$, $G^* = G(V, E)$;
20 **while** $K > 0$ **do**
21 $K = K - 1$;
22 apply *Algorithm* 2 to G to find minimal subgraph $G' = (V', E')$ and remaining bandwidth set $R_{E'}$;
23 $G^* = \min(G^*, G')$;
24 **if** *($\frac{|E'|}{\tilde{g}} \leq 1 + \gamma$) OR ($K = 1$)* **then**
25 use available wavelengths to color edges in G^*;
26 **if** *a feasible assignment W_E can be got* **then**
27 **break**;
28 **end**
29 **end**
30 find edge $e^* = (s^*, t^*)$ with the smallest R_{e^*} in E';
31 find edge $e_1 = (s^*, t_1)$ with the largest R_{e_1} in E';
32 find edge $e_2 = (s_1, t^*)$ with the largest R_{e_2} in E';
33 remove e_1 and e_2 from E;
34 insert (s^*, t^*) and (s_1, t_1) into E;
35 **end**
36 $G = G^*$;

switch have been used (*Lines* 5-12). The available numbers of input/output ports on the ToR switches ($\{N_i^+, N_i^- = N, \forall i \in [1, N]\}$) are also updated in the process (*Line* 8).

Lines 14-18 establish directed edges randomly until all the TRXs have been used. Then, we calculate the lower-limit of the number of TRXs that should be used to support $\mathbf{D}$ as $\tilde{g}$, and set the current optimal topology G^* as the $G(V, E)$ obtained until now (*Line* 19). Next, *Lines* 20-35 describe the procedure for iterative optimization, in which if the solution obtained in an iteration satisfies the expected approximation ratio γ or the number of iterations reaches the upper-limit K, we will stop the iteration and output the current optimal solution.

Algorithm 2: Find Minimal Subgraph G'

Input: $G = (V, E)$, $\mathbf{D}_{ls}$, $\mathbf{D}_{lt}$, C, H, δ, η.

1 $R_E = \{R_e = C,\ \forall e \in E\}$;
2 input $\{G, \mathbf{D}_{ls}, C, R_E, H\}$ to *Algorithm* 3 for R_E;
3 input $\{G, \mathbf{D}_{lt}, C, R_E, -1\}$ to *Algorithm* 3 for R_E;
4 **if** *no feasible solution* **then**
5 **return**(G, R_E);
6 **end**
7 **while** $E \neq \emptyset$ **do**
8 remove all the edges in $\tilde{E} = \{e : e \in E, R_e \geq \delta \cdot C\}$ from E, and update $G(V, E)$;
9 $R_E = \{R_e = C,\ \forall e \in E\}$;
10 input $\{G, \mathbf{D}_{ls}, C, R_E, H\}$ to *Algorithm* 3 for R_E;
11 input $\{G, \mathbf{D}_{lt}, C, R_E, -1\}$ to *Algorithm* 3 for R_E;
12 **if** *no feasible solution* **then**
13 **if** $\delta < 1.0$ **then**
14 add edges in $\tilde{E}$ back to E;
15 $\delta = \delta + \eta$;
16 **else**
17 **break**;
18 **end**
19 **end**
20 **end**
21 **return**(G, R_E);

Specifically, in each iteration, we first apply *Algorithm* 2 to G to find the minimal subgraph G' (*Line* 22) and then check whether it can satisfy the condition to stop the iteration (*Lines* 24-29). Here, *Line* 25 tries to assign available wavelengths to color the edges in the current optimal topology G^* (*i.e.*, determining the right TRX that should be used for each directed edge in G^*). Note that, this is a typical edge coloring problem [40], which can be easily solved by many time-efficient algorithms [41]. *Lines* 30-34 represents our heuristic approach for optimizing the edges in E, *i.e.*, replacing two edges with two better ones. Specifically, we believe that the edge e^* with the smallest remaining bandwidth (R_{e*}) in the minimal subgraph G' can indicate that its source/destination ToR switches are essential for traffic routing. Therefore, *Lines* 31-32 find the edges with the most remaining bandwidth on each of these two ToR switches in G', and *Lines* 33-34 replace their counterparts in G with two better ones to increase the bandwidth between the two ToR switches. Fig. 3 illustrates the procedure in *Lines* 30-34. Note that, as *Lines* 33-34 only replace two edges in E, the total number of edges in $G(V, E)$ stays unchanged (*i.e.*, we still have $|E| = N^2$). Finally, we proceed to the next iteration with the newly-obtained G.

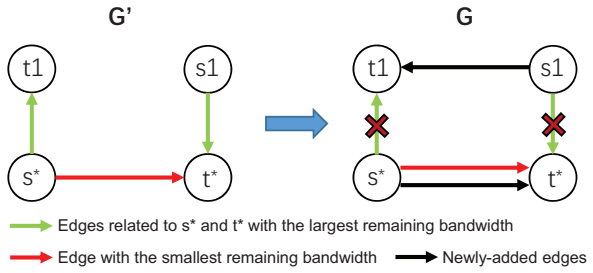


Fig. 3. Heuristic approach for optimizing edge establishment.

B. Sub-procedures for Traffic Routing

We design *Algorithm* 2 to serve all the demands in $\mathbf{D}$ in an inter-rack topology $G(V, E)$ with $|E| = N^2$, such that the TRXs are used as few as possible. *Line* 1 is for the initialization, where the remaining bandwidth on each edge $e \in E$ is initialized as $R_e = C$. In *Lines* 2-3, we try to use *Algorithm* 3 to find the routing schemes for latency-sensitive traffic matrix $\mathbf{D}_{ls}$ and latency-tolerant traffic matrix $\mathbf{D}_{lt}$ in sequence, and R_E is updated sequentially too. If a feasible solution can be found, the while-loop covering *Lines* 7-20 tries to remove unnecessary edges in iterations. In each iteration, we first remove all the edges in set $\tilde{E} = \{e : e \in E, R_e \geq \delta \cdot C\}$ from E, where $\delta \in (0, 1)$ is a preset coefficient to control the edge removal process, and update $G(V, E)$ accordingly (*Lines* 8-9). *Lines* 10-11 try to use *Algorithm* 3 to find the routing schemes for $\mathbf{D}_{ls}$ and $\mathbf{D}_{lt}$ in the updated $G(V, E)$ in sequence. If a feasible solution cannot be found and we still have $\delta < 1$, we add the edges in $\tilde{E}$ back to E and increase δ by a preset step of η (*Lines* 12-15). If we cannot find more edges to remove from E, we will terminate the while-loop and return the current G (*i.e.*, the minimal subgraph G' for serving $\mathbf{D}$) and the corresponding R_E to *Algorithm* 1 (*Line* 21).

