HierTopo: Towards High-Performance and Efficient Topology Optimization for Dynamic Networks

Jing Chen[1,4], Zili Meng[1,4], Yaning Guo[1,4], Mingwei Xu[1,2,4], Hongxin Hu[3]
[1]Institute for Network Science and Cyberspace, Tsinghua University
[2]Department of Computer Science and Technology, Tsinghua University
[3]Department of Computer Science and Engineering, University at Buffalo
[4]Beijing National Research Center for Information Science and Technology (BNRist)
j-chen16@tsinghua.org.cn, {mzl19,gyn17}@mails.tsinghua.edu.cn, xumw@tsinghua.edu.cn, hongxinh@buffalo.edu

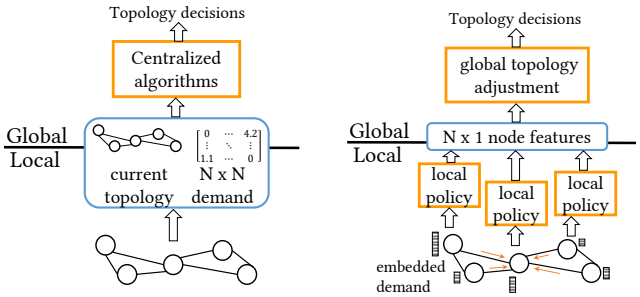
***Abstract*—Dynamic networks have enabled dynamically adapting the network topology to meet the need of real-time traffic demands. However, due to the complexity of topology optimization, existing solutions suffer from a trade-off between performance and efficiency, which either have large optimality gaps or excessive optimization overhead. To break through this trade-off, our key observation is that we could offload the optimization procedure to every network node to handle the complexity. Thus, we propose HierTopo, a hierarchical topology optimization method for dynamic networks that achieves both high performance and efficiency. HierTopo firstly runs a local policy on each network node to aggregate network information into low-dimension features, then uses these features to make global topology decisions. Evaluation on real-world network traces shows that HierTopo outperforms the state-of-the-art solutions by 11.52-38.91% with only milliseconds of decision latency, and is also superior in generalization ability.**

I. Introduction

Due to the time-varying traffic demands, recent advances in the networking community propose to dynamically reconfigure the network topology in real time to further improve the network performance. Examples include reconfigurable data centers [1, 2], wireless sensor networks [3], and low-earth orbit (LEO) satellite networks [4, 5]. Adapting the network topology to fast-changing demands could enable shorter path lengths and therefore higher network performance for large traffic demands. For example, according to Microsoft, by dynamically adjusting the network topology, ProjecToR [1] could improve the flow completion time by one magnitude compared to the static network topology.

When optimizing the network topology, the performance on the average path length and the efficiency of the optimization are of great emphasis. On one hand, a topology that offers a shorter path length could reduce the end-to-end latency for traffic demands and accommodate more traffic with the same number of links. Especially, datacenter applications may have microsecond-scale latency requirements while pursuing a high throughput for cost-efficiency [6]. On the other hand, due to the time-changing demands, algorithms are also expected to efficiently capture the sub-second temporal changes in traffic demands [7]. The optimal topology should also timely adapt to the varying network scale (e.g., when a network device joins or leaves a sensor network), which poses requirements on algorithm efficiency.

978-0-7381-3207-5/21/$31.00 ©2021 IEEE



(a) Centralized. (b) Hierarchical.

Fig. 1. Comparison between centralized and hierarchical topology optimization. Existing centralized algorithms directly take high-dimensional information as inputs, attempt laboriously to draw a topology decision by synthesizing all the information. In contrast, HierTopo first embeds the traffic demand information to network nodes and aggregates these information with the local policy into node features, and then the global algorithm can easily make decisions based on these features.

However, recent research efforts on topology optimization for dynamic networks fail to achieve the *performance* and *efficiency* at the same time. They usually fall into two categories: Sophisticated models achieve good performance but consume too much computation time, while heuristic methods sacrifice performance for fast execution (§II-B). The root cause of the above trade-off is that existing methods try to solve the complicated problem in a centralized way. Topology optimization is an extremely complicated combinatorial optimization problem with complex inputs (e.g., current topology as structural information), a huge action space (i.e., a squarely exponential space), nonlinear optimization goals (e.g., path length) and nontrivial constraints (e.g., limitation on node degree) (§II-A), which is already proven to be NP-hard [8]. Therefore, a centralized topology control is burdened with making topology decisions by taking global-wide information altogether into account (as shown in Figure 1(a)), and eventually overwhelmed by the high complexity. Therefore, existing solutions have to trade off between performance and efficiency.

In response, our key observation is that topology optimization could be decoupled into network information aggregation and link decision. The aggregation of network information on different nodes could be offloaded to every network node and executed distributedly. The global link decision algorithm

could therefore be kept simple. Based on this observation, we present HierTopo, a *hierarchical* method that efficiently configures high-performance topologies for dynamic networks. As illustrated by Figure 1(b), HierTopo first uses a *local policy* that runs on each network node in parallel to make topological decisions. The local policy is designed to be independent of the network scale or a certain traffic pattern, henceforth it can be flexibly applied to reconfigure the topology without reoptimization. Then, HierTopo configures the topology with a global topology adjustment algorithm leveraging the output of the local policy. We further demonstrate the optimality of our hierarchical framework by theoretical analysis (§IV-C).

However, achieving both high efficiency and high performance is still challenging during the design of the hierarchical framework (§II-C). The efficiency is hurdled by the large action space and nontrivial constraints that are inherent to the topology optimization problem. In response, HierTopo reduces the action space by *incremental adjustment* and introduces a decision variable, *node feature*, which is able to express a topology decision with fixed-dimension output from the local policy. As for the performance, we are challenged to collect sufficient yet concise information to the local policy and fully optimize the local policy. To resolve these challenges, HierTopo iterates the local policy for several turns to aggregate distant information, approximates the local policy with a polynomial function, and adopts the *Genetic Algorithm* to optimize the coefficients in the local policy.

We evaluate HierTopo with real-world traces. Experiments show that HierTopo achieves high performance and high efficiency at the same time. Compared with heuristic methods, HierTopo improves the performance (i.e., reduces the average shortest path length) by up to 38.91% over Ego-tree (§VI-B), and 13.93% over greedy matching within only 4 steps of adjustment on a 50-node topology (§VI-E). Compared with sophisticated optimization methods, HierTopo is lightweight for online deployment, since its decision latency is limited to millisecond level (§VI-D). Moreover, HierTopo demonstrates satisfactory generalization ability: Using an old local policy in different-scale networks with up to 50 nodes brings $\leq$1.89% performance degradation (§VI-C1). HierTopo also shows a leading advantage during generalizing to other workload distributions (§VI-C2). Further evaluation shows that HierTopo performs well on parameter sensitivity and has a small optimality gap within limited-step adjustments (§VI-E).

