Neural Network Meets DCN: Traffic-driven Topology Adaptation with Deep Learning

MOWEI WANG, Tsinghua University, China
YONG CUI*, Tsinghua University, China
SHIHAN XIAO, Huawei Technologies, China
XIN WANG, Stony Brook University, USA
DAN YANG, Beijing University of Posts and Telecommunications, China
KAI CHEN, Hong Kong University of Science and Technology, China
JUN ZHU, Tsinghua University, China

The emerging optical/wireless topology reconfiguration technologies have shown great potential in improving the performance of data center networks. However, it also poses a big challenge on how to find the best topology configurations to support the dynamic traffic demands. In this work, we present *xWeaver*, a traffic-driven deep learning solution to infer the high-performance network topology online. xWeaver supports a powerful network model that enables the topology optimization over different performance metrics and network architectures. With the design of properly-structured neural networks, it can automatically derive the critical traffic patterns from data traces and learn the underlying mapping between the traffic patterns and topology configurations specific to the target data center. After offline training, xWeaver generates the optimized (or near-optimal) topology configuration online, and can also smoothly update its model parameters for new traffic patterns. We build an optical-circuit-switch-based testbed to demonstrate the function and transmission efficiency of our proposed solution. We further perform extensive simulations to show the significant performance gain of xWeaver, in supporting higher network throughput and smaller flow completion time.

CCS Concepts: • **Networks** → **Topology analysis and generation**; **Data center networks**; • **Computing methodologies** → *Machine learning*;

Additional Key Words and Phrases: Data Center Networks; Topology Adaptation; Deep Learning

ACM Reference Format:
Mowei Wang, Yong Cui, Shihan Xiao, Xin Wang, Dan Yang, Kai Chen, and Jun Zhu. 2018. Neural Network Meets DCN: Traffic-driven Topology Adaptation with Deep Learning. *Proc. ACM Meas. Anal. Comput. Syst.* 2, 2, Article 26 (June 2018), 25 pages. https://doi.org/10.1145/3224421

*Yong Cui is the corresponding author.

Authors' addresses: Mowei Wang, Tsinghua University, China, wang.mowei@outlook.com; Yong Cui, Tsinghua University, China, cuiyong@tsinghua.edu.cn; Shihan Xiao, Huawei Technologies, China, xiaoshihan@huawei.com; Xin Wang, Stony Brook University, USA, x.wang@stonybrook.edu; Dan Yang, Beijing University of Posts and Telecommunications, China, bupt-steven@foxmail.com; Kai Chen, Hong Kong University of Science and Technology, China, kaichen@cse.ust.hk; Jun Zhu, Tsinghua University, China, dcszj@mail.tsinghua.edu.cn.

2476-1249/2018/6-ART26 $15.00
https://doi.org/10.1145/3224421

1 INTRODUCTION

Data center network (DCN) is the key infrastructure of cloud computing. With the fast growth of large-scale cloud services, the traffic in today's DCNs shows high spatial-temporal dynamics [10, 11, 28, 42, 47]. Conventional wired data centers generally adopt a static network topology and thus have to be overprovisioned to handle the worst-case traffic patterns (e.g., the Clos networks [6, 22]). With the complex topology design and the increasing network scales, data centers are challenged by the significant cost in maintenance, cabling, heat and power consumption [18, 47].

To address the traffic dynamics at low cost, recent work proposes to construct *topology-reconfigurable* DCNs by introducing the new network components such as optical circuit switches (OCS) or wireless radios into the DCNs [13, 14, 16, 18, 19, 23, 24, 33, 35, 36, 41, 51, 53, 55, 56]. Rather than relying on heavy network overprovision, they provide flexible high-bandwidth optical/wireless links to handle the traffic dynamics on demand. Specifically, the agile optical/wireless links can be quickly switched to construct a proper runtime topology to meet the current traffic demands. With such flexibility in changing the network topology, these DCN architectures show the great potential to approach the performance of a fully provisioned network while significantly reducing the complexity and cost in maintenance, cabling and power [18, 24].

The key challenge in supporting the topology-reconfigurable architectures is how to obtain the optimal (or near-optimal) topology configuration for the given traffic demands. Focusing on the *local* link configuration for an OCS switch based on its port demands [18], previous work generally ignores the interactions between OCS switches and the wired network topology. Recent studies [12, 34] show that it is beneficial to adapt the link configuration of an OCS switch along with a single electrical switch. Inspired by these observations, the goal of this work is to construct the best *global* topology to meet the overall traffic demands in a practical DCN. A straightforward method is to model the global interactions between traffic and topology with respect to the optimization objective. However, this is non-trivial as the transmission performance of a topology is affected by many practical system factors, such as the specific routing protocols and congestion control strategies. The modeling becomes more difficult if using the higher-layer application performance as the optimization objective (e.g., the Hadoop job completion time [40]). As shown in the previous work, even if we only consider the simplistic case with one OCS switch and one electrical switch, it often requires solving an integer linear programming (ILP) problem that is unscalable [23, 24, 34] due to the discrete property of topology configurations. The above modeling challenges drive existing work to resort to heuristic solutions that are simple, fast but potentially far away from optimal [16, 18, 23, 24, 51].

To address the above challenges, in this paper, we present *xWeaver*, a traffic-driven deep learning system for the topology configuration in DCNs. The motivation is that the neural network used in deep learning can build up a comprehensive interaction model between traffic and topology automatically with little human efforts. State-of-the-art deep learning technologies, e.g., the convolutional neural network (CNN), are known to be good at learning complex models from data in image processing [45]. Recently, they have made impressive performance breakthroughs in many fields where they are the first to surpass human-level performance (e.g., the image classification [25], game of go [46], etc.). It is also a new trend to apply machine learning techniques on solving the networking problems [52].

Different from the conventional ILP-based modeling, in our system, the neural network does not need to keep solving a complex model online. Instead, in the offline phase, xWeaver uses a specialized neural network to learn and store the critical features of the optimal (or near-optimal) solutions. The parameters of the neural network can be trained offline from the history traces, which are rich and easily available in today's DCNs. The parameters can be also updated with

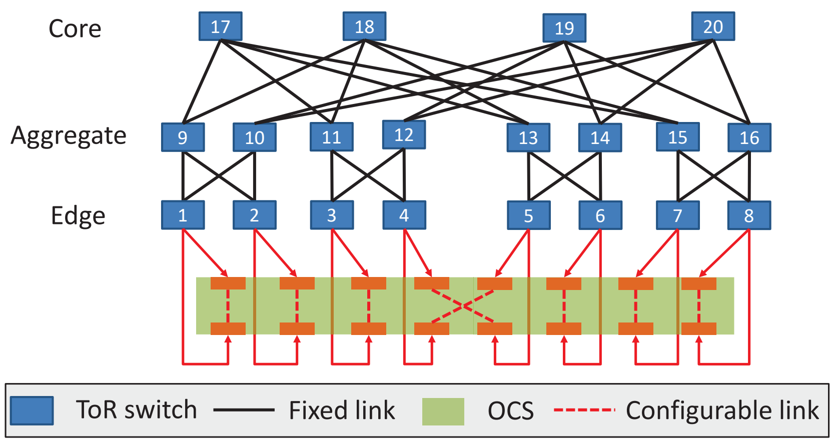


Fig. 1. The example topology case of 4-port Fat-tree.

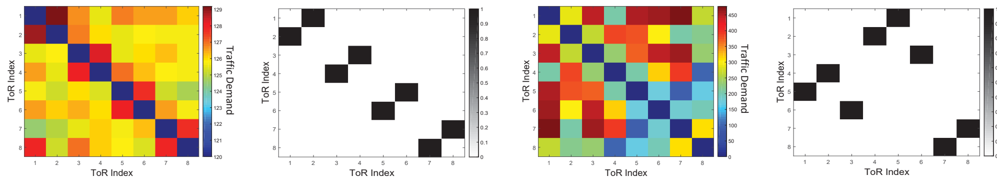
new data available. After the offline training, the topology inference through neural network is generally fast [1], especially supported by the speed-up using the advanced hardwares [26, 50]. Thus it can be easily applied for online topology configuration.

Motivated by the merits of deep learning, in this work, we take the initiative to answer two key questions: *"Can neural networks learn the traffic patterns in DCN?"* and *"Can neural networks learn the global interactions between traffic and topology configuration?"*. In xWeaver, we design two properly-structured neural networks (named the SCNN and FPNN) to address the above questions in details. xWeaver has three key design features:

- **Expressive learning framework (§3)**: Rather than being restricted to simple optimization objectives or specific network architectures, xWeaver provides a framework to support an *expressive* network model. Specifically, it can flexibly support the optimization over both conventional flow-level performance metrics and comprehensive application-level performance metrics in a unified way.
- **Data-driven feature extraction (§4)**: In order to extract the critical features in traffic and topology that are related to the self-defined optimization objective, we design a novel separate-structured convolutional neural network (SCNN) in xWeaver and train its parameters from historical data. With the speed-up of the well-trained SCNN, we propose an efficient method that can *automatically* label high-score topologies for the corresponding traffic demands.
- **Traffic-topology mapping learning (§5)**: We design a traffic-topology-mapping neural network (FPNN) to capture the essential interactions between traffic and topology configurations. FPNN uses a partial structure of SCNN for the traffic feature encoding. Further, we develop a specialized probability graph model to enhance the output performance of FPNN taking advantage of the prior human knowledge about the network architecture.

Our simulations show that xWeaver can support much higher flow performance than conventional solutions under different traffic and topology conditions (§6). We implement an OCS-based testbed and our experimental results confirm the learning efficiency and demonstrate the high performance of xWeaver on handling different traffic demands (§7). Finally, after discussing the practical application issues (§8), we introduce the related work (§9) and draw the conclusions (§10).

[1]To compute the topology output given traffic input, a neural network takes only one weight-forwarding through its neural layers [45].



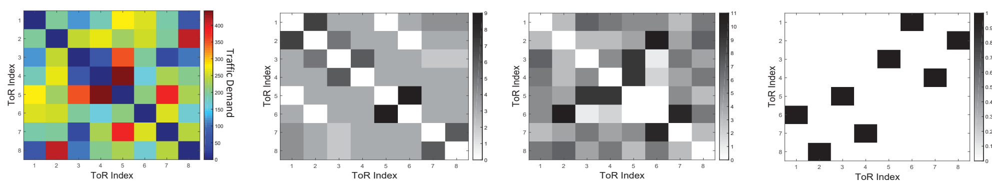
(a) Sum of a class of traffic demand matrices **D**

(b) Topology P_0 that optimally supports the demand class **D**

(c) One sample demand $d \in \mathbf{D}$

(d) Weight-matching topology P_1 for demand d

Fig. 2. An example of traffic pattern attached to a specific topology configuration.