The basic idea of *Algorithm* 2 is to first remove the edge that has the most available bandwidth. Therefore, we deliberately spread out the demands in a traffic matrix when computing their routing paths with *Algorithm* 3. Specifically, we take the bandwidth usage on each edge as its weight, and find the least-weighted paths for the demands. In *Algorithm* 3, we first initialize the weight of each edge $e \in E$ based on its current bandwidth usage (*Line* 1). Then, the while-loop that covers *Lines* 2-21 finds routing schemes for the demands in traffic matrix $\mathbf{D}$ until all the demands there have been served. If the demand is for latency-sensitive traffic, we use *Algorithm* 4 to find the least weighted path $\mathcal{P}$ for it (*Line* 5). Otherwise, we use the Dijkstra's Algorithm to find the least weighted path $\mathcal{P}$ for it (*Line* 7). Next, if $\mathcal{P}$ can be found, *Lines* 12-20 serve the whole or partial demand of $d_{s,t}$ with it and update the network status accordingly. Finally, after all the demands in $\mathbf{D}$ having been served, we return the remaining bandwidth on each edge (*i.e.*, R_E) to *Algorithm* 2.

In *Algorithm* 4, we leverage the breadth-first search (BFS) method [42] to obtain the least-weighted routing path for a

Algorithm 3: Find Traffic Routing Scheme

Input: $G=(V,E)$, $\mathbf{D}$, C, R_E, H.

1 assign a weight $l_e = C - R_e + 1$ to each $e \in E$;
2 **while** *there are still non-zero elements in* $\mathbf{D}$ **do**
3 select a non-zero element $d_{s,t} \in \mathbf{D}$;
4 **if** $H \neq -1$ **then**
5 input $\{G, s, t, \{l_e\}, H\}$ to *Algorithm* 4 to find the least weighted path $\mathcal{P}$ for $s \to t$;
6 **else**
7 find the least weighted path $\mathcal{P}$ for $s \to t$ in G with Dijkstra's Algorithm;
8 **end**
9 **if** $\mathcal{P}$ *cannot be found* **then**
10 **return**(FALSE);
11 **end**
12 find the bottleneck edge e^* on $\mathcal{P}$;
13 **if** $R_e^* \geq d_{s,t}$ **then**
14 $b = d_{s,t}$, $d_{s,t} = 0$, and update $\mathbf{D}$;
15 **else**
16 $d_{s,t} = d_{s,t} - R_e^*$, $b = R_e^*$;
17 **end**
18 **for** *each edge* $e \in \mathcal{P}$ **do**
19 $R_e = R_e - b$, $l_e = l_e + b$;
20 **end**
21 **end**
22 **return**(R_E);

Algorithm 4: Shortest-Path Routing with Hop Constraint

Input: $G=(V,E)$, s, t, $\{l_e\}$, H, C

1 starting point set $S = \{s\}$, $S' = \emptyset$;
2 **for** *each node* $v \in V$ **do**
3 $L_v = +\infty$, $\mathcal{P}_v = \emptyset$;
4 **if** $v = s$ **then**
5 $L_v = 0$, insert s into $\mathcal{P}_v$;
6 **end**
7 **end**
8 **for** *each* $i \in [1, H]$ **do**
9 **for** *each node* $u \in S$ **do**
10 **for** *each node* $v \in V$ **do**
11 **if** *(edge* $e = (u, v) \in E$*) AND (*$l_e < C$*)* **then**
12 **if** $L_u + l_e < L_v$ **then**
13 $L_v = L_u + l_e$, $\mathcal{P}_v = \mathcal{P}_u$;
14 insert v into $\mathcal{P}_v$ and S';
15 **end**
16 **end**
17 **end**
18 **end**
19 $S = S'$, $S' = \emptyset$;
20 **end**
21 **return**($\mathcal{P}_t$);

latency-sensitive traffic demand under the maximum hop-count constraint of H. *Lines* 1-7 are for the initialization, where we initialize the set for the starting points of BFS (S) as the source s and set the known distance from s to each of the other nodes in V as $\{L_v = +\infty, \forall v \in V \setminus s\}$. Then, *Algorithm* 4 starts from s and searches H hops with BFS (*Lines* 8-20). We use set $\mathcal{P}_v$ to record the shortest-path from s to v. When BFS spreads to a node v, we update the known distance for $s \to v$ (*i.e.*, L_v) and $\mathcal{P}_v$ in *Lines* 12-14. When the H-hop BFS has been completed, the least-weighted routing path, which is for $s \to t$ and satisfies the maximum hop-count constraint of H, can be found in $\mathcal{P}_t$. In *Line* 11, the condition of $l_e < C$ is introduced to ensure that edges without any available bandwidth should not be considered in the routing path calculation.

C. Complexity Analysis

In the worst case, *Algorithm* 1 needs to perform K iterations, each of which *Algorithm* 2 runs once and conduct $|E|$ iterations at most. For each iteration in *Algorithm* 2, *Algorithm* 3 finds the least weighted path for each traffic demand, by invoking *Algorithm* 4 ($O(H \cdot |V|^2)$) or the Dijkstra's algorithm ($O(|V|^2)$). Therefore, the complexity of *Algorithm* 1 is $O(K \cdot |E| \cdot (|\mathbf{D}_{ls}| \cdot H \cdot |V|^2 + |\mathbf{D}_{lt}| \cdot |V|^2))$, which verifies that it is in polynomial time.

V. Performance Evaluation

In this section, we perform numerical simulations to evaluate the performance of our proposals.

A. Simulation Setup

We use the number of racks N to denote the scale of an optical DCN in Hyper-FleX-LION. Specifically, an N-Hyper-FleX-LION has N racks and each ToR switch in it has N TRXs for building the inter-rack topology through the WSS' and AWGR. The simulations consider 5 types of Hyper-FleX-LION in different scales[4], *i.e.*, $N = \{4, 8, 16, 32, 64\}$. The bandwidth capacity of each TRX is set as $C = 100$ Gbps [43]. To ensure practicalness of the simulations, we leverage the traffic patterns of real-world network services in DCNs to generate $\mathbf{D}_{ls}$ and $\mathbf{D}_{lt}$. The unit of data-rates in C, $\mathbf{D}$ and all the other related parameters and variables is 1 Mbps. We consider two types of latency-sensitive traffic with point-to-multipoint pattern, which are the Nvidia cloud gaming [44] and YouTube 4K video streaming [45], and set the demands of each of their jobs as 20 and 35 Mbps, respectively, according to [44, 45]. As for the latency-tolerant traffic, we address the data-transfers generated by DML in *Ring-AllReduce* and *Parameter-Server* architectures [37], and set their data-rates randomly distributed in $[1, 25]$ Gbps [46]. To generate $\mathbf{D}_{ls}$ and $\mathbf{D}_{lt}$, we randomly select source and destination racks for the latency-sensitive and latency-tolerant demands. The latency-sensitive demands in $\mathbf{D}_{ls}$ have their maximum hop-count as $H = 3$. To quantify the amount of traffic in the overall matrix $\mathbf{D} = \mathbf{D}_{ls} + \mathbf{D}_{lt}$, we define the **traffic load** as

[4]In practice, the scale of Hyper-FleX-LION is restricted as $N \leq 64$ due to the number of available wavelength channels and the port-counts of optical devices [20]. However, with the modular scaling scheme proposed in [14], large-scale DCNs in 3D-Hyper-FleX-LION can be built. Specifically, by interconnecting 64-Hyper-FleX-LIONs in a 3D-Hyper-FleX-LION, we can realize a large-scale DCN that contains $64^3 = 262,144$ racks.

$$\xi = \frac{\max\limits_{u \in V} \left(\sum\limits_{v \in V} d_{u,v} \right)}{N \cdot C}, \tag{7}$$

where $d_{u,v} \in \mathbf{D}$ denotes the total demands for $u \rightarrow v$.