In summary, this paper makes the following contributions:

- We illustrate the existing trade-off between performance and efficiency of topology optimization for dynamic networks (§II-B). We thus present HierTopo, a hierarchical framework to optimize the topology for dynamic networks with high performance and high efficiency (§III).
- We design a local policy (§IV) and a global algorithm (§V), which constitute HierTopo's topology optimization. We also analyze the optimality of the hierarchical framework.
- We evaluate HierTopo under real-world traffic and topology datasets. Experiments show that HierTopo constructs high-performance topologies efficiently and the local policy has advantageous generalization ability (§VI).

II. Background and Motivation

We first introduce the background of dynamic networks in §II-A, and motivate the design of HierTopo in §II-B.

A. Dynamic networks

Dynamic networks have rapidly evolved in their infrastructures over recent years and have great potential to lead a radical change to future networking. There are lines of research in different scenarios from the academic and industrial communities on replacing static-topology networks with dynamic networks.

For example, datacenters have intensive communications, but only 1-54% of rack pairs are involved in the communications in a time interval of a few minutes [1], so traditional datacenters that provide the same capacity between every top-of-rack switch suffer from congestion between rack pairs that exchange traffic while over-provisioning links between other racks. Motivated by the traffic imbalance, industrial giants such as Microsoft introduced reconfigurable data center [1] infrastructures to replace the static network topology. Supported by the development of communication technologies, such as free space optics (FSO) [1, 9], optical circuit switches (OCS) [10, 11], and 60GHz wireless links [12], reconfigurable datacenters could agilely change the topology within one millisecond [7] to meet the time-varying demand and enhance the network performance by fully exploiting the capacity.

In wide-area networks, with recent advances in satellite communication, researchers also propose to replace the static Internet backbones with dynamic satellite networks to improve the latency and capacity. By transmitting the data freely along a straight line and flexibly building the topology, satellite networks could adapt to the demands and shorten the end-to-end latency against terrestrial Internet. The rise of satellite networks attracts the attention from the industry and academia: Many companies plan to launch tens of thousands of satellites and build their megaconstellations, e.g., Starlink (SpaceX) [5] and Kuiper (Amazon) [13]. Research efforts have also been devoted to optimize the topology among satellites [4].

B. Motivation

Existing solutions to the topology optimization problem essentially fall into two categories, but are flawed in either performance or efficiency.

High-performance but inefficient. A major category of existing solutions is optimization-based methods. For example, based on careful modeling, WiRo [14] casts the optimization problem into an integer linear programming (ILP). However, since this ILP is NP-hard [8], it takes minutes to solve the ILP in a 128-node network even with constraint relaxation. xWeaver [15] adopts deep learning to optimize the topology. However, due to a lack of generalization ability, xWeaver requires rebuilding the training dataset and retraining from scratch every time when adopted to a new-sized network or to an unseen traffic distribution. Even accelerated with specific hardware, hours to days of additional time is needed (§VI-D).

Efficient but non-performant. Another category of topology optimization methods is heuristic and usually rooted in graph theory. For example, matching-based algorithms [1, 16] greedily build up new links where the traffic demand is the largest. cl-DAN [17] considers the scenario of sparse demands and uses ego-trees to build up the network. However, due to the non-triviality of topology optimization, such heuristic algorithms usually result in a huge optimality gap, as the network scales slightly larger. For example, ego-tree has a performance gap of more than 30% in a network with 50 nodes compared to xWeaver (§VI-B).

Therefore, we need to develop a new topology optimization method that enjoys both high performance and high efficiency.

C. Design challenges

However, to simultaneously achieve efficiency and high performance in topology optimization is still challenging.

To achieve high efficiency, we are challenged to reduce the action space and handle the complexity. Enumerating all possible topologies will result in a huge action space. For example, a 30-node network could have an action space that is even larger than the game of Go [18]. Even if we divide it up to every network node, the action space is still exponentially large if the local policy is responsible for deciding whether a network node should be directly connected to every other node. Moreover, the constraints considering practical deployment issues create more obstacles. For example, practically deployed dynamic networks usually have limits on the number of neighbors due to physical constraints [17]. Manually ruling out all invalid topologies is so time-consuming that enforcing the constraints is also a big hurdle for efficiency. In response, HierTopo incrementally adjusts the topology – only adds (or removes) one link in the topology at a time, so the task is simplified to only picking out a link position. Before each time of adjustment, checking the validity of adding a link to the selected position is effortless. In this way, the constraints are easily satisfied. Furthermore, HierTopo introduces a decision variable, denoted as *node features*, to neatly express the decision of a topology adjustment with one real number for each network node. So the final action space of the local policy is fixed-dimensional.

To achieve high performance, our method must collect decision-related information and fully optimize the topology. However, by definition, the local policy only takes neighborhood information, so how to obtain remote critical information and make high-performance topology decisions is an issue. Second, the optimization goals in networking are usually difficult to be explicitly expressed. For example, it is unable to express the length of the shortest path between two nodes in a topology with polynomial expressions of the adjacency matrix. In response, to acquire all related information, HierTopo iterates the local policy until convergence, so global information can be aggregated through several iterations. For full optimization, HierTopo approximates the local policy with a polynomial function that updates node features with neighborhood features and uses the Genetic Algorithm to optimize the coefficients in the polynomial function.

III. HierTopo design

In this section, we first formulate the topology optimization problem for dynamic networks in §III-A, then we outline HierTopo's design framework in §III-B.

A. Problem Formulation

Let $G = (V, E)$ be the graph of network topology, where $v \in V$ is a network node and $e \in E$ an edge. N is the network size, i.e., the number of nodes contained in G, $N = |V|$. $\mathbb{N}(v)$ represents the set of v's neighbor nodes. Based on [8, 17], we briefly reproduce the formulation of the topology optimization problem for dynamic networks in general as follows:

Input: $D \in \mathbb{R}^{N \times N}$ represents the demand matrix. In the scenario of volumetric data communication with frequent reconfigurations (e.g., datacenter networks and vehicle networks), D could be the traffic bandwidth demands between nodes. For sparse communication such as wireless sensor networks and satellite networks, D could also be the communication frequency between two nodes.

Output: We have to output an undirected graph G, which is the network topology we design.

Objective function: To better illustrate the design of HierTopo, we focus on the optimization of *weighted shortest path length* $\mathrm{Avg}(D_{sd} \cdot l(s, d, G))$: average shortest path length of all traffic flows weighted by their bandwidth demands. Path length is the primary optimization goal in dynamic networks. Nevertheless, HierTopo's design in the following sections could also be employed for other goals (e.g., link load, congestion) or compound goals. We will explain this more specifically and discuss other optimization goals in §VII.

Constraints:

- Connectivity: The output graph G must still be connected.
- Degree limitation: Under our simplification, the degree of each node $\delta_i, \forall i \in V$ should be kept under its limitation $\Delta[i]$. In reconfigurable data centers with 60GHz wireless links, the degree constraint arises from avoiding interference between 60GHz wireless antenna signals. Optical circuit switches can only match with one other such switch, and LEO satellites are permitted to have only 4 to 5 laser inter-satellite links [19]. Δ can be configured according to specific scenarios during deployment.