(a) One sample demand $d' \in \mathbf{D}$

(b) Sum of high-score topologies for demand d'

(c) Sum of low-score topologies for demand d'

(d) Weight-matching topology P_2 for demand d'

Fig. 3. An example of the key structure of the high-performance topologies for a specific traffic demand matrix.

2 WHY DEEP LEARNING?

In this section, we will present two key observations to demonstrate the natural advantages and challenges of applying deep learning to solving the global topology configuration problem.

Observation 1. *Each global topology configuration can optimally support a specific class of traffic demands, while each class has a unique distribution pattern.*

Traffic demand pattern. To illustrate the observation 1, we consider a specific example as follows. Suppose there is a DCN architecture with a topology of 4-port Fat-tree [6] and 8 ToR switches. In the example of Fig. 1 [2], there is a 16-port OCS switch that connects all the ToR switches, which allows a ToR switch to connect to another arbitrary ToR switch. We fix the Fat-tree links and only reconfigure OCS links according to a specific traffic demand matrix. The optimization objective is defined as minimizing the time to transmit the demand matrix.

Since the Fat-tree is fixed, we use a matrix of size 8×8 to describe the topology configuration of the OCS switch. Then we randomly generate 10000 ToR-to-ToR demand matrix of size 8×8 and obtain the corresponding optimal topologies by brute-force searching. First, as Fig. 2 shows, we pick one topology sample P_0 (i.e., Fig. 2(b)) from the 10000 topology samples and select all the demand matrices whose optimal topology configuration is exactly P_0. Let **D** denote the set of the selected demand matrices. By summing up the demand matrices in **D**, we obtain the demand matrix shown in Fig. 2(a). Each grid in Fig. 2(a) corresponds to the traffic demand between a pair of ToR switches. The right-most color bar in Fig. 2(a) maps a range of colors (from blue to red) to a range of demand values. For example, a dark red color of the grid indicates that most demand matrices in **D** have a high traffic demand for the ToR pair. We can see that there exists a statistic pattern in the sum of **D**, i.e., certain grids in Fig. 2(a) have a color that is mapped to a much larger demand value

[2] The example follows the DCN architecture proposed in [51]. We assign each switch an index, which will be used to represent the corresponding matrix later. We will use this permutation throughout this paper.

than those of other grids. More specifically, the emerged pattern exactly matches the structure of the optimal topology P_0. Finally, we note that there is no topology configuration that can optimally support all the demand matrices. Instead, each topology configuration optimally supports a certain part of the entire set of demand matrices. This illustrates the observation 1.

Performance gap of heuristic algorithm. For comparison, we randomly pick one demand sample [3] from the set $\mathbf{D}$ in Fig. 2(a) as $d \in \mathbf{D}$ shown in Fig. 2(c). Then we run the well-known Edmonds weight-matching algorithm [17] utilized in existing topology-adaptation research work [13, 18, 51] to get the topology configuration P_1 shown in Fig. 2(d). We can see that the heuristic solution P_1 is quite different from the optimal topology P_0, which results in a 30% longer time to complete the traffic transmission.

The main disadvantage of the heuristic algorithm is that it tries to find the optimal topology configuration for each demand matrix without any knowledge of the global interactions between OCS configurations and the remaining fixed wired network part. As recent work [34] shows, for a large-scale problem, some special assumptions about the input traffic patterns (e.g., skew and sparse) will contribute to more efficient algorithms in solving this problem. However, such assumptions can not be generalized to other data centers which may have quite different traffic patterns due to their distinct up-layer data services [42, 47].

Motivation of deep learning. To address the above issue, the observation 1 motivates us to solve the problem using a deep learning solution as follows. First, in the offline phase, we can learn the specific traffic patterns from the history trace in the target data center. Next, we can learn the mapping relationship (i.e., the global network interactions) between the traffic patterns and the optimal topology configurations. Finally, during the online phase, we can quickly classify the demand matrix to a traffic pattern, and then use the learned mapping to obtain the corresponding optimal topology at a fast speed. In the above example, the network is very small and we can search for the optimal topology configuration by simply brute-forcing. However, when the network further scales up, finding the optimal one from an exponentially large set of candidate topology configurations is highly non-trivial [34]. Hence even in the offline phase with a reasonable time limit (e.g., one day or one week), we can only obtain the approximate optimal topology configurations in practice. Now we present the second observation to address this issue in deep learning:

Observation 2. *The high-performance topologies that support a certain traffic demand are sharing a set of critical links (i.e., a key topology structure).*

Key topology structure. To illustrate the observation 2, following the same setup in Fig. 2, we first randomly select a demand matrix sample $d' \in \mathbf{D}$ shown in Fig. 3(a). Then we generate the Fig. 3(b) by summing up all the topologies that have a performance gap within 5% that of the optimal solution (i.e., the *high-performance topologies*) for d'. Intuitively, the grids that have a darker color mean that they are selected by most of the high-performance topologies. Hence the darker grids in Fig. 3(b) correspond to the critical links in the high-performance topologies and form the key topology structure. For comparison, we also generate the Fig. 3(c) by summing up all the topologies that have a performance gap within 5% that of the worst-case solution (i.e., the *low-performance* topologies) for d'. Similarly, we can see that the low-performance topologies also have some critical links, which do not overlap with any critical links found in the high-performance topologies. For comparison, we show the heuristic solution of weight-matching algorithm in Fig. 3(d). We can see that without any prior knowledge about the key topology structure, the heuristic solution selects some links that fall into the critical links of the worst-case topologies. It further results in a 33% performance gap from the optimal solution.

[3] We also evaluate all the demand samples and find similar performance results as the example demonstrated above.

The observation 2 provides us with a key insight that even though we can not obtain the optimal topology configurations for a large-scale network in the offline phase, there exists important statistic information about the critical links in the suboptimal solutions which will help us effectively infer the high-performance topology configurations close to the optimal solution.

To summarize, the above two observations demonstrate the possibility of learning the critical traffic patterns and key topology structures with respect to the given performance objective from data. This provides us an effective strategy to capture the most essential factors for the topology optimization. As we will show in §6, xWeaver can learn the patterns in both the traffic and topology effectively and thus achieve the topology optimization with higher performance than existing solutions. In the following, we will present the detailed design of xWeaver following the above motivations.

3 XWEAVER FRAMEWORK

In this section, we will first introduce the design goals and constraints in the xWeaver framework, and then present the overview of its system modules.

3.1 Design goals and constraints

System input and objective. xWeaver has two system inputs. The first is an input of a *virtual topology* $G(V, E)$, where the node set V denotes all the network switches. The link set E includes the fixed wired links E_w and all the configurable links E_c. We denote an *instance* of the virtual topology as a subgraph $\widehat{G}(V, \widehat{E})$ that consists of the original node set V, the wired link set E_w and a *subset* of configurable links $\widehat{E_c} \subseteq E_c$ that satisfy all the link constraints (will be discussed below). The second input of xWeaver is a demand matrix D that describes the traffic demands between any pair of racks. In this paper, we do not focus on estimating the traffic demands and the demand matrix can be obtained following similar techniques in previous work [18, 23, 51]. The target of xWeaver system is to find the optimal (or near-optimal) instance of the virtual topology to support the input traffic demands.

Network connectivity. Although providing the highly-flexible topology is beneficial, existing topology reconfiguration technologies will introduce unavoidable delay due to the transmission interruption in the link reconfiguration phase. xWeaver exploits the fixed wired links to construct a *connected graph* (e.g., Fat-tree[6]) for all the ToR switches. In this way, the entire network remains connected for any topology configurations of OCS switches or wireless radios. When the topology is reconfigured, current on-going flows on the affected links will be temporarily redirected to the fixed-connected network part. The solution also increases the flexibility of topology configurations and provides the worst performance guarantee when all link reconfigurations fail.

Link conflict constraints. A *link-conflict* constraint $Conf(S, n)$ denotes that there are at most n links in the set S that can be built simultaneously. This general definition enables a unified and easy way to model the physical constraints of different topology reconfiguration technologies. For example, when considering the configurable links using wireless, two links (e.g., s_1 and s_2) that are physically close to each other can create the wireless interference, which prevents the concurrent transmissions on the two links. It can be described by $Conf(\{s_1, s_2\}, 1)$.

OCS switch and wireless radio. With the above framework, we now show that it is easy to support existing optical/wireless configuration technologies. For a full-duplex $2n$-port OCS switch, the configuration of its inside links form an $n \times n$ binary matrix P where each element $P_{i,j} = \{0, 1\}$ means whether port i is connected to port j. The physical constraint of OCS requires that each sender port connects to only one receiver port. Hence each row/column in the configuration matrix P can only have exactly one 1. This constraint can be easily represented using $2n$ *link-conflict* constraints $\{Conf(S_i, 1)\}$, where S_i denotes the set of potential links connected to the port i.

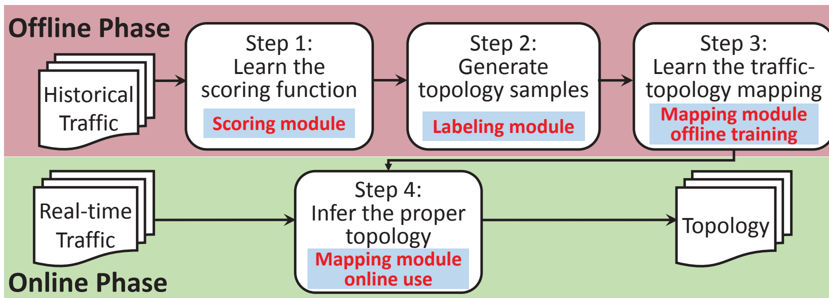


Fig. 4. xWeaver framework overview

For the reconfigurable wireless links, there are two types of link-conflict constraints. The first is the aforementioned wireless interference constraint. The second is the port conflict constraint, where only n links are allowed to be built for each n-port wireless radio. It can be directly described by the link-conflict constraint $Conf(S_j, n)$ where S_j is the set of potential links connected to the jth wireless radio. Since the link constraints in both technologies can be described by the framework, for simplicity, we will focus on the analysis for the OCS-based architecture in the following.

3.2 System modules

The overview of our xWeaver system is shown in Fig. 4. Generally, the mapping module of the xWeaver system provides the key function to infer the best topology from traffic demands, while the scoring module and labeling module are designed to support the efficient training of the mapping module. In the following, we will present the details of these basic modules respectively.

Scoring module. This module is used to provide a fast performance evaluation for any given pair of traffic demand matrix and topology (i.e., the *traffic-topology pair*). The module input is a traffic-topology pair and the output is a real-valued performance *score*. In xWeaver, we allow the network operators to define the score as any performance metric that they aim to optimize, e.g., the flow completion time, network throughput or the up-layer application performance metrics.

Labeling module. This module is designed to label historical traffic traces with corresponding topologies that have high performance scores. It provides the traffic-topology samples for later model training. Existing traffic traces generally do not have the information on the network topologies. The labeling is made possible and automatic with the facilitation of the *scoring module*, which is trained to extract the features of both traffic and topology. Specifically, the label for each traffic demand matrix can be obtained by searching the potential topology space with the *scoring module*.