The simulations consider the following four algorithms to solve the traffic-aware configuration for Hyper-FleX-LION.

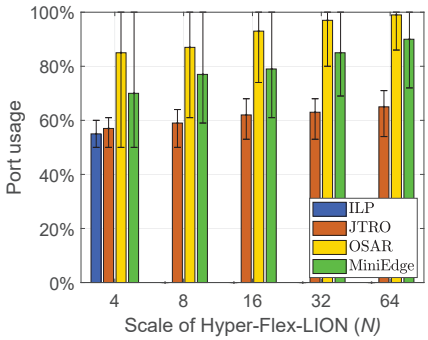
- **ILP**: the ILP model in Section III-C.
- **JTRO**: Our proposed heuristic in *Algorithm* 1.
- **OSAR**: We modify the algorithms in [27] to adapt to our problem. Specifically, we consider the traffic demands in $\mathbf{D}$ as edge weights and use the b-matching scheme in [27] for topology design (*i.e.*, $b = N$ in a Hyper-FleX-LION). Then, with the inter-rack topology determined, we use shortest-path routing to serve all the demands in $\mathbf{D}$.
- **MiniEdge**: We first design the inter-rack topology with *Lines* 1-18 in *Algorithm* 1. Then, we get a subgraph with minimum edges in the topology with the algorithm in [47] to further optimize it. Finally, we use shortest-path routing to serve $\mathbf{D}$ in the optimized topology.

Since OSAR and MiniEdge do not consider latency-sensitive traffic, we prioritize the demands in $\mathbf{D}_{ls}$ when planning traffic routing in OSAR and MiniEdge to ensure fair comparisons.

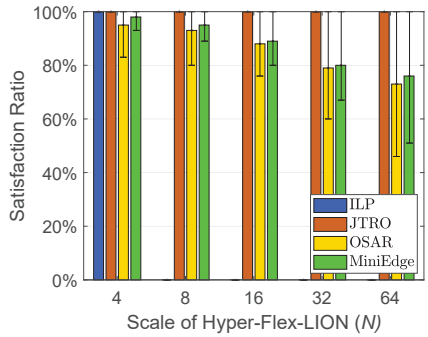
To evaluate the results of traffic-aware configuration, we introduce two metrics. First, the utilization of TRXs (*i.e.*, the **port usage**) is defined as $\frac{|E|}{N^2}$, where $|E|$ is the number of edges in inter-rack topology $G(V, E)$ and N^2 is the total available TRXs in an N-Hyper-FleX-LION. Second, if the QoS requirements (*i.e.*, bandwidth and hop-count) of a demand cannot be satisfied in a configuration of Hyper-FleX-LION, we will not serve it and label it as one without satisfied QoS. Hence, we use the ratio of the amount of the demands with satisfied QoS to the total traffic amount in $\mathbf{D}$ to represent the **satisfaction ratio** of traffic demands. The simulations are programmed with C++ and run on a computer with 4 GHz Intel Core i7-6700K CPU and 64 GB memory. To maintain sufficient statistical accuracy, we average the results from 60 independent runs to get each data point. In each run, we randomly generate a traffic matrix $\mathbf{D}$ according to the current parameters. To show the stability of the algorithms, we mark the range of 95% confidence interval in our results.

B. Performance with Different Hyper-FleX-LION Scales

We first evaluate the algorithms with Hyper-FleX-LIONs in different scales. The traffic load is fixed as 0.5. Fig. 4 shows the results on port usage and satisfaction ratio of traffic. Due to the time complexity of ILP, we can only solve it in the 4-Hyper-FleX-LION. In Fig. 4(a), we can see that JTRO achieves similar port usage as ILP in 4-Hyper-FleX-LION, and it always outperforms OSAR and MiniEdge significantly in all the scenarios. Specifically, JTRO always maintains the port usage at $\sim$60%, while the port usages from OSAR and MiniEdge can easily exceed 80%. The performance gaps between JTRO and OSAR/MiniEdge in Fig. 4(a) confirm that the sequential heuristics, which tackle the topology design and traffic routing in traffic-aware configuration of Hyper-FleX-LION in sequential steps, might not fully explore the flexibility of Hyper-FleX-LION for cost-efficient traffic provisioning.



(a) Port usage



(b) Satisfaction ratio of traffic demands

Fig. 4. Results for Hyper-FleX-LIONs in different scales.

Moreover, the smallest confidence intervals from JTRO in Fig. 4(a) further justify its advantages. In other words, OSAR and MiniEdge have much worse stability than JTRO, *i.e.*, their performance depends severely on the actual traffic matrix $\mathbf{D}$ even though the traffic load is the same. Meanwhile, it is interesting to notice that although we fix the traffic load as 0.5, the port usages from all the algorithms increase with the scale of Hyper-FleX-LION (N). This is because the traffic matrix $\mathbf{D}$ in a larger Hyper-FleX-LION contains more columns and rows to denote demands between more racks, and thus we will need to use more TRXs to serve the demands.

TABLE II
RUNNING TIME OF ALGORITHMS PER DEMAND ($d_{u,v} \in \mathbf{D}$) (MSEC)

Network Scale (N)	4	8	16	32	64
ILP	4258.664	-	-	-	-
JTRO	0.317	1.022	17.308	40.687	60.773
OSAR	0.006	0.156	1.617	3.138	2.740
MiniEdge	0.006	0.234	3.683	9.922	6.410

Fig. 4(b) shows the results on the satisfaction ratio of traffic demands, which indicate that JTRO always achieves 100% satisfaction ratio no matter how large the Hyper-FleX-LION is. However, OSAR and MiniEdge can hardly realize this goal. This is because they accomplish the traffic-aware configuration in sequential steps and do not consider latency-sensitive traffic when performing the topology design. Table II shows the average running time of the algorithms, when the traffic load is fixed as 0.5. Here, we show the average running time because the worst-case and best-case running time for each simulation scenario is always in the same order of magnitude.

