DeepConf: Automating Data Center Network Topologies Management with Machine Learning

Saim Salman
Brown University

Christopher Streiffer
Duke University

Huan Chen
UESTC

Theophilus Benson
Brown University

Asim Kadav
NEC Labs

ABSTRACT

In recent years, many techniques have been developed to improve the performance and efficiency of data center networks. While these techniques provide high accuracy, they are often designed using heuristics that leverage domain-specific properties of the workload or hardware.

In this vision paper, we argue that many data center networking techniques, e.g., routing, topology augmentation, energy savings, with diverse goals share design and architectural similarities. We present a framework for developing general intermediate representations of network topologies using deep learning that is amenable to solving a large class of data center problems. We develop a framework, DeepConf, that simplifies the process of configuring and training deep learning agents by using our intermediate representation to learn different tasks. To illustrate the strength of our approach, we implemented and evaluated a DeepConf-agent that tackles the data center topology augmentation problem. Our initial results are promising — DeepConf performs comparably to the optimal solution.

CCS CONCEPTS

• **Networks → Programmable networks**; • **Computing methodologies → Reinforcement learning**;

KEYWORDS

Data center networks, deep reinforcement learning, topology management

ACM Reference Format:
Saim Salman, Christopher Streiffer, Huan Chen, Theophilus Benson, and Asim Kadav. 2018. DeepConf: Automating Data Center Network Topologies Management with Machine Learning. In *NetAI '18: ACM SIGCOMM 2018 Workshop on Network Meets AI & ML, August 24, 2018, Budapest, Hungary.* ACM, New York, NY, USA, 7 pages.
https://doi.org/10.1145/3229543.3229554

1 INTRODUCTION

Data center networks (DCN) are a crucial and important part of the Internet's ecosystem. The performance of these DCNs can impact a wide variety of services ranging from web browsing and videos to Internet of Things: the poor performance of these DCNs can result in as much as a $4 million loss in revenue [1]. Motivated by the importance of these networks, the networking community has explored techniques for improving and managing the performance of data center networks by: (1) designing better routing or traffic engineering algorithms [6, 8, 12, 15], (2) enriching the fixed topology with a limited number of flexible links [13, 18, 19, 37], and (3) removing corrupted and underutilized links from the topology [16, 20, 39].

Regardless of the approach, these topology-oriented techniques have three things in common: (1) Each is formalized as an optimization problem with a corresponding Linear Program (LP) or Integer Linear Program (ILP) solution. (2) Due to the impracticality of scalably solving these optimizations, greedy heuristics are employed to create approximate solutions. (3) Most of these heuristics do not generalize because they are intricately tied to the high-level application patterns and technological constraints. In particular, determining the optimal routes, the optimal location to add augmenting links, or the optimal set of links to remove under diverse and rapidly evolving conditions is challenging (even NP-hard) [13, 18, 19, 37]. Existing domain-specific heuristics provide suboptimal performance and are often limited to specific scenarios. Thus as a community, we are forced to revisit and redesign these heuristics whenever the application pattern or network details changes – even a minor change. For example, while c-through [37] and FireFly [19] solve broadly identical problems, they leverage different heuristics to account for hardware differences.

In this vision paper, we articulate our vision for replacing domain-specific rule-based heuristics for topology management with a more general machine learning-based (ML) model that quickly learns optimal solutions to a class of problems while adapting to changes in the application patterns, network dynamics, and low-level network details. Unlike recent attempts that employ ML to learn point solutions, e.g., cluster scheduling [25] or routing [9], in this paper, we present a general framework, called DeepConf, that simplifies the process of designing ML models for a broad range of DCN topology problems and eliminates the challenges associated with efficiently training new models.

The key challenges in designing DeepConf are: (1) tackling the dichotomy that exists between deep learning's requirements for large amounts of supervised data and the unavailability of these required datasets, and (2) designing a general, but highly accurate, deep learning model that efficiently generalizes to learning a broad array of data center problems ranging from topology management and routing to energy savings.

The critical insight underlying DeepConf is that intermediate features generated from the parallel convolutional layers using network data, e.g., traffic matrix, allows us to generate an intermediate representation of the network's state that enables learning a broad range of data center problems. Moreover, while labeled production data crucial for machine learning is unavailable, empirical studies [23] show that modern data center traffic is highly predictable and thus amenable to offline learning with network simulators and historical traces.