Mapping module. This module is the core of xWeaver to learn the high-dimensional global mapping between traffic and topology. It consists of two processes: a) building a specialized neural network which maps a traffic demand matrix to a topology configuration; b) training the neural network parameters from the data samples provided by the *labeling module*. Then the trained neural network can be used to perform efficient topology inference online. We also provide a design option to enhance the output of neural network by combining a flexible probability graph model. It can help explicitly embed a variety of prior human knowledge about the target network architecture.

Workflow. Following the prior studies, the xWeaver system utilizes a centralized network controller to implement the flexible topology configuration [24, 33]. Specifically, in the offline phase, given historical traffic traces, xWeaver controller trains the mapping module with samples generated from the labeling module and scoring module. Next, in the online phase, xWeaver controller exploits the

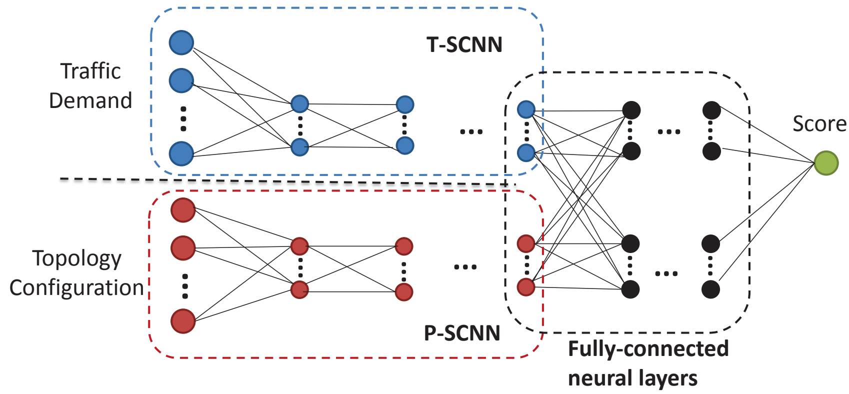


Fig. 5. SCNN structure for feature extraction and scoring function learning

mapping module to periodically update the configurations of OCS switches (or wireless radios) based on the traffic statistics reported from ToR switches [7].

4 TRAFFIC-DRIVEN TRAINING SAMPLE GENERATION

In this section, we will introduce the detailed design of the scoring module and labeling module in xWeaver respectively. By combining the two modules, the historical traffic traces can be *automatically labeled* with the corresponding high-score topologies without any human efforts. This automatic labeling process allows for generating enough labeled data for high performance training. These modules work only in the offline phase, so it can tolerate a relatively-long time to generate enough traffic-topology samples. After offline training from these samples, the mapping module can easily make online topology decisions in real time.

4.1 Topology performance scoring

Problem analysis. The objective of the scoring module is to quantitatively evaluate the performance of a global topology configuration. We denote a performance scoring function as $Score(f, p)$ to give a real-valued score on the transmission performance of the traffic demand f using the global topology configuration p. The score can be defined as any performance metric that the network operator desires to optimize (e.g. the completion time of the hadoop job) according to their own interests. A straightforward method to implement the performance scoring function is to use a conventional network simulator (e.g., the ns-2 [4]) to mimic the real system settings. Intuitively, by running a given traffic demand f in the network simulator with a topology configuration p, we can obtain the target performance metric directly.

However, the above method is not feasible for our deep learning solution. Existing network simulators are relatively slow as they require simulating the detailed network protocols and packet-level transmission events. For example, a simple run to evaluate the topology score for a demand of 100 10MB-flows in a 8-port fat-tree takes about 4 minutes in our ns-2 simulation. The successful training of a neural network at such scale requires more than 10000 score evaluations, which will result in a sample generation time as long as one month. Hence in practice, we turn to find an approximate scoring function running faster but within a tolerable accuracy loss.

Scoring module design. The basic motivation to introduce the speed-up scoring is that we can directly map a traffic-topology pair (f, p) to the score s through the offline training using a neural network (denoted as the *scoring-NN*). We use a number of *traffic-topology-score* (TTS) samples $\{(f, p, s)\}$ to train such a neural network. In our design, the TTS samples can be directly acquired from measurement traces generated from a real network. Due to the lack of measurement data, we

ALGORITHM 1: Topology Generation Algorithm

```
Input: Demand matrix X_t, Previous topology Y_{t-1}
Output: High-score topology Y_t for demand X_t
Y_cur ← Y_{t-1} ;                                   // Searching from last topology
for Search_Depth = 1 to MAX_DEPTH do
    // Local search with random rolls
    Y* ← Y_cur, Progress ← False
    for Search_Width = 1 to MAX_WIDTH do
        Randomly select a neighbor Y_o of Y*
        if score(Y_o) > score(Y_cur) then
            Y_cur ← Y_o, Progress ← True
            Update the beam array B with Y_cur
        end
    end
    Set the current highest-score topology to Y_best
    // Jumping out of the local optimal points
    if Progress is False then
        Set Y_cur as the highest-score topology in B with ε probability, otherwise set Y_cur randomly
          from B
    end
end
return Y_t ← Y_best
```

use the flow-level simulator to generate such data samples in our simulations. We take the traffic demand matrix f and the topology matrix (i.e. the adjacency matrix of the network topology) p as inputs, and take the performance score s as the label. After the training, using the scoring-NN to evaluate the score of a traffic-topology pair takes only 4ms under the same setting of the NS-2 simulation, which is several orders of magnitude faster than directly running the network simulator.

As a straightforward way of implementing the scoring-NN, the input traffic and topology can be mapped to the output score with multiple fully-connected hidden neural layers (called the *fully-connected NN*). However, with extensive simulations, we find that it is challenging to train a multi-layer fully-connected NN for our problem. The mapping between a traffic-topology pair and the arbitrarily-defined score is highly complex, which involves the implicit interactions introduced by the underlying transmission protocols and specific network structures. This will result in a large-scale fully-connected NN with a large number of parameters, which is challenging to train within a reasonable time and a limited number of data samples.

The above issue motivates us to design a specialized neural network that has less complexity and high training efficiency. The key insight is that a pattern that emerges in the input traffic will hardly emerge in the input topology. Hence as Fig. 5 shows, we propose to use two separate multi-layer *convolutional* neural networks [31] as the basic components to perform the feature extraction independently. This separate pre-process contributes to a high learning efficiency because each input only focuses on its own feature extraction. Moreover, the separate structure will generate a neural network with significantly smaller number of parameters (i.e., the number of neural connections) for efficient training. Then at the end of the scoring-NN, we combine their derived feature results and map them to the final score with a small number of fully-connected neural layers. We denote the above neural network as a *separate convolutional neural network* (SCNN). We will evaluate the learning performance of SCNN in §6 and demonstrate that it can achieve a higher scoring accuracy and a faster convergence speed than the conventional fully-connected NN.

4.2 High-score topology sample generation

Since the scoring module enables the fast evaluation of the topology score, in this section, we will introduce how the labeling module generates the labeled data by mapping the traffic traces to high-score topologies.

As the number of candidate topologies may be exponentially large even for a small-scale DCN [18], it is infeasible to find the optimal global topology configuration for each traffic demand. As introduced in §2, rather than finding the optimal topology samples, our labeling module is designed to generate the high-score topologies. This is because the key topology structure of the optimal solution can be learned from the statistics information in the high-score topologies (see the *observation 2* in §2). Intuitively, the higher the scores of the topologies we can obtain, the more accurate the key structure that our solution is able to learn.

In the labeling module, we design a heuristic search algorithm (Algorithm 1) to generate the high-score topology samples with a controllable time overhead. Let $N_\delta(p)$ denote the set of neighbor topologies of a given topology p, where each topology $\widehat{p} \in N_\delta(p)$ has at most δ different edges from the topology p. Here we also refer the parameter δ as the *search depth*. Consider a traffic trace $\{f_t\}$ collected from the target data center. For each traffic sample f_t at time t, we want to generate the topology sample p_t that satisfies: $p_t = \arg\max_{p \in N_\delta(p_{t-1})} Score(f_t, p)$. By limiting the search depth, we are able to search for a local-optimal solution within a reasonable period of time. However, with extensive simulations, the above direct search does not generate the satisfactory topology configurations. Taking a closer look at the problem, we find that this is due to the fact that a lot of different neighbouring topologies are sharing similar scores. Thus it is inefficient to stop the search when finding only one local-optimal solution. In xWeaver, we use both the beam search and the random start schemes to help jump out of the local-optimal points. Thus it keeps exploring a better local-optimal solution until a pre-defined maximum number of iterations are reached.

We note that a direct application of the conventional beam search is infeasible for our problem. This is because the conventional beam search requires computing the scores of all the neighbouring topologies in each iteration. Since both the solution space and the number of neighbouring topologies may be very large, such a computation overhead involved in each iteration would be very high. In xWeaver, in each iteration, we only randomly select a small number of neighbouring topologies via an edge-exchange operation [4] [13, 38] to evaluate their scores, which is denoted as the *search width*. By limiting both the search depth and width, the overall time overhead can be further controlled. The detailed analysis of the sensitivity of the network performance to the topologies identified by the labeling module is discussed in §8.

5 TRAFFIC-TOPOLOGY MAPPING LEARNING

The techniques in previous sections allow us to generate the complete data samples for later model training. In this section, we will present the design of a neural network (FPNN) to perform the traffic-topology mapping in xWeaver. Next, we introduce how to embed flexible prior human knowledge into FPNN for further performance improvement.

5.1 Neural network for mapping learning

Challenge in FPNN design. The objective of the FPNN is to learn the mapping function between the input traffic demands and output topology configurations. A straightforward solution is to build a fully-connected NN, where the input and output neural layers are the traffic demands and the

[4]First, we uniformly select two existing configurable links of $Y*$ $a \to b$ and $c \to d$. Then we connect them via replacing links $a \to b$ and $c \to d$ with links $a \to c$ and $b \to d$.