It can be seen that for 4-Hyper-FleX-LION, all the heuristics

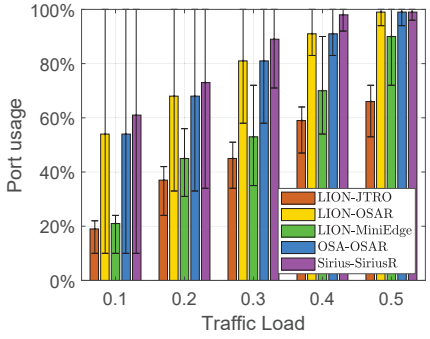
are much more time-efficient than ILP. As JTRO is based on iterative optimization, it takes longer running time than OSAR and MiniEdge. The running time of all the algorithms generally increases with the network scale, because $\mathbf{D}$ is more complex in a larger Hyper-FleX-LION. However, when N increases from 32 to 64, the running time of OSAR and MiniEdge actually decreases. This is because more demands cannot be served with satisfactory QoS by using OSAR and MiniEdge at $N = 64$, which saves certain running time for them. This analysis can be verified by checking the results on satisfaction ratio in Fig. 4(b). Hence, although JTRO takes more running time, it also obtains better DCN configurations by optimizing inter-rack topology and traffic routing jointly. Meanwhile, providing the fact that the frequency of topology changes in a DCN will only be a few times per day [48], the running time of JTRO will be acceptable for practical usage.

C. Performance with Different Traffic Loads

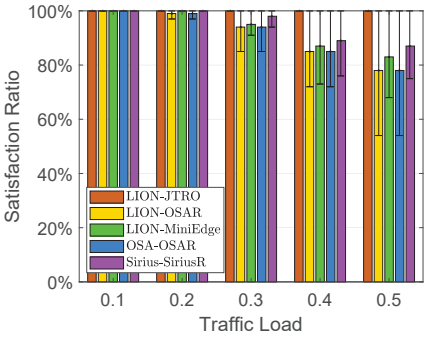
Then, we fix the scale of network scale as the largest ($N = 64$), investigate the performance of Hyper-FleX-LION using different configuration algorithms, and also benchmark Hyper-FleX-LION with two other architectures for all-optical DCNs (*i.e.*, OSA [27] and Sirius [13]). Since the routing strategy in Sirius only considers two-hop routing, we set $H = 2$ in the simulations in this subsection for fair comparisons.

We first apply different configuration algorithms to Hyper-FleX-LION, and in Fig. 5, LION-JTRO, LION-OSAR, and LION-MiniEdge denote the schemes of using JTRO, OSAR, and MiniEdge in Hyper-FleX-LION, respectively. The results in Fig. 5(a) suggest that the port usages of all the three schemes increase with traffic load and LION-JTRO always provides the smallest port usage with the smallest confidence interval. Among the three schemes, LION-OSAR always provides the largest port usage. LION-MiniEdge can obtain similar results as those of LION-JTRO when the traffic load is 0.1, but its performance gap to LION-JTRO increases significantly with traffic load. As for the results on satisfaction ratio in Fig. 5(b), LION-JTRO always ensures a satisfaction ratio of 100%, while the satisfaction ratios of LION-OSAR and LION-MiniEdge reduce significantly when the traffic load is 0.3 or higher.

Next, we discuss the performance comparison of Hyper-FleX-LION and other architectures. OSA-OSAR and Sirius-SiriusR represent the schemes of OSA and Sirius[5] using their own configuration algorithms. Similar to OSAR, we also let SiriusR prioritize the demands in $\mathbf{D}_{ls}$. To ensure a fair comparison, we assume that Sirius is built with 64-port AWGRs, OSA uses 64-port optical switches, and each ToR switch in them has 64 ports. Fig. 5(a) shows that the port usages of LION-OSAR and OSA-OSAR are similar, which again illustrates the importance of configuration algorithms for reconfigurable all-optical DCNs. Sirius-SiriusR always occupies the most ports among all the schemes, which is reasonable considering its operation principle (*i.e.*, spraying packets randomly to intermediate nodes in the TDM manner). As for the satisfaction ratio in Fig. 5(b), Sirius-SiriusR performs better than OSA-OSAR but it still cannot reach 100% when the traffic load is 0.3 or higher. In summary, the results in Fig. 5 suggest that JTRO performs the best in Hyper-FleX-LION in terms of both the port usage and satisfaction ratio, and the scheme of Hyper-FleX-LION using JTRO outperforms all the other schemes no matter which reconfigurable architecture is considered.



(a) Port usage



(b) Satisfaction ratio of traffic demands

Fig. 5. Results for 64-racks network with various traffic loads.

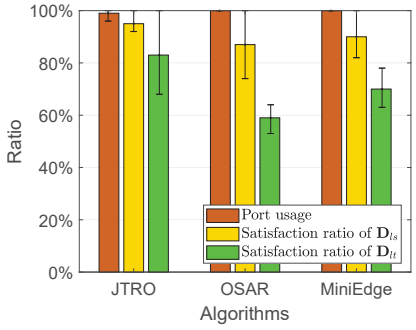


Fig. 6. Results for 64-Hyper-FleX-LIONs with traffic load at 0.8.

Next, we further stress the 64-Hyper-FleX-LION with traffic load at 0.8, and show the results in Fig. 6. Although there is seldom such a high inter-rack traffic load in real-world DCNs [49], we simulate it to observe the algorithms' performance in extreme cases. Fig. 6 indicates that OSAR and MiniEdge use all the TRXs in the Hyper-FleX-LION, while the satisfaction ratios of traffic demands achieved by them are pretty low. On the other hand, the average port usage from JTRO is still less

[5]As Sirius is actually an approach based on fixed time-division multiplexing (TDM), we, in the simulations, first transform the time-varying topology of Sirius into a static one by replacing the time-varying links between source-destination pairs with the static ones whose bandwidths are set according to the time ratio provided by the network schedule table of Sirius [13]. Then, the comparison between Sirius and our proposal becomes fair and meaningful.

than 100%, and it achieves the highest satisfaction ratios for demands in both $\mathbf{D}_{ls}$ and $\mathbf{D}_{lt}$. Meanwhile, it can be seen that for all the algorithms, the satisfaction ratios of the demands in $\mathbf{D}_{ls}$ are higher than those of the demands in $\mathbf{D}_{lt}$. This is because all the algorithms give a higher priority to the demands in $\mathbf{D}_{ls}$ during traffic routing.

D. Performance of JTRO with Different Settings

Finally, we explore the JTRO's algorithm performance. We first focus on the convergence performance, we still set the scale of Hyper-FleX-LION as the largest ($N = 64$) and fix the traffic load as 0.5 to investigate the convergence performance of JTRO. Fig. 7 shows how the port usage and satisfaction ratio from JTRO change in different iterations of *Algorithm* 1, related to the results from OSAR and MiniEdge. We observe that the performance of MiniEdge is similar as that of the initial result from JTRO at *Iteration* 0. As for JTRO, the performance of the traffic-aware configuration obtained by it increases quickly in iterations. Specifically, the satisfaction ratio of traffic demands reaches 100% at *Iteration* 20, while the port usage decreases to ∼65% at *Iteration* 80. These results verify the effectiveness of our proposed iterative optimization, *i.e.*, it can fully explore the flexibility of Hyper-FleX-LION to optimize the topology design and traffic routing in it jointly, for improving the cost-efficiency of provisioning.