B. HierTopo overview

The overview of HierTopo is presented in Figure 2. HierTopo only adds one link at a time, updating the topology incrementally. With incremental adjustment, we benefit in terms of a smaller action space, higher topology stability, and more efficient constraint enforcement:

- *Narrowing down the action space.* Originally, to configure the whole topology, we have to decide if there should be a link between each pair of nodes. So the action space is $2^{N(N-1)/2}$ topologies. By incremental adjustment, we break down the sophisticated problem into a sequential

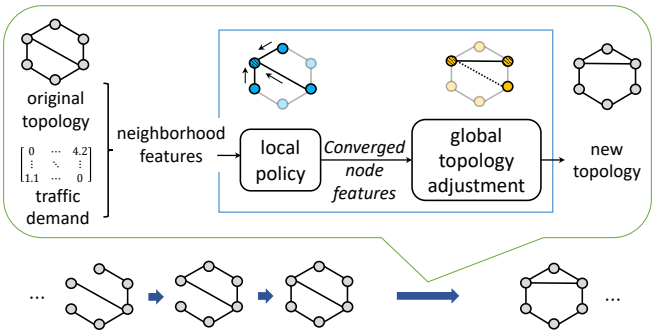


Fig. 2. HierTopo overview.

decision-making process, so we only have to choose one link from at most $N(N-1)/2$ positions at a time. Later, we further narrow down the action space by introducing node features as our decision variables.

- *Maintaining topology stability.* Simultaneous changes involving many links in the topology lead to high flow migration cost and routing instability. HierTopo's step-by-step link adjustment can keep the switching costs acceptably low.
- *Ensuring validity.* The decisions we make should strictly follow the constraints such as degree limitations and network connectivity (the connectivity could be threatened by possible link removing). However, it is easy for us to mask out the invalid positions at each step.

For every topology adjustment (shown in the green box in Figure 2), HierTopo carries out 2 steps hierarchically:

Step 1: Node feature aggregation by local policy. For each node of an existing topology G, HierTopo iteratively calculates its feature based on information from its neighbors. The iteration process continues for several turns to let the node features converge. We present HierTopo's local policy iteration method in §IV.

Step 2: Topology adjustment by a global algorithm. Under the instruction of the converged node features in **Step 1**, we employ a global algorithm to make the topology adjustments. If there is no more space for improvement, HierTopo terminates the topology adjustment. We introduce the adjustment algorithm and termination conditions in §V in detail.

IV. Node feature aggregation by local policy

Before starting to design the local policy, we first discuss the design principle. Aimed at high performance, we inevitably need to optimize the local policy. The optimization overhead, however, might violate the efficiency requirements. Therefore, as a design principle, the local policy is expected to be independent of the network scale or workload to reinforce the generalization ability. If HierTopo can generalize to other-scale networks or other workload distributions, then optimizating the local policy becomes an once-for-all effort and the re-optimization overhead is eliminated.

In this section, we first introduce a new decision variable: node feature in §IV-A. Then, we present the design of the local policy in §IV-B, including embedding inputs, iteratively applying the local policy, and optimizing the local policy. We further demonstrate the optimality of our hierarchical framework in §IV-C.

A. Node features as decision variables

As introduced in §II-C, a primary challenge for the topology optimization problem is that the action space is exponentially large. By incremental adjustment, HierTopo constructs the topology step-by-step and only need to choose a link position in each step, with the action space growing at the speed of $N(N-1)/2$. To ensure that the outputs of the local policy on N nodes can cover the entire action space, the local policy should output with at least $(N-1)/2$ dimensions. $O(N)$ dimension does not reconcile with the local policy's design principle that it should be independent of the network scale.

To address this issue, we introduce a new decision variable called node feature. Each node gets a real number as its feature. Node features are initialized according to the demand matrix and iteratively updated by the local policy (the process is illustrated in Algorithm 1). By our definition, a greater difference between the features of two nodes indicates a higher traffic demand or a longer distance. Based on node features, we can determine the link position to add a link: a new link should be built between the pair of nodes with the largest feature difference to serve higher demands and bridge longer distances. Thus, by trading off the discreteness of the action space, we fix the output dimension of the local policy while maintaining its action expressivity.

B. Node feature aggregation

As mentioned above, after reducing the action space, we are responsible for obtaining node features. Therefore, we introduce how to acquire the node features by input embedding and local policy iteration.

Input embedding. Before applying HierTopo policy, we have to collect and embed relevant information as the inputs. According to the problem formulation in §III-A, the operators should feed the demand matrix between each pair of nodes to HierTopo as the input, and also configure the degree limit of each node. HierTopo enforces the degree limitation during the topology adjustment phase in §V, and embeds the demand matrix into the network topology as the initial node features (Line 1 of Algorithm 1).

However, the demand matrix has N^2 dimensions, so the input of the local policy is also troubled by the N-related dimension, since the $O(N)$-increasing state space contradicts the local policy's independence of the network scale N. In response, we break down the demand matrix by column (each column represents traffic demand to the same destination) and process the local policy for each column in parallel before averaging. The underlying assumption here is that every destination of network flows is equally important.

Local policy iteration. After initializing the node features with the demand matrix, HierTopo is aware of the traffic

Algorithm 1: Node Feature Aggregation.

Input: Traffic demand matrix D, current topology G.
Output: Node features $\boldsymbol{x}^M$ as our decision variables.

1 $\boldsymbol{x}^0 \leftarrow$ Embedding(D)
2 **for** *i=1:M* **do**
3 **for** *each network node* u **do**
4 $\boldsymbol{x}_u^i \leftarrow \text{LocalPolicy}(\boldsymbol{x}_u^{i-1}, \{\boldsymbol{x}_v^{i-1} | v \in \mathbb{N}(u)\})$
5 **end**
6 **end**

demand. Aside from demand, topological information is also required since HierTopo needs to make incremental adjustments based on the current topology. However, the local policy only interacts with the neighboring nodes, so it is difficult for it to capture the connection relations outside the neighborhood.

To cope with this problem, we iteratively apply the local policy so that the feature of a distant node can also be aggregated along a path after several iterations. As shown in Line 2-5 in Algorithm 1, each network node gathers its own feature and the features of its neighboring nodes, applies the local policy to update its feature and repeats the process for M iterations. M is the maximum iteration number. Intuitively, M is related to the degree of separation and is bounded by $N-1$, which is the longest path length between two connected nodes so the feature of any node in a connected topology can be aggregated to the current node within $N-1$ iterations.

Policy choices and optimization. After aggregating both the traffic demand and the topological information, the remaining task is to find the local policy. Unfortunately, we cannot determine the local policy based on existing methods. Among the existing methods reviewed in §II-B, the sophisticated methods (e.g., ILP and convolutional neural networks) do not have an equivalent distributed algorithm. The heuristic methods, though can be transformed into a distributed algorithm, have unsatisfactory performance. It is hardly convincing that we can get a high-performance local policy from these methods.