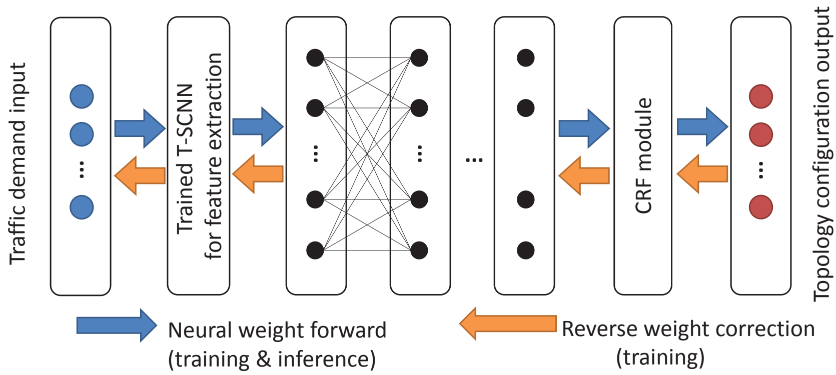


Fig. 6. Structure design of the FPNN neural network

topology configurations respectively, and the median layers can be constructed with multiple fully-connected neural layers. However, the traffic-topology mapping function involves some complex high-dimensional global interactions, such as the arbitrarily-defined optimization objective, the underlying routing protocols and the explicit link-conflict constraints associated with specific reconfiguration technologies (e.g., the OCS switch/wireless radio in §3.1). Since a fully-connected NN blindly maps the input to the output without any prior knowledge about the problem input and output, the high complexity of the mapping function would require training a large-scale fully-connected NN. It is challenging in practice due to the large number of connection parameters in a fully-connected NN [45].

Input feature extraction with T-SCNN. To address the above issue, rather than using the raw traffic input to construct the mapping, we propose to exploit the critical features in the traffic input for dimensionality reduction. Specifically, we carefully select a part of the well-trained SCNN (denoted as the T-SCNN structure in Fig. 5) introduced in §4.1 to extract the critical features from the input traffic. Then we copy both the structure and trained parameters of T-SCNN to construct the input of the neural layers of FPNN (Fig. 6). The benefits of this design are three-folds: (1) The training of T-SCNN is driven by the self-defined optimization objective, which encodes the essential traffic features required to optimize the target performance metric; (2) The T-SCNN is trained together with the feature extraction of the topology configuration, which encodes the essential traffic features required to interact with the topology configurations; (3) The T-SCNN has been well trained in the scoring module and thus does not need the time to perform re-training from the scratch.

After the feature extraction, we use a small number of fully-connected neural layers in FPNN to map the traffic features to the output topology configuration. As shown in the observation 1 (introduced in §2), the mapping function between the traffic patterns and the topology configuration is potentially straightforward and simple. As we will see later in §6, the number of fully-connected neural layers required by FPNN is much smaller than that of a conventional fully-connected NN, which ensures the high learning efficiency of FPNN.

5.2 Prior human knowledge embedding

In previous sections, we introduce a completely-automatic deep learning solution without any human modeling efforts. However, such a pure neural network solution can not exploit any specific human knowledge about the network architecture into the optimization process. In this section, we intend to answer the following natural question: “*Can we do better if we provide some explicit prior knowledge about the link relationship and link constraints in the target DCN?*" For the case where the network operators have some specific knowledge about the network architecture, we design a new module named CRF (Conditional Random Field) to allow the embedding of such knowledge to enhance the performance of FPNN.

CRF module. As Fig. 6 shows, we append a CRF module at the output layer of FPNN. The CRF model is an advanced probability graph model (PGM) that is used for modeling the link and node relationships in social networks [49]. As a natural extension, CRF can also be used to model the data center networks. In our problem, the CRF input is the original output of the FPNN, while the CRF output is a new topology that is *corrected* by the prior human knowledge.

Let G be an undirected model over the set of random variables $\mathbf{X}$ and $\mathbf{Y}$. The $\mathbf{X}$ is a set of *observation* random variables x_{ij}, which denotes whether the link between switch i and switch j exists in the output topology of FPNN. The $\mathbf{Y}$ is a set of *labeling* random variables y_{ij}, whether the link between switch i and switch j exists in the output topology of CRF. Let $\mathbf{C}$ denote a set of cliques in G. The CRF defines the conditional probability distribution of the labeled topology $\mathbf{y}$ (an instance of $\mathbf{Y}$) given the observed FPNN output $\mathbf{x}$ (an instance of $\mathbf{X}$):

$$P(\mathbf{y}|\mathbf{x}) = \frac{1}{\mathbb{Z}(\mathbf{x})} \prod_{c \in \mathbf{C}} \phi(\mathbf{x}, \mathbf{y}|_c),$$

where $\mathbf{y}|_c$ are the elements in $\mathbf{y}$ associated with clique c, and $\mathbb{Z}(X) = \sum_{\mathbf{y}} \prod_{c \in \mathbf{C}} \phi(\mathbf{x}, \mathbf{y}|_c)$ is the normalization factor. $\phi(\cdot)$ is a *potential function* taking a log-linear form [48], i.e., $\phi(\mathbf{x}, \mathbf{y}|_c) = \prod_{c \in \mathbf{C}} \exp(\sum_k \lambda_{k,c} f_{k,c}(\mathbf{x}, \mathbf{y}|_c))$ where $f_{k,c}$ are the feature functions and $\lambda_{k,c}$ are their corresponding weights.

To use a CRF module in FPNN, we first need to define a set of feature functions that describe our prior knowledge about the link relationship in the topology configuration $\mathbf{y}$. Next, given a set of labeled training data $\mathbf{D} = \{\mathbf{x}^i, \mathbf{y}^i\}$, we use the maximum likelihood (MLE) method to train the CRF parameters $\Lambda = \{\lambda_{k,c}\}$, i.e., one weight parameter for each feature. To generate the training data $\mathbf{D}$, we use each input traffic demand f to compute the FPNN output as the $\mathbf{x}$ and take the topology sample (obtained by the sample generation module in §4.2) corresponding to f as the $\mathbf{y}$. After training the weight parameters, the CRF module uses MLE to do the online inference of the new *corrected* topology for each FPNN topology output. The objective of MLE is to maximize the likelihood of the inferred topology configuration that satisfies all the feature functions, i.e., finding the topology configuration y to maximize the conditional probability $P(\mathbf{y}|\mathbf{x})$ given the observed FPNN output $\mathbf{x}$.

In the following, we will present some examples on how to define the feature functions that capture different types of prior human knowledge.

Embedding the link-dependency relationship. Considering the general routing property [5] that each flow requires a routing *path* between the source node and destination node in the topology. Based on the path definition, we have the following observation:

Observation 3. *If a link exists in the topology configuration, then the probability that its neighbouring links exist in the topology configuration becomes high.*

For the above observation, each pair of the *neighbouring* configurable links (e.g., $y_{i_1 j_1}$ and $y_{i_2 j_2}$) defines a clique c. Then the feature function $f_{k,c}$ is defined as a binary indicator function:

$$f_{k,c}(\mathbf{x}, \mathbf{y}|_c) = \begin{cases} 1 & \text{if } y_{i_1 j_1} = 1 \text{ and } y_{i_2 j_2} = 1 \\ 0 & \text{otherwise.} \end{cases} \quad (1)$$

[5]For simplicity, we only gives a general property independent of specific routing protocols. More complex link dependency in specific routing protocols can also be modeled in a similar way.

The above feature function embeds the basic prior knowledge about routing path and also the specific information on link neighbouring in the target DCN.

Embedding the link-conflict constraints. In the following, we present how to define the feature function to embed the explicit *link-conflict constraints.* A link-conflict constraint $Conf(S, n)$ means at most n links are allowed to be built simultaneously in the link set S. It can be used to describe the link constraints for both the OCS switches and wireless radios (see details in §3). Based on the definition, it can be translated as:

Observation 4. *If a link exists in the topology configuration, then the probability that its conflict links exist in the topology configuration becomes zero.*

For the above observation, each link-conflict constraint $Conf(S, n)$ defines a clique c' which contains all the links in S. Then the feature function $f_{k,c}$ is defined as

$$f_{k,c}(\mathbf{x}, \mathbf{y}|_{c'}) = \begin{cases} 1 & \text{if } \sum_{y_{ij} \in c'} y_{ij} <= n \\ -\infty & \text{otherwise.} \end{cases} \tag{2}$$

Note that when the feature function takes value $-\infty$, its corresponding factor in the potential function ϕ will be zero, which forces the probability of the link-conflict cases in the inferred topology of CRF module to be zero.

6 SIMULATION

Simulation setup. Our simulations are conducted through a customized flow-level simulator. Since it is infeasible to obtain the optimal solutions when the network scale becomes large, to compare the performance of different solutions with the optimal topology configuration, we first focus on a DCN topology of a 4-port fat-tree [6] with 10Gbps wired links. For the reconfigurable part, we follow the same OCS-based architecture setting in [13, 51] and use a single 16-port 10Gbps OCS switch to connect all the ToR switches so that each ToR switch can connect to one of the other ToR switches directly through the OCS switch with a 10Gbps optical link. Then we scale up the simulation setup to a 16-port fat-tree to verify the performance gains under a larger scale topology. Finally, to evaluate the scalability of our framework, we compare the running time of different solutions under various topology scales (§8).

In xWeaver, we implement the two neural networks (SCNN in §4 and FPNN in §5) with the deep-learning library `caffe` [26]. The CRF module designed in FPNN is implemented using the probability graph model library `grante` [1]. In simulations, our optimization objective is defined as minimizing the **completion time of the demands** (CTD) (i.e., the time it takes to complete all the flow transmissions in the demand matrix [23]). Hence the *score* of a topology is defined as the CTD under the topology configuration. We will discuss other objectives and score definitions in §8.

We compare the topology performance of our xWeaver solution with three other solutions. The first is the *Weight-matching* solution, which generates the topology configuration with the well-known Edmonds algorithm [17] used in existing topology-adaptation research work [13, 18, 51]. The second is the *Sample* solution, where the topology configurations are generated by a heuristic search in our labeling module (§4.2). The third is the *Optimal* solution, which has the lowest CTD for all the demands [6]. It is implemented as a brute-force search over all the candidate topology configurations. Our xWeaver solution uses the FPNN to infer the topology configurations and the CRF module is enabled by default. For the traffic input, we first use a random traffic pattern with 20000 demand matrices to provide a comprehensive performance evaluation for each xWeaver module under various potential traffic cases. Next, we analyze the performance under typical

[6]We can obtain the optimal solution in most of the test scenarios, which implies that our framework could achieve a good performance in general.