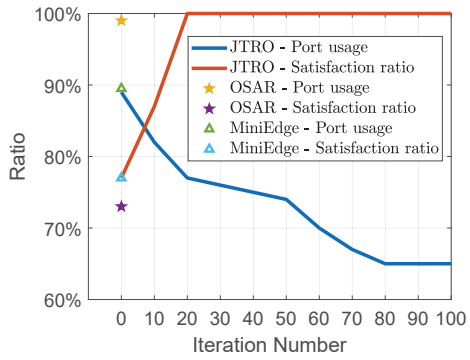


Fig. 7. Performance in different iterations.

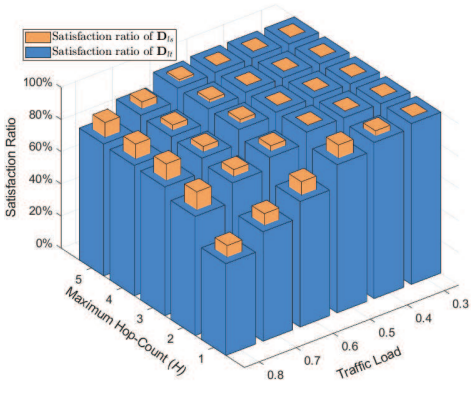


Fig. 8. Performance in different maximum hop-counts and traffic loads.

Then, we study the impacts of the maximum hop-count (H) and traffic load on the performance of JTRO in 64-Hyper-FleX-LION. Fig. 8 shows the worst satisfaction ratios of the demands in $\mathbf{D}_{ls}$ and $\mathbf{D}_{lt}$ with different combinations of H and traffic load. In Fig. 8, we can see that the satisfaction ratios of demands in $\mathbf{D}_{ls}$ and $\mathbf{D}_{lt}$ both decrease when the traffic load increases or H decreases. When we have $H = 1$, the satisfaction ratios can drop precipitously with the increase of traffic load. This is because multi-hop routing is not allowed in this case, which highlights the issue of coarse granularity of the wavelength switching in Hyper-FleX-LION.

Although a larger H can make the routing of the demands in $\mathbf{D}_{ls}$ more flexible to improve their satisfaction ratio, the gain converges after H reaches 3. This is because routing a demand with more hops will occupy more bandwidth resources, which will eventually offset the benefit bought by flexible routing. On the other hand, when the traffic load is greater than 0.6, the satisfaction ratios cannot reach 100% even when we have $H = 5$. This suggests that when the traffic load is relatively high, the available bandwidth provided by Hyper-FleX-LION can become insufficient to satisfy the QoS requirements of all the demands even with flexible multi-hop routing.

VI. Experimental Demonstrations

In this section, we build a small-scale all-optical DCN testbed in Hyper-FleX-LION, and leverage DML to demonstrate the effectiveness of our proposal experimentally.

A. Experimental Setup

We build an all-optical DCN testbed to interconnect 4 racks (*i.e.*, 4-Hyper-FleX-LION), whose key components and setup are shown in Fig. 9. Here, we slice a commercial OpenFlow-enabled packet switch (Pica8) into 4 virtual switches, each of which is assigned a dedicated group of ports, and leverage the virtual switches to realize 4 independent ToR switches. Each ToR switch uses one port to connect to the server in its rack, and connects its four other ports to the optical inter-rack network in 4-Hyper-FleX-LION (as shown in Fig. 1(a)). The port to the server operates in the wavelength range of ∼850 nm, while the four ports involved in the inter-rack network operates in the C-band at ∼1535 nm. We build the 4-Hyper-FleX-LION with one 8×8 AWGR, eight 1×9 WSS', eight couplers, and other passive components, which are all commercial products, according to the architecture in Fig. 1(a).

Each rack consists of a high-performance server that contains four 6-core CPUs and 32 GB of memory, and we instantiate two virtual machines (VMs) on it to emulate a rack of two servers. All the TRXs used for the ports in the 4-Hyper-FleX-LION have a capacity of 10 Gbps. However, we limit the throughput of each of them as 1 Gbps to make sure that the data traffic of DML can easily saturate a port in the 4-Hyper-FleX-LION, for stressing the optical inter-rack network and fully exploring its flexibility for service provisioning.

As for the DML, we leverage the famous CIFAR-10 data set, which contains 60,000 32×32 color pictures in 10 different categories, and train a convolutional neural network (CNN) with it for image classification. The CNN is architected with the well-known ResNet model, and contains 18 layers (*i.e.*, ResNet-18). Our experiments consider two commonly-used DML architectures, *i.e.*, *Ring-AllReduce (Ring)* and *Parameter*

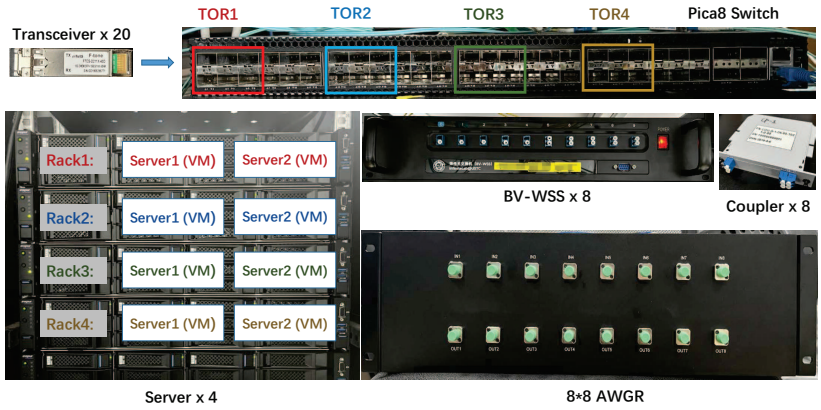


Fig. 9. Equipments included in the 4-Hyper-FleX-LION testbed.

Server (PS) [37]. The computing nodes of a DML job in *Ring* generate ring-like traffic demands, *i.e.*, each node sends/receives data to/from its next/previous node. There are two types of computing nodes involved in a DML job in *PS*, *i.e.*, the master and worker nodes. Specifically, there is always one master node and multiple worker nodes, and they form a tree, where the master node is the root. Hence, we can denote a DML job in *PS* that contains n worker nodes as *PS*(n).