Unassisted by domain-specific knowledge and inspired by the data-driven solution in the machine learning community, we construct a general model and optimize its coefficients by "trial-and-error". Operationally, we choose a polynomial function as the local policy, which is formulated as:

$$\tau(v) = \left[x_v^{k-1}, x_v^{k-2}, \cdots, x_v^0\right] \cdot \hat{\alpha} + \left[\sum_{j\in\mathbb{N}(v)} x_j^{k-1}, \sum_{j\in\mathbb{N}(v)} x_j^{k-2}, \cdots, \sum_{j\in\mathbb{N}(v)} x_j^0\right] \cdot \alpha$$

, where k is the order of the polynomial function, and both $\hat{\alpha}$ and α are k-dimension coefficient vectors to be optimized. In our implementation, we set the polynomial order as 3 and iterate the the local policy for N times. We show that a polynomial function is expressive enough and analyze the sensitivity of these hyperparameters in §VI-E1.

However, due to the nonsmooth optimization object (§III-A), we cannot directly adopt the common optimization methods for graph neural networks. To resolve this challenge, we employ the Genetic Algorithm (GA) to optimize the polynomial coefficients $\hat{\alpha}$ and α. Genetic Algorithm effectively searches the coefficients by stochastic gene mutation and crossover of population members, meantime, does not require a smooth objective function. In our GA implementation, we use single point crossover, set the population size as 50, and mutate 10% of genes in each generation.

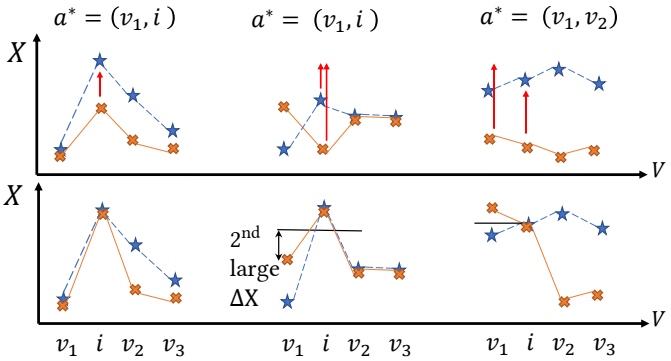


Fig. 3. Finding a function to output the optimal node features. Better viewed in color.

C. Theoretical analysis

We have described how HierTopo offloads topology decisions to each network node from the operational aspect. Now we try to discuss whether the optimal topology can be theoretically achieved.

We denote the feature of node i as x_i (the converged feature as X_i), and a topology adjustment decision (to add a new link between a pair of nodes) as $a = (n_1, n_2)$. Our theoretical analysis below shows the optimality of HierTopo's hierarchical framework.

Theorem 1: There always exists a function that is capable of mapping any legitimate input to node features Xs that can lead to the best topology action a^*.

Proof: Firstly, given any topology action $a = (u, v)$, we can always construct a set of node features that lead to a (e.g., by assigning the minimum and maximum feature to u and v respectively). Thus, a mapping (denoted as ϕ) from any legitimate input to node features Xs that lead to a^* always exists. The question is whether the mapping could be a function.

By definition, a function maps the same input to the same output. If ϕ is already a function, we have found what is requested. If not, we are going to demonstrate that we could always find a function by adjusting the output features of ϕ without affecting the final topology action. Assume that ϕ outputs different node features X^1 and X^2 given the same input. By our definition of ϕ, both X^1 and X^2 lead to the optimal action a^*, but X^1 and X^2 are not exactly same. Assume X^1 and X^2 differs on node i, $X_i^1 \neq X_i^2$. Without loss of generality, we assume $X_i^1 > X_i^2$. We plot examples for better illustration in Figure 3. The blue marks represent the values of X^1 on all the nodes respectively, and the orange marks represent X^2. In the first row of the figure, both X^1 and X^2 can lead to a^* (tagged at the top of the figure), but they contradict on an arbitary node i. We can tune them to the second row to fix the conflict as follows:

Algorithm 2: Adjust: Topology adjustment.

Input: current network graph G, node features $\boldsymbol{x}$, action candidate set $\mathbb{C}$
Output: network topology G^* (after adding or removing links from G)

1 Find $(u, v) \in \mathbb{C}$ so that $|\boldsymbol{x}_u - \boldsymbol{x}_v| \geq |\boldsymbol{x}_i - \boldsymbol{x}_j|, \forall (i, j) \in \mathbb{C}$
2 **if** $\delta_u < \Delta$ *and* $\delta_v < \Delta$ **then**
3 $G^* \leftarrow G.add\_edge(u, v)$ `// unsaturated u and v`
4 **else**
5 $G' \leftarrow G$
6 **if** $\delta_u = \Delta$ **then**
  `// remove one of u's links`
7 Find $u' \in \mathbb{N}(u)$ so that $|\boldsymbol{x}_u - \boldsymbol{x}_{u'}| \leq |\boldsymbol{x}_u - \boldsymbol{x}_i|, \forall i \in \mathbb{N}(u)$
8 $G' \leftarrow G'.remove\_edge(u, u')$
9 **end**
10 **if** $\delta_v = \Delta$ **then**
  `// remove one of v's links`
11 Find $v' \in \mathbb{N}(v)$ so that $|\boldsymbol{x}_v - \boldsymbol{x}_{v'}| \leq |\boldsymbol{x}_v - \boldsymbol{x}_i|, \forall i \in \mathbb{N}(v)$
12 $G' \leftarrow G'.remove\_edge(v, v')$
13 **end**
14 $G' \leftarrow G'.add\_edge(u, v)$
15 $\boldsymbol{x}' \leftarrow$ NodeFeatureAggregation(D, G')
16 $s = |\boldsymbol{x}_u - \boldsymbol{x}_v| + |\boldsymbol{x}_u - \boldsymbol{x}_{u'}| + |\boldsymbol{x}_v - \boldsymbol{x}_{v'}|$
17 $s' = |\boldsymbol{x'}_u - \boldsymbol{x'}_v| + |\boldsymbol{x'}_u - \boldsymbol{x'}_{u'}| + |\boldsymbol{x'}_v - \boldsymbol{x'}_{v'}|$
18 **if** $s \geq s'$ *and* G'*is connected* **then**
19 $G^* \leftarrow G'$ `// remove links for (u,v)`
20 **else**
21 $G^* \leftarrow G$ `// unworthy, don't change`
22 **end**
23 **end**
24 $\mathbb{C} \leftarrow \mathbb{C}/(u, v)$ `// don't reconsider (u,v)`

(1) In a simple case, if X_1 still leads to a^* after assigning X_i^2 to X_i^1 (or reversely, X_2 still leads to a^* after assigning X_i^1 to X_i^2), we can do the assignment and resolve this conflict (See the left of Figure 3). Yet, if assigning one to the other changes the overall topology action, there are two other possible situations:

(2) node i is in the selected node pair, e.g., $a^* = (v_1, i)$. In this case (the middle of Figure 3), we can adjust both X_i^1 and X_i^2 to a value above the maximum (or below the minimum) of features on other nodes (aside from i). The new value should exceed the maximum (or the minimum) by more than the second largest feature difference among other nodes, so that the adjusted node features still lead to a^*.