DeepConf builds on these insights by using reinforcement learning (RL), a deep learning technique, that learns through experience and makes no assumptions on how the network works. Rather, they are trained through the use of a reward signal which "guides" them towards an optimal solution and thus do not require real-world supervised data, and, instead, they can be trained using simulators.

The DeepConf framework provides a predefined RL model with the intermediate representation, a specific design for configuring this model to address different problems, an optimized simulator to enable efficient learning, and a Software-defined networking (SDN) platform for capturing network data and reconfiguring the network.

In this paper, we make the following contributions:

- We present a novel RL-based SDN architecture for developing and training deep ML models for a broad range of DCN tasks.
- We design a novel input feature extraction for DCNs for developing different ML models over this intermediate representation of network state.
- We implemented a DeepConf-agent tackling the topology augmentation problem and evaluated it on representative topologies [4, 17] and traces [3], showing that our approach performs comparable to optimal.

2 RELATED WORK

Our work is motivated by the recent success of applying machine learning and RL algorithms to computer games and robotic planning [27, 33, 35]. The most closely related works [9, 36] apply RL to Internet service provider (ISP) networks. Unlike these approaches, DeepConf focuses on data center networks which have significantly larger scale and higher velocity than traditional ISP networks. Additionally, while these approaches [9, 36] focus on network routing, DeepConf tackles the topology augmentation problem and explores the use of deep networks as function approximators for RL. Existing applications of machine learning to data centers focus on improving cluster scheduling [25] and more recently by Google to optimize Power Usage Effectiveness (PUE) [2]. In this vision paper, we take a different stance and focus on identifying a class of equivalent data center management operations, namely topology management and configuration, that are amenable to a common machine learning approach and design a modular system that enables different agents to interoperate over a network.

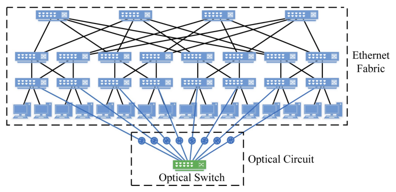

Figure 1: A k=4 Fat Tree topology with an optical switch.

3 BACKGROUND

This section provides an overview of data center networking challenges and solutions and provides background on our reinforcement learning methods.

3.1 Data Center Networks

Data center networks introduce a range of challenges from topology design and routing algorithms to VM placement and energy-saving techniques. In this section, we focus on a subset of these problems and illustrate high-level patterns that arise across these problems and their solutions.

Data centers support a large variety of workloads and applications with time-varying bandwidth requirements. This variance in bandwidth requirements leads to hotspots at varying locations and at different points in time. To support these arbitrary bandwidth requirements, data center operators can employ non-blocking topologies; however, non-blocking topologies are prohibitively costly. Instead, these operators employ techniques ranging from hybrid architectures [13, 18, 19, 37] and traffic engineering algorithms [6, 8, 12, 15] to energy saving techniques [20, 30]. Below, we describe these techniques and illustrate common design requirements and patterns.

Augmented Architectures: This class of approaches build on the intuition that at any given point in time, there are only a small number of hotspots. Thus, there is no need to use a non-blocking topology, and the existing topology can be augmented with a small amount of links to support these demands. For example, Figure 1 shows a traditional Fat-Tree topology supplemented with an optical switch. Additionally, over time these links can be moved around to different locations as the hotspots move. These approaches augment the data center's electrical packet switched network with a small number of flexible links (optical [13, 37], wireless [18], or free optics [19]. [1]) These proposals argue for monitoring the traffic, using an Integer Linear Program (ILP) or heuristic to detect the hotspots and placing these flexible links at the locations with these hotspots.

Traffic Engineering: Orthogonal approaches [6, 8, 12, 15] focus on routing. Instead of changing the topologies, these approaches change the mapping of flows to paths within a fixed topology. These proposals, also, argue for monitoring traffic and detecting hotspots. Instead of changing topologies, these techniques move a subset of flows from congested links to un-congested links.

Energy Savings Data centers are notorious for their energy usage [16]. To address this, researchers have proposed techniques to improve energy efficiency by detecting periods of low utilization and selectively turning off links [20, 30]. These proposals argue for

[1]The number of augmented links is significantly smaller than the number of data center's permanent links.

monitoring traffic, detecting low utilization, and powering-down links in portions of the data center with low utilizations. A key challenge with these techniques is to turn on the powered-down links before demands rise.