In each experiment, we run the computing nodes of a DML job in the VMs, make the control plane of the all-optical DCN collect the bandwidth usage on each TRX with periodic polling to get the traffic matrix $\mathbf{D}$, and apply OSAR or JTRO as the algorithm to configure the 4-Hyper-FleX-LION and route traffic demands in it. The initial topology of the 4-Hyper-FleX-LION is always the fully-connected one in the left part of Fig. 1(b). Two metrics are used to evaluate the performance of the algorithms, *i.e.*, the job completion time (JCT) and port usage.

B. Results and Discussions

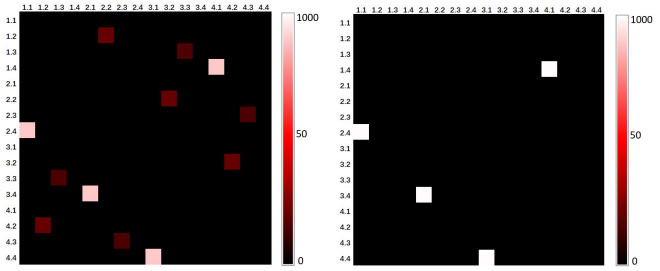
We first run a DML job in *Ring* that consists of 4 computing nodes, each of which is placed in a rack in the testbed. In the experiment, we notice that the DML job generates three types of traffic demands, forming a clockwise ring, a counter-clockwise ring, and a sparse mesh, respectively. The demands in the clockwise ring are for the data exchange among the computing nodes, and their average throughput is $\sim$900 Mbps, which contributes to over 94% of the bandwidth usage of the DML job. As the processing of these demands directly affects the JCT of the DML job, we treat them as latency-sensitive traffic to put into $\mathbf{D}_{ls}$ and set their maximum hop-count as $H = 2$. On the other hand, the demands in the counter-clockwise ring and sparse mesh are for the control signaling among the computing nodes to maintain the operation of the DML job, and their average throughputs are $\sim$30 Mbps and $\sim$15 Mbps, respectively. These demands can be treated as latency-tolerant traffic, and thus we put them into $\mathbf{D}_{lt}$.

Fig. 10 shows the 2D heat-maps for port usage, when the 4-Hyper-FleX-LION is managed by OSAR and JTRO. Here, we denote a TRX as "$x.y$", where x refers to the index of the TRX's ToR switch and y is the index of the TRX on its ToR switch. For instance, "1.1" refers to the first TRX on *ToR Switch* 1. In each 2D heat-map, the horizontal and vertical axes respectively represent the TX and RX parts of TRXs. The color of each square, which is determined by a pair of TRXs, denotes the bandwidth usage of the traffic between them in the corresponding direction. To clearly illustrate any bandwidth usage within $[0, 1]$ Gbps, the heat-maps use a nonlinear color scale, where the mediate point is 50 Mbps and colored as red.

Fig. 10(a) indicates that OSAR uses 12 pairs of TRXs to accommodate the traffic demands generated by the DML job in *Ring*, while the heat-map in Fig. 10(b) suggests that JTRO can consolidate the port usage much better and only uses 4 pairs of TRXs. Meanwhile, by comparing Figs. 10(a) and 10(b), we can see that the four brightest squares in them are the same, but the heat-map in Fig. 10(a) contains more squares in dark red. Specifically, in Fig. 10(a), the bright squares correspond to the demands in the clockwise ring ($\mathbf{D}_{ls}$), while the squares in dark red are for those generated by the control signaling among the computing nodes ($\mathbf{D}_{lt}$). Therefore, the results in Fig. 10 confirm the effectiveness of JTRO. In other words, as OSAR does not optimize the configuration of 4-Hyper-FleX-LION and the routing of traffic demands in it jointly, more pairs of TRXs have to be used to provision the small but irregular traffic for the control signaling of DML.

Table III lists the results on JCT, which show that even though JTRO uses much fewer pairs of TRXs, it achieves a JCT that is almost the same as that from OSAR. Therefore, by combining the results in Fig. 10 and Table III, we demonstrate that without sacrificing the QoS of DML jobs, JTRO optimizes the configuration of 4-Hyper-FleX-LION and the routing of traffic demands in it jointly, and uses TRXs more efficiently.

Next, we change the DML job to use the architecture of *PS*(3), *i.e.*, one master node and three worker nodes form a tree-type topology. Each of the nodes is placed in a rack. Similar to the case of *Ring*, the demands among the nodes are for the data exchange among the computing nodes and the control signaling to maintain the operation of the DML job. Our measurements show that the former type of demands ($\mathbf{D}_{ls}$) have an average throughput is $\sim$300 Mbps, while the average throughput of the latter type of demands ($\mathbf{D}_{lt}$) is $\sim$15 Mbps. We still set the maximum hop-count of demands in $\mathbf{D}_{ls}$ as $H = 2$. Figs. 11(a) and 11(b) show the 2D heat-maps for port usages from OSAR and JTRO, which illustrate the similar

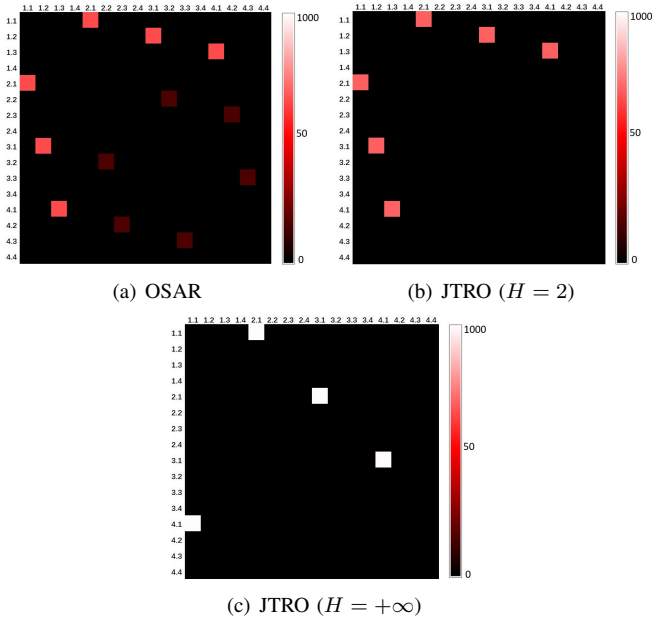


(a) OSAR (b) JTRO

Fig. 10. 2D heat-maps for 4-Hyper-FleX-LION (DML in *Ring*).

trend as that in Fig. 10. This time, OSAR uses 6 more pairs of TRXs than JTRO to accommodate the demands from the DML job, still verifying the effectiveness of JTRO on consolidating port usage. The results on JCT in Table III also confirm that JTRO achieves almost the same JCT as OSAR.