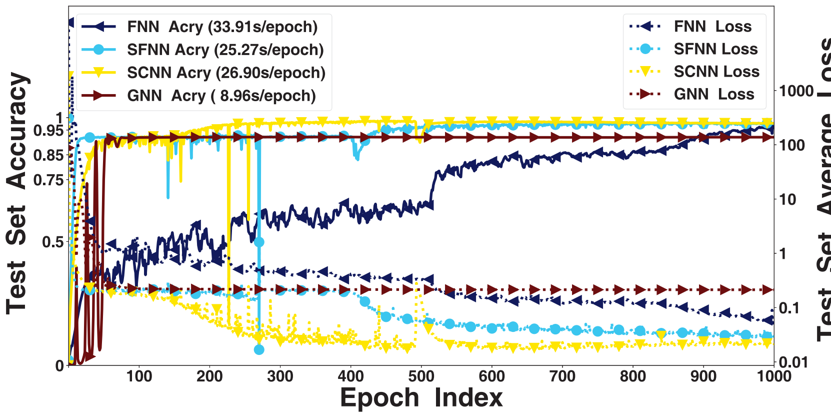
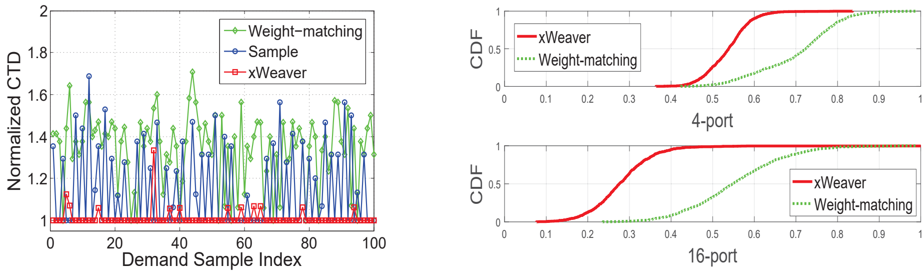


Fig. 7. Test loss and accuracy of different solutions.

traffic patterns in production DCNs, including the application traffic pattern [9], hot-spot traffic pattern [24] and hybrid traffic pattern [28].

Performance of scoring module. To analyze the performance of the scoring module, we evaluate four neural network solutions. In SCNN of xWeaver (§4.1), we have the first part implemented with two separate convolutional neural structures (i.e., the T-SCNN and P-SCNN in Fig. 5), each consisting of two convolutional neural layers. We then have the second part of SCNN constructed with three fully-connected neural layers. In the second solution, we directly replace the convolution neural layers in SCNN with the same number of fully-connected neural layers (SFNN). Then we implement the baseline solution as a fully-connected NN (FNN). Finally, we implement the graph neural networks (GNN) following the similar setting in [30] as another baseline. We generate 250000 random traffic matrices $\{f_i\}$ and topologies $\{p_i\}$. For each pair (f_i, p_i), we take the the mean flow completion time of the demand as the performance score and obtain the score s_i by running the flow-level simulator. Thus we obtain 250000 fully-observed *traffic-topology-score* samples $\{(f_i, p_i, s_i)\}$. The *training set* is set as the first 90% TTS samples while the *test set* is set as the last 10% TTS samples. For each input pair (f_i, p_i), we denote s_i' as the output score of a neural network. Intuitively, the smaller the difference between the neural network output s_i' and the score s_i in the TTS sample, the higher the *scoring accuracy* achievable by the neural network. Denoting the *relative error* as $\frac{|s_i'-s_i|}{s_i}$, we compute the scoring accuracy for each solution as follows. First, we get the output score s_i' and compute the *relative error* of all the test samples. We then find the number of samples whose relative errors are within 5%, and calculate the *scoring accuracy* by dividing this number by the total number of test samples. We train all neural networks to compare for enough epochs over the same training set.

The accuracy and loss associated with scoring are shown in Fig. 7. We see the results of all neural networks converge over time. GNN converges first at around epoch 100 with the scoring accuracy of 92.1%. SCNN converges soon and achieves the highest scoring accuracy of 98.7% around the epoch 430. By adding in more training time to all networks, FNN and SFNN finally converge around the epoches 1000 and 910 with the scoring accuracy of 96.7% and 97.8%, respectively. This indicates that SCNN can achieve higher scoring accuracy than the other three solutions if enough training epochs are given. When the training time is limited, SFNN and GNN can quickly achieve a good scoring accuracy, but SCNN can achieve higher accuracy just with a little more time. It confirms that the excellent feature extraction ability of convolutional neural layers shown in other fields [29, 31] also applies to our problem.



(a) The CTD normalized by the *Optimal* solution under 4-port fat-tree

(b) The CDF of the CTD normalized by maximum value under 4-port and 16-port fat-tree

Fig. 8. Performance comparison of different topology configuration solutions under different network scales.

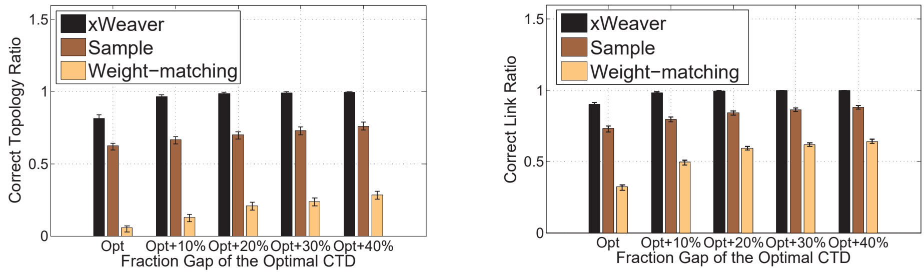
In this experiment, we set the sizes of all neural networks to be similar thus their training durations are comparable. Depending on the neural network complexity, each network costs different amount of time to go through all training samples in an epoch. The training time required by each network is listed besides its legend in Fig. 7. To achieve similar scoring accuracy, the total training time of FNN can be more than 9 hours, which is three times that of SCNN. To balance the efficiency and accuracy, the architecture of neural networks in scoring module should be designed to meet different requirements and constraints, such as the computation resources, the performance metric, and the scale of the DCN.

Performance of topology configuration. In Fig. 8, the CTD of four solutions are evaluated under different network scales. For the 4-port fat-tree, the CTD results in Fig. 8(a) are all normalized by the *Optimal* solution. We can see that although the performance of *Sample* varies quite a lot for different demands, our *xWeaver* solution trained from *Sample* outperforms *Sample* in the CTD metric, and has its performance very close to that of the *Optimal* solution. *Sample* has a large fraction of high-quality topologies together with a small fraction of low-quality topologies. *xWeaver* shows the ability of learning the important demand patterns and key topology structures from the high-quality topology samples, and then using them to infer the high-quality topologies for new traffic demands. On the other hand, *Weight-matching* has the worst performance among all the solutions and experiences a large variation of topology qualities. Its completion time is about 50% higher than that of *xWeaver* [7]. We also evaluate the performance of xWeaver on achieving other four objectives, i.e. the mean, median, 90th percentile and 99th percentile flow completion time. *xWeaver* also performs very close to the *Optimal* solution. Similar to the above case, the completion time of *Weight-matching* Solution is about 19%, 21%, 20% and 29% higher than that of *xWeaver* with respect to the four objective values in order, which shows that xWeaver can be applied to the optimization of different objectives.

In Fig. 8(b), the CDFs of the CTD under 4-port and 16-port fat-tree are presented, respectively. Since the size of the candidate topology set expands exponentially with the increase of network scales, it is non-trivial to obtain the *Optimal* solution under the 16-port fat-tree. We can see that the performance gap between *xWeaver* and *Weight-matching* becomes more than 40% larger when the network scales up. This is because the interactions between the fixed part and the configurable part of the topology will become even more complex for a larger-scale data center network.

Performance of key topology learning. In Fig. 9, we present the detailed learning performance of *xWeaver* on the topology structures. As Fig. 9(a) shows, we evaluate the fraction of high-performance topologies found by different solutions. The case of $OPT + X\%$ corresponds to the

[7] We also evaluate the topology configuration performance of our solution for the wireless-based architecture using FSO. Since FSO has a comparable link bandwidth (10Gbps) as OCS with little interference footprints [19, 24], we found it generated similar performance results as the above OCS case.



(a) Learning performance of the optimal topologies

(b) Learning performance of the critical topology links

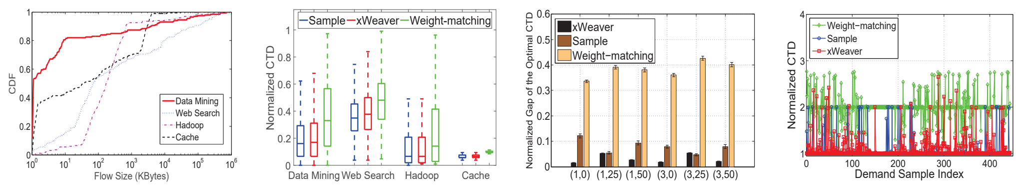
Fig. 9. Learning performance for topology structures.

topology configurations with CTD less or equal to $(1 + X\%)$ times of the optimal. The fraction of high-performance topologies found by all the solutions increases as expected. We can see that the fraction of optimal topologies found by *xWeaver* is above 80%, while that of *Weight-matching* takes less than 10%. This is because *xWeaver* can classify most of the n ew traffic demands (i.e., 80% in current settings) to correct traffic patterns which are then mapped to the optimal solution. However, *Weight-matching* lacks the prior knowledge about the specific traffic patterns, which results in a small number of high-performance topologies.

Similarly, in Fig. 9(b), the link fraction at case of $OPT + X\%$ is defined as the fraction of links built by different solutions which are exactly the same as those in the high-performance topologies with a CTD within $(1 + X\%)$ times that of the optimal. Thus the link fraction indicates the ratio of the links found to be high-performance critical links. We can see that *xWeaver* wins the highest link fraction at all the cases, which is on average 40% higher than that of the *Weight-matching*, and 20% higher than that of the *Sample*. This is because *xWeaver* has the ability of learning the critical links from the high-quality data samples generated by *Sample* and then using them to form the key topology structure close to the optimal.

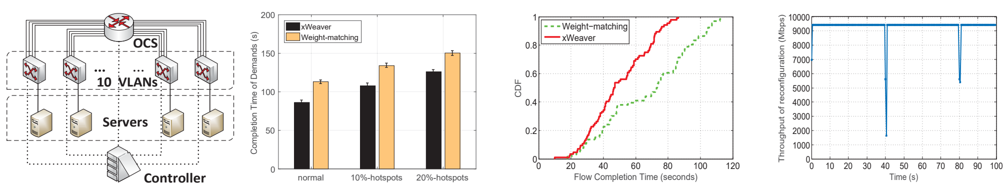
Learning performance for different traffic patterns. *(1) application traffic patterns.* We first evaluate the learning performance over different application traffic patterns in DCNs. Following the previous work [9], we use four application traffic patterns from production DCNs (Fig. 10(a)), i.e., the workloads of data mining [22], web searching [8], Hadoop [42] and caching [42]. The CTD values shown in Fig. 10(b) are normalized by the maximum CTD value of this set of experiments. We can see that the CTD performance of *xWeaver* and *Sample* is better than that of *Weight-matching* in all the traffic cases. An interesting finding is that the performance gap of CTD between *xWeaver* and *Sample* is relatively larger in the traffic case of *web searching* than that in all the other cases. This is because the flow distribution of *web searching* is relatively uniform while other traffic cases have certain skew traffic patterns. Thus the trace of *web searching* contains richer traffic patterns than that of the others, which makes it more difficult to learn when provided with the same number of data samples. Finally, we can find that the *Weight-matching* has the smallest performance variance in the traffic case of *cache*, whose flow size is two orders of magnitude smaller than the others. It indicates that the *Weight-matching* works more stably for the light-load traffic cases. Rather than limiting the performance gains to local links in *Weight-matching*, *xWeaver* learns the global network interactions from data traces and thus gains much higher flow performance for all the network loads.