Taking a step back, these techniques roughly follow the same design and operate in three steps (1) gather network traffic matrix, (2) run an ILP to predict heavy (or low) usage, and (3) perform a specific action on a subset of the network. The actions range from augmenting flexible links, turning off links, or moving traffic. In all situations, the ILP does not scale to a large network and a domain-specific heuristic is often used in its place.

These set of tasks are ideal for a form of deep learning called deep reinforcement learning — where in an algorithm learns through experience the set of actions to perform under a variety of conditions. Unlike supervised learning techniques which require a significantly large corpus of labeled-data which is prohibitively expensive, unsupervised learning techniques, e.g., DeepRL, learns from unlabeled-data and merely requires data sets and a simulation.

3.2 Reinforcement Learning

Reinforcement learning (RL) algorithms learn through experience with a goal towards maximizing rewards. Unlike supervised learning where algorithms train over labels, RL algorithms learn by interacting with an environment such as a game or a network simulator.

In traditional RL, an agent interacts with an environment over a number of discrete time steps. Hence, at each time step t, the agent in a *world* observes a state s_t in order to select an action a_t from a possible action set A. The agent is guided by policy, π, which is a function that maps state s_t to actions a_t. The agent receives a reward r_t for each action and transitions to the next state s_{t+1}. The goal of the agent is maximizing the total reward. This process continues until the agent reaches a final state or time limit, after which the environment is reset, and a new training episode is played. After a number of training episodes, the agent learns to pick actions that maximize the rewards and can learn to handle unexpected states. RL is effective and has been successfully used to model robotics, game bots, etc.

Instead of learning over all state-action predictions, policy based methods directly learn the policy to improve learning efficiency. The goal of commonly used policy-based RL is to find a policy, π, that maximizes the cumulative reward and converges to a theoretical optimal policy. In deep policy-based methods, a neural network computes a policy distribution $\pi(a_t|s_t;\theta)$, where θ represents the set of parameters of the function. Deep networks as function approximators is a recent development and other learning methods can be used. We now describe the REINFORCE and actor-critic policy methods which represent different methods to score the policy $J(\theta)$. REINFORCE methods [38] use gradient ascent on $\mathbb{E}[R_t]$, where $R_t = \sum_{i=0}^{\infty} \gamma^i r_{t+i}$ is the accumulated reward starting from time step t and discounted at each step by $\gamma \in (0, 1]$, the discount factor. The REINFORCE method, which is the Monte-Carlo method, updates θ using the gradient $\nabla_\theta \log \pi(a_t|s_t;\theta)R_t$, which is an unbiased estimator of $\nabla_\theta \mathbb{E}[R_t]$. The value function is computed as $V^\pi(s_t) = \mathbb{E}[R_t|s_t]$ which is the expected return for following the policy π in state s_t. This method provides actions with high returns but suffers from high-variance of gradient estimates.

Asynchronous Advantage Actor Critic (A3C): A3C [26] improves REINFORCE performance by operating asynchronously and by using a deep network to approximate the policy and value faction. A3C uses the actor-critic method which additionally computes a critic function that approximates the value function. A3C, as used by us, uses a network with two convolutional layers followed by a fully connected layer. Each hidden layer is followed by a nonlinearity function (ReLU). A softmax layer which approximates the policy function and a linear layer to output an estimate of the value function $V(s_t;\theta)$ together constitute the output of this network. Asynchronous gradient descent using multiple agents is used to train the network, and this improves the training speed. A central server (similar to a parameter server) coordinates the parallel agents – each agent calculates the gradients and sends the updates to the server after a fixed number of steps, or when a final state is reached. Furthermore, following each update, the central server propagates new weights to the agents to achieve a consensus on the policy values. There is a cost function with each deep network (policy and value). Using two loss functions has found to improve convergence and produce better-regularized models. The policy cost function is given as:

$$f_\pi(\theta) = \log \pi(a_t|s_t;\theta)(R_t - V(s_t;\theta_t)) + \beta H(\pi(s_t;\theta)) \quad (3.1)$$