Then, to observe the effect of H on the performance of JTRO, we relax it for the demands in $\mathbf{D}_{ls}$ (*i.e.*, $H = +\infty$). Fig. 11(c) shows the 2D heat-maps of JTRO in this case. We can see that when there is no restriction on H, JTRO only needs to uses 4 pairs of TRXs to support the demands of the DML job in *PS*(3). Specifically, JTRO configures the 4-Hyper-FleX-LION as a ring, and the demands in $\mathbf{D}_{ls}$ can be routed through 3 hops in the worst case. The JCT in Table III indicates that relaxing the constraint on H does degrade the QoS of the DML job slightly, *i.e.*, its JCT gets increased by 2.8%. This verifies the motivation of restricting the maximum hop-count of latency-sensitive traffic demands.

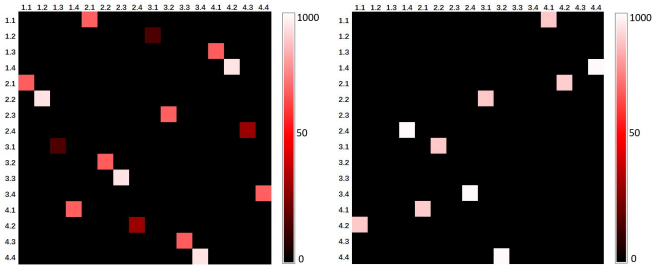


(a) OSAR (b) JTRO ($H = 2$)

(c) JTRO ($H = +\infty$)

Fig. 11. 2D heat-maps for 4-Hyper-FleX-LION (DML in *PS*(3)).

Finally, we try to run DML jobs in different architectures in the 4-Hyper-FleX-LION simultaneously to generate a more complicated traffic matrix and further evaluate the performance of the algorithms. Specifically, we run one DML job in *Ring* and two others in *PS*(2) at the same time, and still set $H = 2$. The 2D heat-maps in Fig. 12 still show that JTRO can consolidate the port usage in the 4-Hyper-FleX-LION better than OSAR, saving 6 pairs of TRXs. Meanwhile, the results in Table III also confirm that JTRO and OSAR achieve almost the same JCT, where the JCT from JTRO is only 0.3% longer than that from OSAR. In all, the aforementioned experimental results prove that JTRO can fully explore the flexibility of Hyper-FleX-LION to efficiently support arbitrary traffic matrices, not only effectively reducing unnecessary port usage but also maintaining the QoS of network services well.



(a) OSAR (b) JTRO

Fig. 12. 2D heat-maps for 4-Hyper-FleX-LION (DML in *Ring* and *PS*(2)).

TABLE III
RESULTS ON AVERAGE JCT (SECONDS)

	OSAR	JTRO	JTRO ($H = +\infty$)
Ring	151.78	151.89	-
PS(3)	779.38	779.92	801.23
Ring & 2×*PS*(2)	1886.38	1892.22	-

VII. CONCLUSION

This work studies how to realize traffic-aware configuration of all-optical DCNs in Hyper-FleX-LION. To reduce the OPEX of such an all-optical DCN, we tried to jointly optimize the configuration of Hyper-FleX-LION and the provisioning schemes of demands in it, for minimizing its port usage, and formulated an ILP model for the problem. By performing a complexity analysis on the problem, we proved that it is $\mathcal{APX}$-hard. Therefore, we proposed a polynomial-time heuristic based on iterative optimization to solve it effectively and time-efficiently. Extensive numerical simulations confirmed the performance of our proposed algorithm and showed that it could outperform existing benchmarks. Finally, we built a small-scale but real all-optical DCN testbed in Hyper-FleX-LION, and leveraged DML as the network services in it to demonstrate the effectiveness of our proposal experimentally. The results indicated that our proposal could fully explore the flexibility of Hyper-FleX-LION to efficiently support arbitrary traffic matrices, not only effectively reducing unnecessary port usage but also maintaining the QoS of network services well.

ACKNOWLEDGMENTS

This work was supported by NSFC project 62371432.

REFERENCES

[1] P. Lu *et al.*, "Highly-efficient data migration and backup for Big Data applications in elastic optical inter-datacenter networks," *IEEE Netw.*, vol. 29, pp. 36–42, Sept./Oct. 2015.

[2] "Cisco Annual Internet Report (2018-2023)," *Online White Report*. [Online]. Available: https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html.

[3] W. Lu *et al.*, "AI-assisted knowledge-defined network orchestration for energy-efficient data center networks," *IEEE Commun. Mag.*, vol. 58, pp. 86–92, Jan. 2020.

[4] M. Abadi *et al.*, "Tensorflow: A system for large-scale machine learning," in *Proc. of OSDI 2016*, pp. 265–283, Nov. 2016.

[5] Z. Zhu, W. Lu, L. Zhang, and N. Ansari, "Dynamic service provisioning in elastic optical networks with hybrid single-/multi-path routing," *J. Lightw. Technol.*, vol. 31, pp. 15–22, Jan. 2013.

[6] L. Gong *et al.*, "Efficient resource allocation for all-optical multicasting over spectrum-sliced elastic optical networks," *J. Opt. Commun. Netw.*, vol. 5, pp. 836–847, Aug. 2013.

[7] N. Bitar, S. Gringeri, and T. Xia, "Technologies and protocols for data center and cloud networking," *IEEE Commun. Mag.*, vol. 51, pp. 24–31, Sept. 2013.

[8] Z. Zhu *et al.*, "Energy-efficient translucent optical transport networks with mixed regenerator placement," *J. Lightw. Technol.*, vol. 30, pp. 3147–3156, Oct. 2012.

[9] Y. Yin *et al.*, "Spectral and spatial 2D fragmentation-aware routing and spectrum assignment algorithms in elastic optical networks," *J. Opt. Commun. Netw.*, vol. 5, pp. A100–A106, Oct. 2013.

[10] N. Farrington *et al.*, "Helios: a hybrid electrical/optical switch architecture for modular data centers," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 40, pp. 339–350, Oct. 2010.

[11] G. Wang *et al.*, "c-Through: Part-time optics in data centers," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 41, pp. 327–338, Oct. 2011.

[12] J. Benjamin *et al.*, "PULSE: Optical circuit switched data center architecture operating at nanosecond timescales," *J. Lightw. Technol.*, vol. 38, pp. 4906–4921, May 2020.

[13] H. Ballani *et al.*, "Sirius: A flat datacenter network with nanosecond optical switching," in *Proc. of ACM SIGCOMM 2020*, pp. 782–797, Jul. 2020.

[14] G. Liu *et al.*, "Architecture and performance studies of 3D-Hyper-FleX-LION for reconfigurable All-to-All HPC networks," in *Proc. of SC 2020*, pp. 1–16, Nov. 2020.

[15] S. Wang *et al.*, "A scalable, high-performance, and fault-tolerant network architecture for distributed machine learning," *IEEE/ACM Trans. Netw.*, vol. 28, pp. 1752–1764, Aug. 2020.

[16] L. Gong and Z. Zhu, "Virtual optical network embedding (VONE) over elastic optical networks," *J. Lightw. Technol.*, vol. 32, pp. 450–460, Feb. 2014.