(3) node i is not in the selected node pair, but assigning X_i^2 to X_i^1 makes the feature difference between i and another node become the largest (the right of Figure 3). In this case, we can first assign X_i^1 to X_i^2, then raise the originally highest feature in X_2 above X_i^2.

In either way, node features can be tuned into determined values without changing the optimal action, proving that a function leading to the optimal topology decisions exists. ■

V. Topology adjustment by a global algorithm

In this section, we introduce the topology adjustment algorithm of HierTopo.

As introduced in §IV, we can calculate node features $\boldsymbol{x}$ with our local policy and choose the pair of nodes with the largest difference of $\boldsymbol{x}$ to add an edge. However, in practice, there are still two problems: 1) the node pair we select might be invalid. For example, there may already exist a link between them, or one of the nodes has met the degree limitation (§III-A); 2) the incremental adjustment could repeat forever, so we need to judge when to finish.

We now describe our algorithm that solves the above problems and constructs a better topology over an existing one. Before beginning our adjustment, we collect action candidates into a set $\mathbb{C}$ containing all node pairs in the network that are not directly linked. Then, given the traffic demand matrix, we calculate node features based on current topology. When we attempt to add a link between the greatest feature difference node pair (u, v) (among the node pairs in $\mathbb{C}$) and both u and v have residual degree, we add a link between u and v and remove (u, v) from $\mathbb{C}$ (Line 2-3 in Algorithm 2).

However, consider at least one of u and v is saturated (i.e., already connected by Δ links). Assume u is saturated, we find the link of the least importance (u, u'), that u' has the smallest feature difference with u (Line 6-7). Before trading (u, u') for (u, v), we need to ensure that the benefit of building up (u, v) outweighs the cost of removing (u, u') (or both if v is also saturated). To hold a fair competition between (u, v) and (u, u'), we first calculate node features $\boldsymbol{x}'$ under a hypothetical topology G' where (u, v) is added and (u, u') is removed (Line 15). Then we compare the sum of the feature differences of (u, v) and (u, u') under topology G and G' respectively. Smaller feature difference under G' indicates that it is worthwhile to remove (u, u') for (u, v) (Line 17). Otherwise we do not change the topology (Line 19). In both cases, remove (u, v) from $\mathbb{C}$ (Line 22 in Algorithm 2). We finish our adjustment when there is no candidate left in $\mathbb{C}$.

In this algorithm, we use the candidate set $\mathbb{C}$ to mask out existing links. Since we never remove a link if the resulting topology becomes disconnected and never add a link to a saturated node before removing one of its links, the constraints in §III-A are strictly enforced.

VI. Evaluation

In this section, we first introduce our evaluation settings (§VI-A) and evaluate HierTopo from the following aspects:

- *Performance.* Our results show that HierTopo significantly outperforms existing heuristic algorithms by up to 40% and has similar or better performance against optimization-based algorithms (§VI-B).
- *Generalization ability.* We demonstrate that HierTopo's local policy can generalize to different-scale networks: even optimized at an 8-node network, HierTopo's performance degradation during scaling up to 50 nodes can be maintained within 3%. We also show that HierTopo can generalize well to different traffic distributions (§VI-C).
- *Overhead.* We further evaluate the optimization time and runtime of HierTopo. Experiments show that HierTopo can

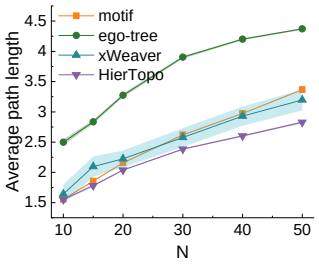


Fig. 4. Overall performance. Better viewed in color.

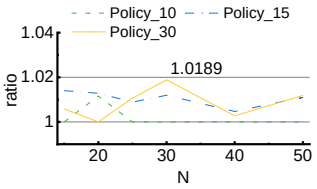
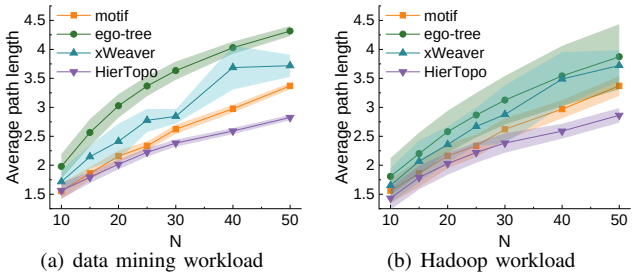


Fig. 5. Normalized performance of different HierTopo policies.



(a) data mining workload (b) Hadoop workload

Fig. 6. Traffic generalization ability. Better viewed in color.

efficiently construct a topology with tens of nodes within tens of microseconds (§VI-D).

- *Microbenchmarking.* Finally, we demonstrate that HierTopo could achieve satisfactory performance in a wide range of hyperparameter settings and achieve a small optimality gap within limited adjustment steps (§VI-E).

A. Experimental setup

1) Network traces: To better characterize real-world traffic, we generate our demand matrices based on the datasets of real world flow distribution of cache workload in Facebook [20]. Under this distribution, we generate 1000 demand matrices as our testing dataset to eliminate the randomness. We further use two different distributions from the real world to evaluate the generalization ability of HierTopo in §VI-C2. The degree limit (Δ) is configured as 4 in our experiment, but it can adapt to specific scenarios accordingly.

2) Baselines: To compare the performance of HierTopo, we also implement several baselines, including:

- *Motif*: an optimization-based algorithm by searching the optimal regular repetitive patterns [4].
- *Ego-tree*: a heuristic-based algorithm by generating a tree-like topology sorted by the traffic demand [17].
- *xWeaver*: a deep learning model based on convolutional neural networks by scoring the demand matrix and finding the topology with maximum scores [15].

3) Testbed Setup: We conduct our experiments on a server with an Intel Core i7-8700 CPU (6 physical cores, 12 virtual cores). xWeaver is trained on a Nvidia 2080 Ti GPU.

B. Performance

We evaluate the performance of a topology by the average shortest path length in the topology weighted by traffic bandwidth demand (denoted as `average path length` in short). We first test the performance of HierTopo among the baseline algorithms in networks with different scales (ranging from 8 to 50 nodes). The network topology has no link initially before we construct the whole topology via different methods.

As shown in Figure 4, the average shortest path lengths in the topologies designed by HierTopo are kept below all other methods. Compared to the heuristic method ego-tree, HierTopo improves its performance by 38.91% in 50-node networks. Motif and xWeaver, on the other hand, are better optimized but HierTopo also improves their performance by up to 11.52%. The standard deviation of HierTopo is very small whereas xWeaver suffers larger performance fluctuations.

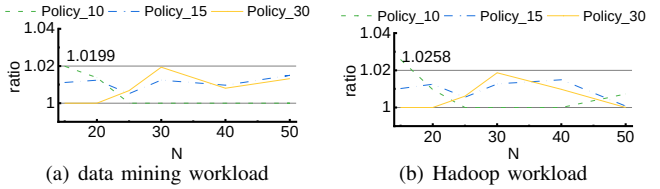
C. Generalization ability

We highlight the generalization advantages of HierTopo over baseline methods: HierTopo can generalize to different-scale networks and different traffic distributions with negligible performance degradation.