where θ_t represents the values of the parameters θ at time t, $R_t = \sum_{i=0}^{k-1} \gamma^i r_{t+i} + \gamma^k V(s_{t+k};\theta_t)$ is the estimated discounted reward. $H(\pi(s_t;\theta))$ is used to favor exploration and its strength is controlled by the factor β.The cost function for the estimated value function is:

$$f_v(\theta) = (R_t - V(s_t;\theta))^2 \quad (3.2)$$

Additionally, we augment our A3C model to learn current states apart from accumulating rewards for good configurations using GAE [32]. The deep network which replaces the transition matrix as the function approximator learns the value of the given state and the policy of the given state. The model uses GAE to compute the value of a given state that not only returns the reward for the model for the given policy decision but also rewards the model for estimating the value of the state. This reward helps to guide the model to learn the states instead of just maximizing rewards.

4 VISION

Our vision is to automate a subset of data center management and operational tasks by leveraging DeepRL. At a high-level, we anticipate the existence of several DeepRL agents, each trained for a specific set of tasks, e.g., traffic-engineering, energy-savings, or topology-augmentations. Each agent will run as an application atop an SDN controller. The use of SDN provides the agents with an interface for gathering network state and a mechanism for enforcing actions. For instance, DeepConf should be able to assemble the traffic matrix by polling the different devices within the network, compute a decision for how the optical switch should be best configured to accommodate the current network load and reconfigure the network.

At a high-level, DeepConf's architecture consists of three components (Figure 2): the network simulator to enable offline training of the DeepRL agents, the DeepConf abstraction layer to facilitate communication between the DeepRL agents and the network, and the DeepRL agents, called DeepConf-agents, which encapsulate

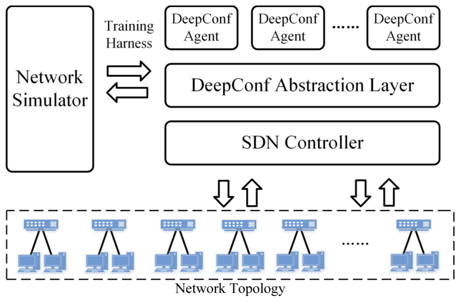

Figure 2: DeepConf Architecture.

data center functionality. Realizing this vision introduces several interdisciplinary challenges across the networking, systems, and AI communities.

Applying learning: The primary challenges in applying machine learning to network problems are (1) the deficiency of training data pertaining to operator and network behavior and (2) the lack of models and loss functions that can accurately model the problem and generalize to unexpected situations. This shortage of data presents a roadblock for using supervised ML approaches which require data. To address this issue, DeepConf uses RL where the model is trained by exploring different network states and environments generated by a simulators. Coupled with the wide availability of network job traces, this allows for DeepConf to learn a highly generalizable policy.

DeepConf Abstraction Layer: Today's SDN controllers expose a primitive interface with low-level information. The DeepConf applications will instead require high-level abstractions and interfaces. For example, our agents will require interfaces that provide control over paths rather than overflow table entries. While emerging approaches [21] argue for similar interfaces, these approaches do not provide a sufficiently rich set of interfaces for the broad range of agents we expect to support and do not provide composition primitives for safely combining the output from the different agents. Existing composition operators [14, 22, 29] assume that the different SDN applications (or DeepConf-agent in our case) are generating non-conflicting actions – hence these operators can not tackle conflicting actions. SDN Composition approaches [7, 28, 31] that do tackle conflicting actions, require a significant rewrite of the SDNApps.

More concretely, we require high-layer SDN abstractions that enable our DeepConf-agent to more easily learn and act on the network. Additionally, we require novel composition operators that can reason about and tackle conflicting actions generated by the different DeepConf-agent.

Domain-specific Simulators: Existing SDN research leverages a popular emulation platform, Mininet, which fails to scale to large experiments. A key requirement for employing DeepRL is to have efficient and scalable simulators that replay traces and enables learning from these traces [34]. We build on existing flow-based simulators [10, 11] to model the various dimensions that are required to train our models. To improve efficiency, we plan to explore techniques that partition the simulation and enables reuse of results — in essence, to allow *incremental simulations*.

In addressing our high-level vision and developing solutions to the above challenges, there are several high-level goals that a production-scale system must address: (1) our techniques must generalize across topologies, traffic matrixes, and a range of operational settings, e.g., link failures; (2) our techniques must be as accurate and efficient as existing state-of-the-art techniques; and (3) our solutions must incur low operational overheads, e.g., minimizing optical switching time or TCAM utilization.