[17] S. Li, D. Hu, W. Fang, and Z. Zhu, "Source routing with protocol-oblivious forwarding (POF) to enable efficient e-health data transfers," in *Proc. of ICC 2016*, pp. 1–6, Jun. 2016.

[18] Z. Zhu *et al.*, "Impairment- and splitting-aware cloud-ready multicast provisioning in elastic optical networks," *IEEE/ACM Trans. Netw.*, vol. 25, pp. 1220–1234, Apr. 2017.

[19] J. Liu *et al.*, "On dynamic service function chain deployment and readjustment," *IEEE Trans. Netw. Serv. Manag.*, vol. 14, pp. 543–553, Sept. 2017.

[20] X. Xiao *et al.*, "Silicon photonic Flex-LIONS for bandwidth-reconfigurable optical interconnects," *IEEE J. Sel. Top. Quantum Electron.*, vol. 26, pp. 1–10, Mar./Apr. 2020.

[21] X. Xiao, R. Proietti *et al.*, "Multi-FSR silicon photonic Flex-LIONS module for bandwidth-reconfigurable all-to-all optical interconnects," *J. Lightw. Technol.*, vol. 38, pp. 3200–3208, Jun. 2020.

[22] H. Yang, Z. Zhu, R. Proietti, and B. Yoo, "Which can accelerate distributed machine learning faster: Hybrid optical/electrical or optical reconfigurable DCN?" in *Proc. of OFC 2022*, pp. 1–3, Mar. 2022.

[23] T. Benson, A. Anand, A. Akella, and M. Zhang, "Understanding data center traffic characteristics," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 40, pp. 92–99, Jan. 2010.

[24] Q. Li *et al.*, "Scalable knowledge-defined orchestration for hybrid optical/electrical datacenter networks," *J. Opt. Commun. Netw.*, vol. 12, pp. A113–A122, Feb. 2020.

[25] Z. Zhao, B. Guo, Y. Shang, and S. Huang, "Hierarchical and reconfigurable optical/electrical interconnection network for high-performance computing," *J. Opt. Commun. Netw.*, vol. 12, pp. 50–61, Mar. 2020.

[26] M. Teh, Z. Wu, and K. Bergman, "Flexspander: augmenting expander networks in high-performance systems with optical bandwidth steering," *J. Opt. Commun. Netw.*, vol. 12, pp. B44–B54, Apr. 2020.

[27] K. Chen *et al.*, "OSA: An optical switching architecture for data center networks with unprecedented flexibility," *IEEE/ACM Trans. Netw.*, vol. 22, pp. 498–511, Apr. 2013.

[28] Y. Tang *et al.*, "Effectively reconfigure the optical circuit switching layer topology in data center network by OCBridge," *J. Lightw. Technol.*, vol. 37, pp. 897–908, Feb. 2019.

[29] M. William *et al.*, "Expanding across time to deliver bandwidth efficiency and low latency," in *Proc. of NSDI 2020*, pp. 1–18, Feb. 2020.

[30] S. Zhao and Z. Zhu, "On virtual network reconfiguration in hybrid optical/electrical datacenter networks," *J. Lightw. Technol.*, vol. 38, pp. 6424–6436, Dec. 2020.

[31] H. Fang *et al.*, "Predictive analytics based knowledge-defined orchestration in a hybrid optical/electrical datacenter network testbed," *J. Lightw. Technol.*, vol. 37, pp. 4921–4934, Oct. 2019.

[32] C. Wang, N. Yoshikane, F. Balasis, and T. Tsuritani, "Acceleration and efficiency warranty for distributed machine learning jobs over data center network with optical circuit switching," in *Proc. of OFC 2021*, pp. 1–3, Jun. 2021.

[33] D. Borthakur, *The Hadoop Distributed File System: Architecture and Design*. Apache Software Foundation, 2007.

[34] S. Choy, B. Wong, G. Simon, and C. Rosenberg, "The brewing storm in cloud gaming: A measurement study on cloud to end-user latency," in *Proc. of NetGames 2012*, pp. 1–6, Dec. 2012.

[35] M. Besta *et al.*, "FatPaths: Routing in supercomputers and data centers when shortest paths fall short," in *Proc. of SC 2020*, pp. 1–18, Nov. 2020.

[36] S. Kaur, K. Kumar, and N. Aggarwal, "A review on P4-programmable data planes: Architecture, research efforts, and future directions," *Comput. Commun.*, vol. 170, pp. 109–129, Mar. 2021.

[37] J. Verbraeken *et al.*, "A survey on distributed machine learning," *arXiv preprint arXiv:1912.09789*, Dec. 2019. [Online]. Available: https://arxiv.org/abs/1912.09789.

[38] M. Garey and D. Johnson, *Computers and Intractability: a Guide to the Theory of NP-Completeness*. W. H. Freeman & Co. New York, 1979.

[39] S. Even, A. Itai, and A. Shamir, "On the complexity of timetable and multicommodity flow problems," *SIAM J. Comput.*, vol. 5, pp. 691–703, Dec. 1976.

[40] Edge coloring. [Online]. Available: http://en.wikipedia.org/wiki/Edge_coloring.

[41] J. Misra and G. David, "A constructive proof of Vizing's theorem," *Inf. Process. Lett.*, vol. 41, pp. 131–133, Mar. 1992.

[42] Breadth-first search. [Online]. Available: https://en.wikipedia.org/wiki/Breadth-first_search.

[43] Optical Transceiver Market with COVID-19 Impact Analysis, by Form Factor (SFF and SFP; SFP+ and SFP28; XFP; CXP), Data Rate, Wavelength, Fiber Type, Connector, Distance, Protocol, Application (Data Center, Enterprise), and Region - Global Forecast to 2026. [Online]. Available: https://www.marketsandmarkets.com/Market-Reports/optical-transceiver-market-161339599.html.

[44] Nvidia GEFORCE NOW. [Online]. Available: https://www.nvidia.com/en-us/geforce-now/system-reqs/.

[45] How much speed do I need to stream video? [Online]. Available: https://www.highspeedinternet.com/resources/how-much-speed-do-i-need-to-watch-netflix-and-hulu.

[46] Amazon EC2 G4 instances. [Online]. Available: https://aws.amazon.com/ec2/instance-types/g4/?nc1=h_ls.

[47] G. Frederic, M. Joanna, and K. Truong, "Optimizing rule placement in software-defined networks for energy-aware routing," in *Proc. of GLOBECOM 2014*, pp. 2523–2529, Dec. 2014.

[48] M. Zhang *et al.*, "Gemini: practical reconfigurable datacenter networks with topology and traffci engineering," *arXiv preprint arXiv:2110.08374*, Oct. 2021. [Online]. Available: http://arxiv.org/abs/2110.08374.

[49] S. Chandrasekaran, *Understanding Traffic Characteristics in a Server to Server Data Center Network*. Rochester Institute of Technology, 2017.