(2) hot-spot traffic patterns. In Fig. 10(c), we evaluate the transmission performance of different topology configuration solutions over different hot-spot traffic patterns. Following previous work [16, 24], we use a pair of (X, Y) to denote the case of a hotspot traffic pattern where the



(a) CDF of flow size over application traffic patterns (b) Normalized CTD over application traffic patterns (c) Optimality gaps for hot-spot traffic patterns (d) CDF of normalized CTD over real data center traces

Fig. 10. Learning performance for different traffic patterns.



(a) OCS Testbed (b) CTD of different traffic patterns (c) CDF of flow completion time (d) Throughput of reconfiguration

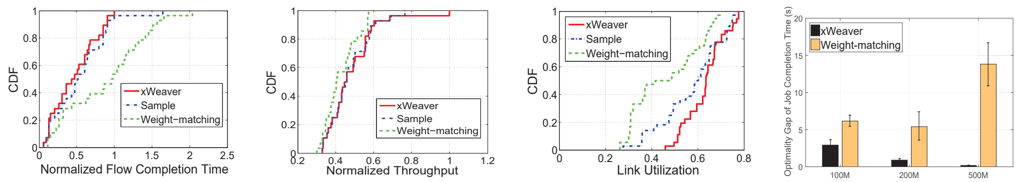
Fig. 11. Performance evaluation in an OCS-based testbed.

average demand size is increased by X times with additional Y percentage of hot-spot flows. For each case of the traffic pattern, we generate 20000 data samples for training the FPNN in xWeaver, and another 1000 data samples as the test set for performance comparison. As Fig. 10(c) shows, *xWeaver* achieves the optimality gap below 5% for all the cases while *Weight-matching* generates an optimality gap above 30% for all the cases. It demonstrates the effectiveness of *xWeaver* in successfully learning different types of hot-spot traffic patterns in DCNs. Moreover, for different traffic patterns, the performance gains of *xWeaver* over the *Sample* vary a lot. Generally, *xWeaver* achieves an optimality gap less or comparable to that of the *Sample*. Despite the difficulty in learning different traffic patterns with specific data samples, *xWeaver* can always exploit the critical traffic patterns from the data samples generated by *Sample* to achieve more efficient topology inference than the simple heuristic search in *Sample*.

(3) hybrid traffic trace. We derive the demand matrices from hybrid traffic traces of a university data center in [28] and analyze the transmission performance of *xWeaver* on handling the real traffic traces. In the traces, we find that the number of large flows (i.e., the flows with size larger than 1MB [8]) takes about 1.3% that of all the flows, while the total traffic size of large flows takes about 84.7%. It indicates that the demand matrices derived in the traffic trace show specific skew patterns. As Fig. 10(d) shows, we evaluate the CTD of different solutions for a sequence of demand matrices derived from a 15-minute traffic trace. All the CTD results are normalized by the optimal solution. We can see that the number of demand matrices that are completed within 1.25 times that of the optimal takes above 80% in *xWeaver*, while that number takes less than 10% in *Weight-matching*. Moreover, *xWeaver* slightly outperforms *Sample* and achieves a CTD within 1% that of the optimal. It highlights the ability of *xWeaver* in learning the critical traffic patterns from real traffic traces.

7 IMPLEMENTATION

Testbed setup. In this section, as Fig. 11(a) shows, we implement an OCS-based testbed to validate the performance gains of our xWeaver solution. The testbed is built with two 48-port 10Gbps electrical switches and five 16-port 10Gbps OCS switches. To achieve the maximum topology flexibility with the limited number of OCS switches, in our testbed all the five OCS switches are



(a) CDF of normalized FCT for small flows

(b) CDF of normalized throughput for large flows

(c) CDF of link utilization

(d) Optimality gaps of Hadoop job completion time

Fig. 12. Performance for supporting the specialized optimization objectives.

connected to construct a full-mesh topology as a single 80-port OCS switch. Furthermore, we virtualize 10 ToR switches on top of the two electrical switches by enabling their `Open-vSwitch` mode and isolating them into 10 VLANs. For each VLAN, there are four ports of the electrical switches connected to the OCS switch. Each virtualized ToR switch has one port connected to a physical server. Each server is equipped with 4-core Intel Xeon E5-1410 2.8GHz CPU, 24GB memory and one 10GbE NIC for the data transmission. We use another stand-alone server to serve as the *network controller*. All of the above electrical switches and OCS switches are connected to the controller through another electrical switch. The traffic are randomly generated for each pair of servers using `iperf` [3]. To match the 10Gbps link bandwidth, the traffic size is set within 15GB for each pair of ToR switches. We add different fractions of hot-spot flows into the demands to simulate different traffic patterns. The ECMP is implemented as the default routing scheme.

Experiment results. As Fig. 11(b) shows, for different traffic patterns with different fractions of hot-spot flows, the CTD of *xWeaver* is on average 17% lower than that of *Weight-matching*. The detailed flow performance is presented in Fig. 11(c). We can see that the number of flows that have a flow completion time (FCT) smaller than 70s takes about 80% in *xWeaver* while it takes less than 50% in *Weight-matching*. Different from the previous simulations that use a fat-tree as the fixed-connected part, the testbed uses a network architecture where the fixed connected part is a ring constructed by the optical links [13]. It validates the effectiveness of *xWeaver* in adapting the learning ability for different DCN architectures to achieve high-performance topology configurations in a real testbed.

In Fig. 11(d), we evaluate the throughput changes when reconfiguring the topology using the OCS switch. The throughput is recorded through `iperf` by switching the paths between two communicating servers. We find that the throughput drops during the topology reconfiguration and it takes about 300 ms for the throughput to restore. We note that this delay consists of both the actual optical circuit switching delay and also the delay due to some hardware-based internal signal processing, which matches the measurement result in previous work [18]. We will discuss later that the running time of *xWeaver* can be potentially limited within 100ms for a median-size data center, which matches such reconfiguration delays for easy online topology configuration.

8 DISCUSSION

Optimization for different specialized objectives. We have studied the performance optimization for minimizing the CTD. However, the network operators may prefer flexible and sophisticated optimization objectives in DCNs. The support of optimization over self-defined objectives may potentially enable new applications of topology reconfiguration in DCNs. In the following, we evaluate the performance of xWeaver for optimization on another two comprehensive objectives.

The first comprehensive objective is a fine-grained flow optimization objective, i.e., minimizing the FCT for small flows while maximizing the throughput for large flows [27]. This can be achieved in *xWeaver* by simply replacing the CTD *score* with a new score, which is defined as a weighted sum between the FCT of small flows and the throughput of large flows. Without loss of generality, we set an equal weight for the two metrics.

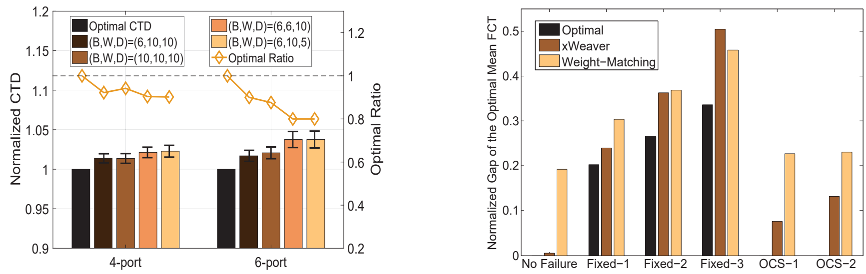
The detailed performance are evaluated in Fig. 12(a) and Fig. 12(b). The maximum FCT and throughput are all normalized by that of the optimal solution. We can see that *xWeaver* performs the best on the performance in terms of both the FCT and throughput. This can be further illustrated by the detailed link utilization in Fig. 12(c). We can see that the number of links that have a link utilization larger than 60% takes about 70% in *xWeaver* while the number takes less than 30% in *Weight-matching*. It indicates that *xWeaver* can fully utilize the reconfigurable link resources to optimize the transmissions for both the small flows and large flows.

The second one is an application-level performance metric on the *job completion time* of the benchmark Hadoop Terasort application. The job completion time is defined as the total time consumed to complete one Terasort job. We build a Hadoop (version 2.7.3) testbed on a partial Fat-tree topology with 10 Open-vSwitch-based virtual switches. Each half of the tree topology is built on a physical server with a dual-core 3.30GHz CPU and 16GB DDR3 memory. The links among edge switches are all reconfigurable, where each edge switch can connect to one of the others through a reconfigurable link. All the virtual switches are connected to a `Floodlight` controller [2], which provides the RestAPI interface to deploy the flow tables. We repeat running testbed measurements of generated traffic traces and job completion time for three different job sizes (i.e., the data file size for sorting): 100, 200 and 500 MBytes. Then we train the *xWeaver* and compare its performance against the *Weight-matching* on the testbed for new Terasort jobs. We also measure the optimal topology solution using the brute-forcing method as a benchmark.

Fig. 12(d) shows the gap against the optimal topology solution for both the *xWeaver* and *Weight-matching*. We can see that the optimality gap of *xWeaver* is at most half that of *Weight-matching*. Moreover, the optimality gap difference between *xWeaver* and *Weight-matching* increases when the job size becomes larger. This is because the job completion time mainly consists of the computation time and transmission time. When the job size becomes larger, the transmission time increases much faster than the computation time, which leads to a larger optimization space for the traffic-learning algorithm. It demonstrates the flexibility of *xWeaver* on optimizing application-level performance metrics by learning from real application traffic traces.

Learning adaptation for new traffic patterns. In Table 1, we compare the number of iterations used to train the FPNN under two different cases: 1) *Independent training*, where we re-train the FPNN for each traffic pattern; 2) *Adaptive training*, where we only initialize the FPNN by completely training it for the first pattern and then keep updating the parameters of FPNN after receiving the new data from the latter traffic patterns. As Table 1 shows, we find that the number of iterations required to update the FPNN for new traffic patterns takes only one tenth that of re-training independently. Moreover, the CTD performance of *adaptive training* is kept within a 10% gap from that of *independent training*. It demonstrates xWeaver's ability of adaptation for automatically updating its model parameters and quickly learning new traffic patterns from real-time traffic data. In practice, the flow size distributions and the locality of traffic are highly correlated with applications (e.g., web search, data mining, key-value stores) and their deployment formats[37]. From this perspective, the frequency to retrain FPNN can be adjusted according to the update cycle of applications, such as weeks. On the other hand, the FPNN can be updated once the network operator finds that the traffic pattern changes or a new application is launched since xWeaver can quickly learn new traffic patterns.

Sensitivity of CTD to the topology labeling. In Fig. 13a, the CTD of the samples generated by the labeling module are evaluated under different network scales following the setup in §6. We vary



(a) Sensitivity of labeling module with different network scales.