1) Generalization over network scale: To illustrate the generalization ability over different network scales, we directly apply the local policy optimized under small networks to larger networks. Specifically, we first optimize the local policy with GA in 10-, 15-, and 30-node networks (denoted as `Policy_10`, `Policy_15` , and `Policy_30` respectively). Then we apply these policies on different-size networks with 15 to 50 nodes. We plot the ratio of the performance (average path length) under each policy to the best performance among all these policies (denoted as `ratio` in short).

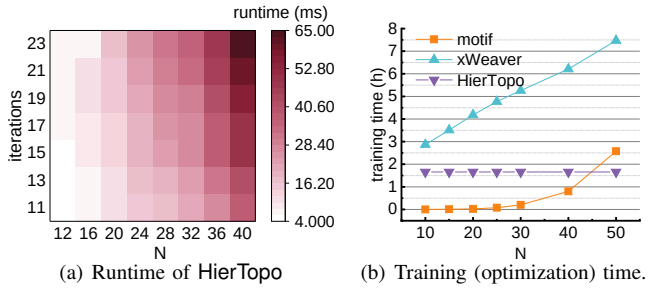
From Figure 5, we observe that old policies have at most 1.89% performance degradation compared to the optimized policy. In rare cases, a policy is outperformed by another policy optimized under a smaller network. This might attribute to the inherent randomness of the stochastic mutations during genetic algorithm optimization. Nonetheless, applying the local policy to an unseen-size network, the performance differences are less than 2% for networks with up to 50 nodes. Thus, HierTopo can generalize to networks of different sizes without re-optimizing the local policy.

2) Generalization over workload distribution: We evaluate HierTopo's performance under another 2 input traffic distributions: a data mining workload [21] in Microsoft and a Hadoop workload [20] in Facebook. Under each distribution, we also generate 1000 traffic matrices for our evaluation. For HierTopo and all the other baselines, instead of re-optimizing them under new traffic distributions, we directly employ the policy optimized under the Facebook cache workload. As plotted in Figure 6, under unseen workloads, HierTopo still achieves shorter and slowlier-increasing average path lengths than other baselines. In 50-node networks, we highlight that HierTopo enhances the performance by 16.13-34.55% under the data mining workload and 15.04-26.09% under the Hadoop workload, and the standard deviation of HierTopo is also smaller. Again, we measure the performance ratio (defined in §VI-C1) under data mining and Hadoop workloads. Figure 7 demonstrates that under both workloads, the performance degradations of old policies are no more than 2.58%. It indicates that HierTopo can generalize to unseen workloads without losing its performance advantage.



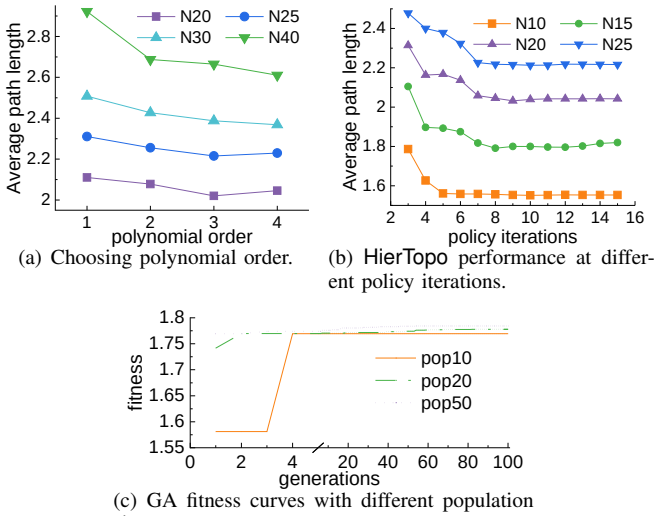
(a) data mining workload (b) Hadoop workload

Fig. 7. Normalized performance of policies under unseen traffic distributions.



(a) Runtime of HierTopo (b) Training (optimization) time.

Fig. 8. Time overhead.



(a) Choosing polynomial order. (b) HierTopo performance at different policy iterations.

(c) GA fitness curves with different population size.

Fig. 9. Parameters sensitivity and convergence efficiency.

D. Overhead

We measure the time overhead of HierTopo. We first measure the runtime for HierTopo to complete the topology construction from an empty graph. The runtime is affected by the number of local policy iterations (the times we apply the local policy, denoted as `iterations` in the figure) and the network scale N. As demonstrated in Figure 8(a), the network size is the dominant factor. Through fitting analysis, the runtime increases with the square of N and linearly with the number of policy iterations. Results show that it takes less than 65 milliseconds to construct a topology from scratch for 40-node networks by iterating the policy for 22 times.

We then measure and plot the optimization time of HierTopo in Figure 8(b) (1.7h under 8-node networks with 10 policy iterations and 50 as the GA population size). As discussed in §VI-C, HierTopo can be transferred to other network scales without reoptimization thanks to its generalization ability. In contrast, both Motif and xWeaver need retraining/reoptimizing whenever the size of the network changes. It takes about 7.5 hours for xWeaver to train to convergence under 50-node network. Meanwhile, their training time increases sharply as the network scales up.

E. Microbenchmarks

In this section, we present the summary of experiments on the sensitivity of hyperparameters in HierTopo (§VI-E1), the convergence efficiency of local policy iteration (§VI-E2) and the optimality gap during HierTopo's incremental adjustments (§VI-E3).

1) Hyperparameter sensitivity: We first evaluate the sensitivity of the order of the polynomial function of our local policy. Higher-order polynomial policies have stronger expressiveness, but they consume longer time to be applied and optimized (since there are more coefficients in the function). To find the proper polynomial order, we test HierTopo's performance by varying the polynomial order from 1 to 4. As presented in Figure 9(a), the performance is well enough even at a low order of 3 or 4, so the network operators only have to configure a simple policy on each network device.

We then evaluate the influence of the population size of the Genetic Algorithm. As a randomized search algorithm mimicking the process of revolution, GA maintains a population (i.e., a cluster of individuals with candidate solutions embedded in their genes) and evolves through generations according to the fitness function (i.e., "survival of the fittest"). We set the fitness function inversely proportional to the average path length in the topology. We plot the evolutionary curves of GA optimization under a 10-node network in Figure 9(c). Results show that a bigger population size enables convergence at earlier generations, but the fitness converges to the same level under different population sizes, so the performance of our policy is not sensitive to the GA population size.

2) Convergence efficiency: As any other iterative algorithm does, the convergence efficiency of HierTopo contributes to its decision latency. The convergence efficiency could be reflected by the number of local policy iterations. If we iterate the local policy too few, the node features cannot converge and will undermine the overall performance. If we iterate the policy for too many times after the convergence, the performance generally levels off. To reveal the exact converging iteration number, we evaluate the performance of HierTopo when applying the local policy for various iterations in Figure 9(b) (`N10`, `N15`, `N20`, `N25` represent the network has 10, 15, 20, and 25 nodes respectively). As observed from the figure, as the network size grows from 10 to 25, the converging iteration number only increases from 5 to 7, and the increase gradually slows down. Thus, in reality, we only have to iterate the policy for a few times (much smaller than the network size N).