5 DESIGN

In this section, we provide a broad description of how to define and structure existing data center network management techniques as RL tasks, then describe the methods for training the resulting DeepConf-agents.

5.1 DeepConf Agents

In defining each new DeepConf-agent, there are four main functions that a developer must specify: state space, action space, learning model, and reward function. The action space and reward are both specific to the management task being performed and are, in turn, unique to the agent. Respectively, they express the set of actions an agent can take during each step and the reward for the actions taken. The state space and learning models are more general and can be shared and reused across different DeepConf-agents. This is because of the fundamental similarities shared between the data center management problems, and because the agents are specifically designed for data centers. The state-space consists of both general state space and specific state space — we discuss the general state space below and present a case study on a specific state space for the topology problem in Section 6.

Next, we focus on the general components, i.e., state-space and the learning model.

State Space: In general, the state space consists of two types of data – each reflecting the state of the network at a given point in time. First, the general network state that all DeepConf-agents require: the network's traffic matrix (TM) which contains information on the flows which were executed during the last *t* seconds of the simulation. Second, a DeepConf-agent specific state-space that captures the impact of the actions on the network. For example, for the topology augmentation problem, this would be the network topology – note, the actions change the topology. Whereas for the traffic engineering problem, this would be a mapping of flows to paths – note, the actions change a flow's routes.

Learning Model: Our learning model utilizes a Convolutional Neural Network (CNN) to compute policy decisions. The exact model for a DeepConf-agent depends on the number of state-spaces used as input. In general, the model will have as many CNN-blocks as there are state spaces – one CNN-block for each state space. The output of these blocks are concatenated together and input into two fully connected layers, followed by a softmax output layer. For example, for the topology-augmentation problem, as observed in Figure 4, our DeepConf-agent has two CNN blocks to operate on both the topology and the TM states spaces in parallel. This model allows for the lower CNN layers to perform feature extraction on the input

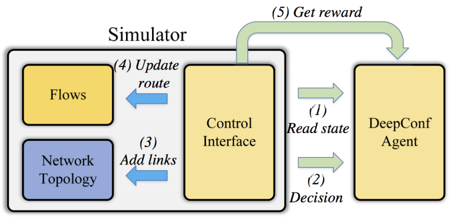

Figure 3: DeepConf-agent model training.

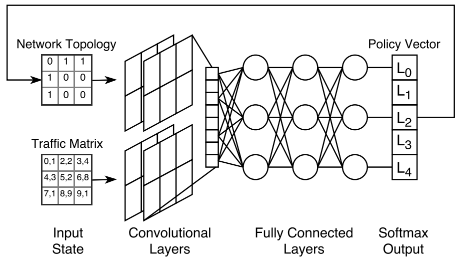

Figure 4: The CNN model utilized by the DeepConf-agent.

spatial data, and for the fully connected layers to assemble these features in a meaningful way.

5.2 Network Training

To train a DeepConf-agent, we run the agent against a network simulator. The interaction between the simulator and the DeepConf-agent can be described as follows (Figure 3): (1) The DeepConf-agent receives state s_t from the simulator at training step t. (2) The DeepConf-agent uses the state information to make a policy decision about the network and returns the selected actions to the simulator. For topology augmentation problem's DeepConf-agent, called the Augmentation-Agent, the actions are the set of K optical links to activate. (3) If the topology changes, the simulator re-computes the paths for the active flows. (4) The simulator executes the flows for x seconds. (5) The simulator returns the reward r_t and state s_{t+1} to the DeepConf-agent, and the process restarts.

Initialization During the initial training phase, we force the model to explore the environment by randomizing the selection process — the probability that an action is picked corresponds to the value of the index which represents the action. For instance, with the Augmentation-Agent, link i has probability w_i of being selected. As the model becomes more familiar with the states and corresponding values, the model will better formulate its policy decision. At this point, the model will associate a higher probability with the links it believes to have a higher reward, which will cause these links to be selected at a higher frequency. This methodology allows for the model to *reinforce* its decisions about links, while the randomization helps the model avoid local-minima.