(b) Robustness with different numbers and types of link failures.

Fig. 13. Sensitivity and robustness analysis.

the parameters of the labeling module to check the sensitivity of CTD to the parameter selection and the topologies labeled. As shown in Algorithm 1, the parameters include the width of *Beam* array, the *search width* and the *search depth*, denoted as (B, W, D). All the CTD results are normalized by the optimal solution. The fractions of optimal topologies found by the labeling module are also shown as the *optimal ratio* in Fig. 13a. We can see that the optimal ratio ranges from about 80% to 94%, while the CTD is kept within 5% that of the *Optimal* solution. The performance of CTD is more stable than that of the optimal ratio under different parameter settings. It indicates that stable and near-optimal solutions can be obtained even if the topologies identified by our labeling module are sub-optimal. As expected, the performance further improves when the *search width* and *search depth* increase under different network scales.

Robustness to link failures. In Fig. 13b, we evaluate the robustness of the mapping module to different numbers and types of link failures following the similar setup in §6. We evaluate the performance of our learned configurations with the number of link failures selected from 1 to 3 in the fixed-connected part and 1 to 2 in the configurable-part of the topology. The performance metric is the mean flow completion time and the results are normalized by the value obtained by *Optimal* solution in the scenario of no link failure. When link failures happen in the fixed-connected part, in each set of the experiments, we randomly select a pre-determined number of links to set as failed ones for each demand sample. We also recompute the *Optimal* solution under each failure scenario. When link failures happen in the configurable part, we traverse all failure possibilities for each scenario and show the average performance degradation.

We can see that *xWeaver* can outperform *Weight-matching* in both fixed-link failure and configurable-link failure scenarios when the number of link failures is no more than 2 (more than 6% of all links in our setting), which shows the basic fault tolerance ability of our framework. When the number of link failures goes up, the performance of *xWeaver* declines more. This is because the *xWeaver* solution tries to balance the use of configurable and fixed part of links, while the *Weight-matching* solution only tries to maximize the flow volume transmitting over configurable links and is less impacted when a link failure happens in the fixed-connected part of topology. However, it is not a common case where up to 10% of all links fail at the same time in a large-scale DCN. According to a failure measure study [20], over half of failure events are isolated and only 10% of them contain more than four failures. Moreover, the network redundancy (e.g., back up links) can further reduce the impact of link failures [20].

xWeaver can also benefit from a fault-tolerant method. Since our mapping module does not take the fixed-connected part of topology as input, the mapping module (i.e. FPNN) is not aware of the link failures. However, from the results of our training process, there may be several near-optimal candidate topologies with negligible difference in their performance scores but with different levels

Table 1. Comparison of iterations required for different training processes.

Traffic pattern	(3, 50)	(3, 25)	(3, 0)	(1, 50)	(1, 25)	(1, 0)
Independent training (iter.)	23369	35783	20739	21502	20032	19341
Adaptive training (iter.)	23369	2431	2678	1440	2343	2536

Table 2. Comparison of online running time over different network scales.

Number of racks	10	100	200	500
Running time of FPNN (CPU) (ms)	0.02	0.29	1.56	10.4
Running time of FPNN (GPU) (ms)	0.09	0.18	0.48	2.37
Running time of Weight-matching (ms)	0.02	0.15	0.94	24

of robustness to link failures. Therefore, we could choose the topologies that have high scores and are also robust to failures. However, the evalutation of topology robustness is very computationally intensive, which needs a special design to speed up the process. We leave this to our future work.

Scalability of xWeaver in DCN. Training and running a large-scale neural network is a popular topic in recent deep learning research [15]. In Table 2, we present the online running time of different solutions to obtain the topology configurations over different network scales. The running time is measured on a computer with i7-core 4GHz CPU and 16G DDR4 memory. Since recent deep learning well supports the speed-up using GPU, we also compare the running time of FPNN when enabling a GPU of NVIDIA Geforce GTX 970 for online neural network computation. We can see that FPNN with only CPU enabled has a running time comparable to that of *Weight-matching*. When enabling the GPU, FPNN achieves a running time about 3-4 times faster for the network larger than 100 racks. For a smaller-scale network, the case of GPU is slower than that of CPU, as the copy cost from DRAM to GPU's graphic memory is larger than the computation cost when the scale is small. When the rack number increases, the scale of FPNN also increases with a large number of parameters, e.g., about 100 million parameters for the case of 500 racks. We find that our 16G memory can not support the network scale with more than 500 racks.

As for the offline training time of FPNN, we monitor the training processes under different traffic patterns and optimization objectives described in §6 and §8. Intuitively, the diverse traffic patterns will involve different distributions and the difference in objectives reflects the different demand-topology interactions, which will result in different training difficulty and time. However, we find that, in each of the above mentioned scenarios, the training process will take approximately 4-5 hours with the topology of 4-port fat-tree. With the 16-port fat-tree, the training process will take more than 1 day.

The scalability of memory and training speed can be effectively addressed by taking advantage of the rich distributed computation resources in data centers. Specifically, a large-scale neural network model can be partitioned across a cluster of commodity servers to perform parallel training, where their parameters are shared and updated through a parameter server [15]. This allows for highly efficient training of the neural network with a large scale (2 billion parameters on 120 servers [15]) and a fast speed (10 million samples in 1 hour [21]). When training the CRF module, we can trade off between the efficiency and accuracy, since we do not need to obtain the exact optimal parameters. First, we could benefit from the well developed solver library [1, 5, 39] to conduct the training and inference of CRF. If the scale of the problem is too large to solve accurately, approximation algorithms (e.g. relaxation and rounding) can also be used as alternatives for more efficient training.

Flow-level simulator. In xWeaver, we use a flow-level simulator to generate data samples and evaluate the performance of different reconfiguration schemes, which is a trade-off between fidelity and efficiency. For each traffic demand and topology, the simulator first partitions each flow into several subflows with their routing paths found by ECMP scheme. In the simulator, the time is discretized with RTT. In each time step, the simulator computes the transmission rate for each flow with the conventional TCP AIMD strategy. Then the simulator decreases the volume of each subflow with its current rate. The flow-level simulation will inevitably lose accuracy since it can not accurately simulate the packet level mechanism (e.g., ECN). The direct impact is that we cannot easily deploy DCTCP [8], the de facto protocol in today's DCN, in our simulator. However, using this flow-level simulator, we can generate data samples much faster than the packet-level simulator. For example, it takes only 1 hour to generate 20000 samples with the topology of 4-port fat-tree.

The choice of CNN. In xWeaver, we use CNN to extract features from traffic demands and topology configurations. The proper permutation scheme of CNN (shown in Fig. 1) and the often regular topology of data center networks allow us to efficiently map the local network topologies to the matrix representations. For example, the ToR switches on the same pod are indexed next to each other, thus the local topology structures and traffic can be easily captured by CNN. From another perspective, the underlying structure of topologies is a graph, which has a special property permutation invariance. Recently the graph neural network [43] has been proposed to capture the permutation invariance and achieve good performance in tasks involved irregular graphs, such as link prediction [54] or node classification [30] in social networks. We compared CNN to graph neural network in §6. The result shows that GNN has a great potential to deal with the graph-based problems. However, existing approaches based on graph neural network cannot efficiently deal with the dynamic network topology we consider and is more complicated to run for the regular data center topology. It requires a special design to extend the graph neural network for efficiently solving the problem of this paper. We leave this to our future work.

9 RELATED WORK

In recent years, a number of different topology-reconfigurable DCN architectures have been proposed with flexible networking technologies (e.g., the OCS switches and wireless radios) to address the dynamic traffic demands [13, 14, 16, 18, 19, 23, 24, 33, 35, 36, 41, 51, 53, 55, 56]. For example, c-Through [51] uses an architecture where flexible OCS links can be built among all the ToR switches, while Helios [18] proposes to build the OCS links among different pod switches. However, their topology flexibility is restricted by a limited number of single-hop optical links. OSA [13] proposes an architecture that supports higher topology flexibility via multi-hop optical links. Besides, xFabric [32] further pushes the topology reconfiguration ability into the racks. Recently, flat-tree [53] achieves the global topology flexibility with "converter swtiches" and could change the network to one of the three predefined architectures when necessary. With the extra functionality of ToR buffer, RotorNet [36] proposes a scalable circuit-based architecture that achieves flexibility by cycling through a series of optical matchings. They provide similar benefits on high topology flexibility, high transmission bandwidth and low cost in wiring and power consumption.

With the increasingly high topology flexibility, existing work generally adopt the fast heuristic solutions, which may result in performance potentially far away from the optimal. Recent work [12, 34] fully explores different algorithms to trade off the speed of configuration computation and the quality of OCS configuration. However, the proposed algorithms are designed under a simplified architecture with one OCS switch and one electrical switch, which is difficult to be generalized for conventional DCNs.

Previous work (Condor [44]) is the first to use the constraint-based synthesis to generate candidate topologies for efficient offline network planning. Since the topology searching may take tens of seconds, it is not suitable for our problem of online topology configuration. In this work, xWeaver intends to support the global topology optimization for existing hybrid architectures in a unified way. Rather than relying on specific assumptions and human-designed models, in xWeaver, we address the modeling difficulty and the running-time issue by learning specific traffic patterns offline from historical traces in the target data center. Benefited from the offline training, xWeaver achieves both the high-quality topology solutions and satisfactory online running speed.

10 CONCLUSION

In this paper, we present xWeaver, a traffic-driven deep learning solution that can enable the high-performance global topology configurations in DCNs with little human modeling efforts. Benefited from its automatic configuration features, xWeaver can support a variety of DCN architectures. Moreover, it further supports the flexible optimization on self-defined objectives that the network operators prefer. While maintaining the above desired properties, our experiments on an OCS-based testbed and extensive simulations demonstrate that xWeaver can achieve much shorter flow completion time and higher throughput than those of conventional topology configuration solutions.

ACKNOWLEDGMENTS

We thank our shepherd Vishal Misra and the anonymous reviewers for their feedbacks. This work is supported by National Key R&D Program of China (no. 2017YFB1010002) and NSFC Project (no. 61422206). Xin Wang's research is supported by NSF CNS 1526843.