Although HierTopo converges within linear iterations in our experiments, we admit the uncertainty rooted in the iterative process. Therefore, we also suggest referring to the principle of iterative algorithm once confronted with convergence issues. For example, if the local policy iteration oscillates before convergence, we can update by a smaller "learning rate" (i.e.,

original topology	NSF	Geant2	Germany
1-step optimality gap	1.00%	1.82%	1.81%

TABLE I
OPTIMALITY GAP OF HIERTOPO WITH 1-STEP TOPOLOGY ADJUSTMENT.

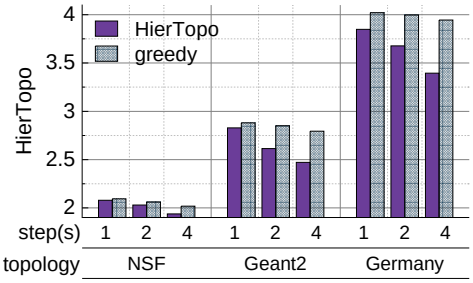


Fig. 10. Topology adjustment within 1, 2, and 4 steps.

discounting α) to eliminate the oscillation.

3) Local policy iteration deep dive: Questioning the optimality of HierTopo policy, we compare our performance of 1-step topology adjustment against the optimal results. Three real-world networks are taken as the original topology: the 14-node NSF network, the 24-node Geant2 network, and the 50-node Germany network [22]. The optimal results are exhaustively searched among all possible actions. As shown by the results in Table I, the optimality gap of HierTopo for one step adjustment is less than 2% in all the three networks, primarily verifying that HierTopo can make near-optimal decisions for each step of topology adjustment.

We evaluate the stability of HierTopo's performance by using HierTopo to adjust the original topology within limited steps (1, 2, and 4 steps). The original topologies are still NSF, Geant2, and the Germany network. Since the baseline algorithms above are not suitable for incremental adjustment, we implement a greedy algorithm that always tries to link the two nodes with the highest traffic demand for comparison. The results in Figure 10 show that HierTopo progressively and efficiently improves the original topology and enhances the performance by 13.93% over the naive greedy method with only 4-step adjustments in the Germany network.

VII. DISCUSSION

In this section, we discuss the limitations of HierTopo and highlight several potential future research directions.

Real-world deployment. In this paper, we mainly focus on the algorithmic side of HierTopo. In the real-world deployment, operators could deploy the HierTopo in two ways. For a dynamic network with a centralized controller (e.g., a ground station in the satellite network), operators could collect the statistics from each network node and make topology adjustments with HierTopo. For distributedly deployed scenarios (e.g., vehicle network), the iteration of the local policy could also be distributedly deployed on each node. In this paper, we evaluate HierTopo with real-world traces. In the future, we plan to deploy HierTopo in practice to improve the performance of real-world applications.

Various network requirements. When deploying HierTopo in practice, operators might confront various network requirements other than the problem formulated in §III-A. For example, network operators might want to minimize operating expenses [23] or energy consumption [24], which need further formulation on the optimization goal. Other requirements, such as the capacity limitation of a reconfigurable link, could be fulfilled by jointly considering the routing optimization and topology optimization. We leave further optimization on other network requirements as our future work.

Other objectives of topology optimization. In this paper, HierTopo is targeted at optimizing the path length in dynamic networks. On one hand, the path length is the primary concern in the application scenarios of dynamic networks according to their need for low latency. Path length is also set up as the objective in prior works [4, 8, 17]. On the other hand, goals on the transport layer (e.g., link load, congestion), could be handled by the optimization of routing strategies [14], which is orthogonal to our research scope. Moreover, HierTopo can adapt to other objectives since the Genetic Algorithm makes no assumption on the optimization goal, thus HierTopo is agnostic to the specific objective by design.

VIII. RELATED WORK

Centralized vs. distributed methods in networking. The classification between centralized and distributed methods is not rare in the networking domain. For example, in the routing domain, distributed routing algorithms (e.g., link-state routing algorithm [25]) make all routers run the same algorithm (e.g., Dijkstra Algorithm) in parallel and only leverage routing states from neighboring routers. As a result, the distributed routing algorithm is quick and more adaptive to network dynamics. Multiple traffic engineering works [26–28], on the other hand, optimize the routing configurations in a centralized way to serve sophisticated traffic engineering goals (e.g., fault tolerance, link utilization) and take longer computation time correspondingly. To balance between centralization and decentralization, [29] hierarchically distributes route calculation to designated nodes. [29] is hierarchical among network nodes, in contrast, network nodes are all equal in HierTopo whose hierarchy is algorithmic.

The trade-off between performance and efficiency. The trade-off between performance and efficiency is a long-lasting problem in the networking community for decades. For example, at the transport layer, widely deployed congestion control algorithms only require simple decision logic [30, 31] while the performance-oriented algorithms proposed by the academia seek to achieve better performance via sophisticated optimization [32, 33]. Similar trade-offs could also be observed in the network layer (e.g., distributed routing [25] vs. centralized traffic engineering [26]) and application layer (e.g., the trade-off between the performance and complexity of adaptive bitrate algorithms [34]). HierTopo is inspired by the research efforts above and designed to tackle the domain-specific challenges in the topology optimization of dynamic networks.

Other research efforts focused on dynamic networks. A few other works focused on dynamic network topology design mentioned as follows are not used as our baselines, because they are very similar to one of our baselines thus could be represented. As a counterpart of xWeaver [15] in wireless sensor networks, DRL-TC [3] employs deep reinforcement learning to optimize the network topology, but also requires retraining as the network changes. Upon topology design for LEO satellite network, both +Grid [35] and CubeSats matching [36] build satellite topologies with naively repetitive patterns, so their performance is bested by motif [4], a baseline that searches for the optimal pattern to repeat throughout the constellation. In satellite networks, more research efforts focus on optimizing trajectories [37] and routing [19, 38], which are orthogonal to the scope of HierTopo.

IX. Conclusion

This paper proposes HierTopo, a hierarchical method of the topology optimization for dynamic networks. HierTopo distributedly runs a polynomial local policy and aggregates these local features to make global topology adjustments. We show the optimality of this hierarchical framework with theoretical analysis. Evaluations show that HierTopo efficiently constructs high-performance topologies and has good generalization ability compared to state-of-the-art solutions.

Acknowledgment

We sincerely thank Yunfei Li and anonymous IWQoS'21 reviewers for their valuable suggestions on this paper. The research is supported by the National Key R&D Program of China under Grant 2017YFB0801701 and the National Natural Science Foundation of China under Grant 61625203 and 92038302. Mingwei Xu is the corresponding author.

References

[1] M. Ghobadi, R. Mahajan, A. Phanishayee, N. Devanur, J. Kulkarni *et al.*, "Projector: Agile reconfigurable data center interconnect," in *Proc. ACM SIGCOMM*, 2016.

[2] X. Zhou, Z. Zhang, Y. Zhu, Y. Li, S. Kumar *et al.*, "Mirror mirror on the ceiling: Flexible wireless links for data centers," in *Proc. ACM SIGCOMM*, 2012.