Learning Optimization: To improve the efficiency of learning, the RL agent maintains a log containing the state, policy decision, and corresponding reward. The RL agent performs *experience replay* after n simulation steps. During replay, the log is unrolled to compute the policy loss across the specified number of steps using Equation 3.1. The agent is trained using Adam stochastic optimization [24], with an initial learning rate of 10^{-4} and a learning rate decay factor of 0.95. We found that a smaller learning rate and low decay helped the model better explore the environment and form a more optimal policy.

6 USE CASE: TOPOLOGY AUGMENTATION

More formally defined, in the topology augmentation problem the data center consists of a fixed hierarchical topology with electrical packet switches and an optical switch, which connects all the top-of-rack switches. While the optical switch is physically connected to all ToR switches, unfortunately, the optical switch can only support a limited number of active links. Given this limitation, the rules for the topology problem are defined as: (i) The model must select K links to activate at a given step during the simulation. (ii) The model receives a reward based on the link utilization and the flow duration. (iii) The model collects the reward on a per-link basis after x seconds of simulation. (iv) All flows are routed using equal-cost multi-path routing (ECMP).

State Space: The agent specific state space is the network topology, which is represented by a sparse matrix where entries within the cells correspond to active links within the network.

Action Space: The agent interacts with the environment by adding links between ToR switches. The *action space* for the model, therefore, corresponds to the different possible link combinations and is represented as an n dimensional vector. The values within the vector correspond to a probability distribution, where w_i is equal to the probability of link i being the optimal pick for the given input state s. The model selects the highest K values from this distribution as the links that should be added to the network topology.

Reward: The goal of the model can be summarized as: (1) Maximize link utilization and (ii) Minimize the average flow-completion time. With this in mind, we formulate our reward function as:

$$R(\Theta, s, t) = \sum_{f \in F} \sum_{l \in f} \frac{b_f}{d_f} \tag{6.1}$$

Where F represents all active and completed flows during the previous iteration step, l represents the links used by flow f, b_f represents the number of bytes transferred during the step time, and d_f represents the total duration of the flow. The purpose of this reward function is to reward for high link utilization but penalize for long lasting flows. The design of this function has the effect of guiding the model towards completing large flows within a smaller period of time.

7 EVALUATION

In this section, we analyze DeepConf under realistic workload with representative topologies.

Experiment Setup We evaluate DeepConf on a trace driven flow-level simulator using a large scale map-reduce traces from Facebook [3].We evaluate two state-of-the-art clos-style data center topologies: K=4 Fat-tree [5] and VL2 [17]. In our analysis, we focus on flow completion time (FCT) a metric which captures the duration between the first and last packet of a flow. We augment both topologies by adding an optical switch with four links. We compare DeepConf against: **Optimal**, the optimal solution derived from a linear program with perfect knowledge of future application demands — Note: this optimal solution can not be solved with larger topologies [8].

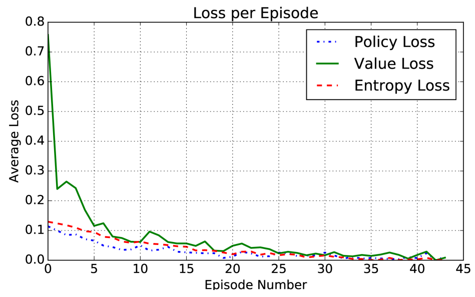

Figure 5: This figure shows loss value vs number of episodes and that our model can learn in reasonable number of episodes.

7.1 Model Learning

Figure 5 shows the performance of the DeepConf-agent during training on a per-episode basis: the loss decreases as training increases with the largest decrease occuring during the initial training episodes, a result consistent with the learning rate decay factor employed during training.

Figure 6 presents the performance of each independent agent and we observe that each agent independently maximizes the total reward for each episode. Around episode 35-40, the performance of both agents begins to plateau as the policy becomes fine-tuned towards maximizing the reward.

The training results demonstrate that the RL agent learns to optimize its policy decision to increase the total reward received across each episode.

7.2 DeepConf Performance

Figure 7 shows that DeepConf is able to learn a solution that's close to the optimal [13, 37] across representative data center topologies. Other results, not included due to space constraints, show that DeepConf outperforms greedy heuristics [13, 37] in terms of FCT and remains competitive with the optimal solution.

We believe these initial results are promising, and that more work is required in order to understand and improve the performance of DeepConf.