REFERENCES

[1] 2012. Grante library. http://www.nowozin.net/sebastian/grante.
[2] 2017. Floodlight. http://www.projectfloodlight.org.
[3] 2017. iperf tool. https://iperf.fr.
[4] 2017. NS-2 Simulator. https://www.nsnam.org.
[5] 2018. UGM library. http://www.cs.ubc.ca/~schmidtm/Software/UGM.html.
[6] Mohammad Al-Fares, Alexander Loukissas, and Amin Vahdat. 2008. A scalable, commodity data center network architecture. In *SIGCOMM*.
[7] Mohammad Al-Fares, Sivasankar Radhakrishnan, Barath Raghavan, Nelson Huang, and Amin Vahdat. 2010. Hedera: Dynamic Flow Scheduling for Data Center Networks.. In *NSDI*.
[8] Mohammad Alizadeh, Albert Greenberg, David A Maltz, Jitendra Padhye, Parveen Patel, Balaji Prabhakar, Sudipta Sengupta, and Murari Sridharan. 2010. Data center tcp (dctcp). In *SIGCOMM*.
[9] Wei Bai, Li Chen, Kai Chen, and Haitao Wu. 2016. Enabling ECN in multi-service multi-queue data centers. In *NSDI*.
[10] Theophilus Benson, Aditya Akella, and David A Maltz. 2010. Network traffic characteristics of data centers in the wild. In *IMC*.
[11] Theophilus Benson, Ashok Anand, Aditya Akella, and Ming Zhang. 2010. Understanding data center traffic characteristics. *ACM SIGCOMM Computer Communication Review* 40, 1 (2010), 92–99.
[12] Shaileshh Bojja, Mohammad Alizadeh, and Pramod Viswanath. 2016. Costly Circuits, Submodular Schedules and Approximate Carathéodory Theorems. In *SIGMETRICS*.
[13] Kai Chen, Anubhav Singla, Ashutosh Singh, Kishore Ramachandran, Lei Xu, Yueping Zhang, Xitao Wen, and Yan Chen. 2014. OSA: an optical switching architecture for data center networks with unprecedented flexibility. In *NSDI*.
[14] Li Chen, Kai Chen, Joshua Zhu, Minlan Yu, George Porter, Chunming Qiao, and Shan Zhong. 2017. Enabling Wide-Spread Communications on Optical Fabric with MegaSwitch. In *NSDI*.
[15] Trishul Chilimbi, Yutaka Suzue, Johnson Apacible, and Karthik Kalyanaraman. 2014. Project adam: Building an efficient and scalable deep learning training system. In *OSDI*.
[16] Yong Cui, Shihan Xiao, Xin Wang, Zhenjie Yang, Chao Zhu, Xiangyang Li, Liu Yang, and Ning Ge. 2016. Diamond: nesting the data center network with wireless rings in 3D space. In *NSDI*.
[17] Jack Edmonds. 1965. Paths, trees, and flowers. *Canadian Journal of mathematics* 17, 3 (1965), 449–467.
[18] Nathan Farrington, George Porter, Sivasankar Radhakrishnan, Hamid Hajabdolali Bazzaz, Vikram Subramanya, Yeshaiahu Fainman, George Papen, and Amin Vahdat. 2010. Helios: a hybrid electrical/optical switch architecture for modular data centers. In *SIGCOMM*.
[19] Monia Ghobadi, Ratul Mahajan, Amar Phanishayee, Nikhil Devanur, Janardhan Kulkarni, Gireeja Ranade, Pierre-Alexandre Blanche, Houman Rastegarfar, Madeleine Glick, and Daniel Kilper. 2016. ProjecToR: Agile Reconfigurable Data Center Interconnect. In *SIGCOMM 2016*. ACM, 216–229.
[20] Phillipa Gill, Navendu Jain, and Nachiappan Nagappan. 2011. Understanding network failures in data centers: measurement, analysis, and implications. In *ACM SIGCOMM Computer Communication Review*, Vol. 41. ACM, 350–361.

[21] Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. 2017. Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour. *Facebook Inc., arXiv preprint:1706.02677* (2017).

[22] Albert Greenberg, James R Hamilton, Navendu Jain, Srikanth Kandula, Changhoon Kim, Parantap Lahiri, David A Maltz, Parveen Patel, and Sudipta Sengupta. 2009. VL2: a scalable and flexible data center network. In *SIGCOMM*.

[23] Daniel Halperin, Srikanth Kandula, Jitendra Padhye, Paramvir Bahl, and David Wetherall. 2011. Augmenting data center networks with multi-gigabit wireless links. In *SIGCOMM*.

[24] Navid Hamedazimi, Zafar Qazi, Himanshu Gupta, Vyas Sekar, Samir R Das, Jon P Longtin, Himanshu Shah, and Ashish Tanwer. 2014. FireFly: a reconfigurable wireless data center fabric using free-space optics. In *SIGCOMM*.

[25] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*. 1026–1034.

[26] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. 2014. Caffe: Convolutional architecture for fast feature embedding. In *Proceedings of the 22nd ACM international conference on Multimedia*. ACM, 675–678.

[27] Xin Jin, Yiran Li, Da Wei, Siming Li, Jie Gao, Lei Xu, Guangzhi Li, Wei Xu, and Jennifer Rexford. 2016. Optimizing Bulk Transfers with Software-Defined Optical WAN. In *SIGCOMM*.

[28] Srikanth Kandula, Sudipta Sengupta, Albert Greenberg, Parveen Patel, and Ronnie Chaiken. 2009. The nature of data center traffic: measurements & analysis. In *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement conference*. ACM, 202–208.

[29] Andrej Karpathy, George Toderici, Sanketh Shetty, Thomas Leung, Rahul Sukthankar, and Li Fei-Fei. 2014. Large-scale video classification with convolutional neural networks. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition (CVPR)*. 1725–1732.

[30] Thomas N. Kipf and Max Welling. 2017. Semi-Supervised Classification with Graph Convolutional Networks. In *International Conference on Learning Representations (ICLR)*.

[31] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems (NIPS)*. 1097–1105.

[32] Sergey Legtchenko, Nicholas Chen, Daniel Cletheroe, Antony Rowstron, Hugh Williams, and Xiaohan Zhao. 2016. XFabric: a reconfigurable in-rack network for rack-scale computers. In *NSDI*.

[33] He Liu, Feng Lu, Alex Forencich, Rishi Kapoor, Malveeka Tewari, Geoffrey M Voelker, George Papen, Alex C Snoeren, and George Porter. 2014. Circuit switching under the radar with reactor. In *NSDI*.

[34] He Liu, Matthew K Mukerjee, Conglong Li, Nicolas Feltman, George Papen, Stefan Savage, Srinivasan Seshan, Geoffrey M Voelker, David G Andersen, Michael Kaminsky, et al. 2015. Scheduling techniques for hybrid circuit/packet networks. In *CoNEXT*.

[35] Yunpeng James Liu, Peter Xiang Gao, Bernard Wong, and Srinivasan Keshav. 2014. Quartz: a new design element for low-latency DCNs. In *SIGCOMM*.

[36] William M Mellette, Rob McGuinness, Arjun Roy, Alex Forencich, George Papen, Alex C Snoeren, and George Porter. 2017. RotorNet: A Scalable, Low-complexity, Optical Datacenter Network. In *SIGCOMM*.

[37] Mohammad Noormohammadpour and Cauligi S Raghavendra. 2017. Datacenter Traffic Control: Understanding Techniques and Trade-offs. *IEEE Communications Surveys & Tutorials* (2017).

[38] Katia Obraczka and Peter Danzig. 1997. Finding low-diameter, low edge-cost, networks. *Univ. Southern California Technical Report* (1997).

[39] Naoaki Okazaki. 2007. CRFsuite: a fast implementation of Conditional Random Fields (CRFs). http://www.chokkan.org/software/crfsuite/

[40] Kay Ousterhout, Ryan Rasti, Sylvia Ratnasamy, Scott Shenker, and Byung-Gon Chun. 2015. Making sense of performance in data analytics frameworks. In *NSDI*.

[41] George Porter, Richard Strong, Nathan Farrington, Alex Forencich, Pang Chen-Sun, Tajana Rosing, Yeshaiahu Fainman, George Papen, and Amin Vahdat. 2013. Integrating microsecond circuit switching into the data center. In *SIGCOMM*.

[42] Arjun Roy, Hongyi Zeng, Jasmeet Bagga, George Porter, and Alex C Snoeren. 2015. Inside the social network's (datacenter) network. In *SIGCOMM*.

[43] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. 2009. The graph neural network model. *IEEE Transactions on Neural Networks* 20, 1 (2009), 61–80.

[44] Brandon Schlinker, Radhika Niranjan Mysore, Sean Smith, Jeffrey C Mogul, Amin Vahdat, Minlan Yu, Ethan Katz-Bassett, and Michael Rubin. 2015. Condor: Better topologies through declarative design. In *SIGCOMM*.

[45] Jürgen Schmidhuber. 2015. Deep learning in neural networks: An overview. *Neural Networks* 61 (2015), 85–117.

[46] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. 2016. Mastering the game of Go with deep neural

networks and tree search. *Nature* 529, 7587 (2016), 484–489.

[47] Arjun Singh, Joon Ong, Amit Agarwal, Glen Anderson, Ashby Armistead, Roy Bannon, Seb Boving, Gaurav Desai, Bob Felderman, Paulie Germano, et al. 2015. Jupiter rising: A decade of clos topologies and centralized control in google's datacenter network. In *SIGCOMM*.

[48] Charles Sutton and Andrew McCallum. 2011. An Introduction to Conditional Random Fields. *Machine Learning* 4, 4 (2011), 267–373.

[49] Jie Tang, Jing Zhang, Limin Yao, Juanzi Li, Li Zhang, and Zhong Su. 2008. Arnetminer: extraction and mining of academic social networks. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 990–998.

[50] Vincent Vanhoucke, Andrew Senior, and Mark Z Mao. 2011. Improving the speed of neural networks on CPUs. In *NIPS*.

[51] Guohui Wang, David G Andersen, Michael Kaminsky, Konstantina Papagiannaki, TS Ng, Michael Kozuch, and Michael Ryan. 2010. c-Through: Part-time optics in data centers. In *SIGCOMM*.

[52] Mowei Wang, Yong Cui, Xin Wang, Shihan Xiao, and Junchen Jiang. 2017. Machine Learning for Networking: Workflow, Advances and Opportunities. *IEEE Network* (2017).

[53] Yiting Xia, Xiaoye Steven Sun, Simbarashe Dzinamarira, Dingming Wu, Xin Sunny Huang, and TS Ng. 2017. A Tale of Two Topologies: Exploring Convertible Data Center Network Architectures with Flat-tree. In *SIGCOMM*.

[54] Muhan Zhang and Yixin Chen. 2018. Link Prediction Based on Graph Neural Networks. *arXiv preprint arXiv:1802.09691* (2018).

[55] Xia Zhou, Zengbin Zhang, Yibo Zhu, Yubo Li, Saipriya Kumar, Amin Vahdat, Ben Y Zhao, and Haitao Zheng. 2012. Mirror mirror on the ceiling: flexible wireless links for data centers. In *SIGCOMM*.

[56] Yibo Zhu, Xia Zhou, Zengbin Zhang, Lin Zhou, Amin Vahdat, Ben Y Zhao, and Haitao Zheng. 2014. Cutting the cord: a robust wireless facilities network for data centers. In *MOBICOM*.

Received February 2018; revised April 2018; accepted June 2018