[3] X. Meng, H. Inaltekin, and B. Krongold, "Deep reinforcement learning-based topology optimization for self-organized wireless sensor networks," in *Proc. IEEE GLOBECOM*, 2019.

[4] D. Bhattacherjee and A. Singla, "Network topology design at 27,000 km/hour," in *Proc. ACM CoNEXT*, 2019.

[5] "Fcc starlink filing," https://www.fcc.gov/document/fcc-authorizes-spacex-provide-broadband-satellite-services, 2018.

[6] I. Cho, A. Saeed, J. Fried, S. J. Park, M. Alizadeh, and A. Belay, "Overload control for μs-scale rpcs with breakwater," in *Proc. USENIX OSDI*, 2020, pp. 299–314.

[7] M. K. Mukerjee, C. Canel, W. Wang, D. Kim, S. Seshan, and A. C. Snoeren, "Adapting tcp for reconfigurable datacenter networks," in *Proc. USENIX NSDI*, 2020, pp. 651–666.

[8] K.-T. Foerster, M. Ghobadi, and S. Schmid, "Characterizing the algorithmic complexity of reconfigurable data center architectures," in *Proc. ACM ANCS*, 2018.

[9] N. Hamedazimi, Z. Qazi, H. Gupta, V. Sekar, S. R. Das *et al.*, "Firefly: A reconfigurable wireless data center fabric using free-space optics," in *Proc. ACM SIGCOMM*, 2014.

[10] W. M. Mellette, R. McGuinness, A. Roy, A. Forencich, G. Papen, A. C. Snoeren, and G. Porter, "Rotornet: A scalable, low-complexity, optical datacenter network," in *Proc. ACM SIGCOMM*, 2017.

[11] G. Porter, R. Strong, N. Farrington, A. Forencich, P. Chen-Sun, T. Rosing, Y. Fainman, G. Papen, and A. Vahdat, "Integrating microsecond circuit switching into the data center," in *Proc. ACM SIGCOMM*, 2013.

[12] Y. Cui, S. Xiao, X. Wang, Z. Yang, C. Zhu, X. Li, L. Yang, and N. Ge, "Diamond: Nesting the data center network with wireless rings in 3d space," in *Proc. USENIX NSDI*, 2016.

[13] A. Boyle, "Amazon's project kuiper aims to offer satellite broadband access - geekwire," https://www.geekwire.com/2019/amazon-project-kuiper-broadband-satellite, 2019.

[14] Z. Yang, Y. Cui, S. Xiao, X. Wang, M. Li *et al.*, "Achieving efficient routing in reconfigurable dcns," *Proc. ACM Sigmetrics*, 2020.

[15] M. Wang, Y. Cui, S. Xiao, X. Wang, D. Yang *et al.*, "Neural network meets dcn: Traffic-driven topology adaptation with deep learning," *Proc. ACM Sigmetrics*, 2018.

[16] M. Bienkowski, D. Fuchssteiner, J. Marcinkowski, and S. Schmid, "A competitive b-matching algorithm for reconfigurable datacenter networks," in *Proc. IFIP Performance*, 2020.

[17] C. Avin, K. Mondal, and S. Schmid, "Demand-aware network design with minimal congestion and route lengths," in *Proc. IEEE INFOCOM*, 2019.

[18] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot *et al.*, "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[19] M. Handley, "Delay is not an option: Low latency routing in space," in *Proc. ACM HotNets*, 2018.

[20] A. Roy, H. Zeng, J. Bagga, G. Porter, and A. C. Snoeren, "Inside the social network's (datacenter) network," in *Proc. ACM SIGCOMM*, 2015.

[21] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan, "Data center tcp (dctcp)," in *Proc. ACM SIGCOMM*, 2010, pp. 63–74.

[22] "Kdn training datasets." http://www.knowledgedefinednetworking.org/, 2019.

[23] C. S. Yeo and R. Buyya, "Service level agreement based allocation of cluster resources: Handling penalty to enhance utility," in *2005 IEEE International Conference on Cluster Computing*. IEEE, 2005.

[24] R. C. Shah and J. M. Rabaey, "Energy aware routing for low energy ad hoc sensor networks," in *2002 IEEE Wireless Communications and Networking Conference Record. WCNC 2002*. IEEE, 2002.

[25] J. Moy, "OSPF Version 2," RFC 2328, 1998.

[26] T. Benson, A. Anand, A. Akella, and M. Zhang, "Microte: Fine grained traffic engineering for data centers," in *Proc. ACM CoNEXT*, 2011.

[27] H. H. Liu, S. Kandula, R. Mahajan, M. Zhang, and D. Gelernter, "Traffic engineering with forward fault correction," in *Proc. ACM SIGCOMM*, 2014, pp. 527–538.

[28] P. Kumar, Y. Yuan, C. Yu, N. Foster, R. Kleinberg, P. Lapukhov, C. L. Lim, and R. Soulé, "Semi-oblivious traffic engineering: The road not taken," in *Proc. USENIX NSDI*, 2018, pp. 157–170.

[29] R. E. Ali, B. Erman, E. Baştuğ, and B. Cilli, "Hierarchical deep double q-routing," in *Proc. IEEE ICC*, 2020.

[30] S. Ha, I. Rhee, and L. Xu, "Cubic: a new tcp-friendly high-speed tcp variant," *ACM SIGOPS operating systems review*, pp. 64–74, 2008.

[31] N. Cardwell, Y. Cheng, C. S. Gunn, S. H. Yeganeh, and V. Jacobson, "Bbr: Congestion-based congestion control: Measuring bottleneck bandwidth and round-trip propagation time," *ACM Queue*, pp. 20–53, 2016.

[32] M. Dong, T. Meng, D. Zarchy, E. Arslan, Y. Gilad, B. Godfrey, and M. Schapira, "Pcc vivace: Online-learning congestion control," in *Proc. USENIX NSDI*, 2018.

[33] S. Abbasloo, C.-Y. Yen, and H. J. Chao, "Classic meets modern: A pragmatic learning-based congestion control for the internet," in *Proc. ACM SIGCOMM*, 2020.

[34] Z. Meng, Y. Guo, Y. Shen, J. Chen, C. Zhou *et al.*, "Practically deploying heavyweight adaptive bitrate algorithms with teacher-student learning," *IEEE/ACM Transactions on Networking*, 2021.

[35] L. Wood, "Internetworking with satellite constellations," Ph.D. dissertation, University of Surrey, 2001.

[36] B. Soret, I. Leyva-Mayorga, and P. Popovski, "Inter-plane satellite matching in dense leo constellations," in *Proc. IEEE GLOBECOM*. IEEE, 2019, pp. 1–6.

[37] T. W. Beech, S. Cornara, M. B. Mora, and G. Lecohier, "A study of three satellite constellation design algorithms," in *14th international symposium on space flight dynamics*, 1999.

[38] M. Handley, "Using ground relays for low-latency wide-area routing in megaconstellations," in *Proc. ACM HotNets*, 2019.