8 DISCUSSION

We now discuss open questions:

Learning to generalize: In order to avoid over-fitting to a specific topology, we train our agent over a large number of simulator configurations. DeepRL agents need to be trained and evaluated

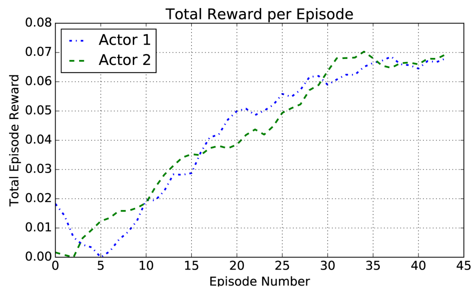

Figure 6: This figure shows that the model is able to maximize the rewards over the training period.

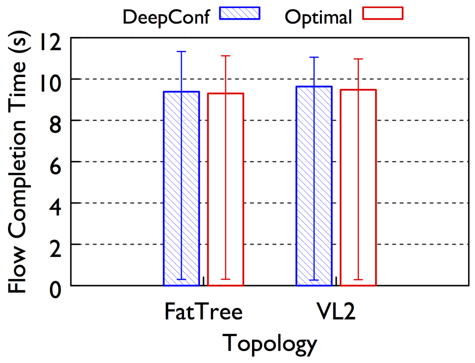

Figure 7: Mean flow completion time (Error Bars (5th and 95th percentiles)).

on many different platforms to avoid being overtly specific to few networks and correctly handle unexpected scenarios. Solutions that employ machine learning to address network problems using simulators need to be cognizant of these issues when deciding the training data.

Learning new reward functions: DeepRL methods need appropriate reward functions to ensure that they optimize for the correct goals. For some networks problems like topology configuration this may be straightforward. However, other problems like routing may require a weighted combination of network parameters that need to be correctly designed for the agent to operate the network correctly.

Learning other data center problems. In this paper, we focused on problems that center around learning to adjust the topology and routing. Yet, the space of data center problems is much larger. As part of ongoing work, we are investigating intermediate representations and models for capturing other high-level tasks.

9 CONCLUSION

Our high-level goal is to develop ML-based systems that replace existing heuristic-based approaches to tackling data center networking challenges. This shift from heuristics to ML will enable us to design solutions that can adapt to changes in patterns by consuming data and relearning – an automated task.

In this paper, we take the first steps towards achieving these goals by designing a reinforcement learning based framework, called DeepConf, for automatically learning and implementing a range of data center networking techniques. This framework is supported by a novel intermediate representation of network state that is amenable to creating different sets of reinforcement learning models for different tasks and an SDN-based architecture that abstracts away low level network details. We believe that DeepConf represents a promising step towards a datacenter framework that eschews human generated domain-specific heuristics in favor of automated and general algorithms generated by a machine learning framework. As part of future work, we are planning to extend DeepConf to tackle the routing problem, to perform a broader evaluation of DeepConf across different topologies, and to explore the use of interpretable techniques to explain the decisions made by DeepConf's deep models.

REFERENCES

[1] Amazon found every 100ms of latency cost them 1% in sales. https://blog.gigaspaces.com/amazon-found-every-100ms-of-latency-cost-them-1-in- sales/.

[2] DeepMind AI Reduces Google Data Centre Cooling Bill by 40%. https://deepmind.com/blog/deepmind-ai-reduces-google-data-centre-cooling-bill-40/.

[3] Statistical Workload Injector for MapReduce . https://github.com/SWIMProjectUCB/SWIM/ wiki.

[4] M. Al-Fares, A. Loukissas, and A. Vahdat. A scalable, commodity data center network architecture. In *Proceedings of ACM SIGCOMM 2008*.

[5] M. Al-Fares, A. Loukissas, and A. Vahdat. A scalable, commodity data center network architecture. In *Proceedings of ACM SIGCOMM 2008*.

[6] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat. Hedera: Dynamic flow scheduling for data center networks. In *Proceedings of USENIX NSDI 2010*.

[7] A. AuYoung, Y. Ma, S. Banerjee, J. Lee, P. Sharma, Y. Turner, C. Liang, and J. C. Mogul. Democratic resolution of resource conflicts between sdn control programs. In *CoNext*, 2014.

[8] T. Benson, A. An, A. Akella, and M. Zhang. Microte: The case for fine-grained traffic engineering in data centers. In *Proceedings of ACM CoNEXT 2011*.

[9] J. A. Boyan, M. L. Littman, et al. Packet routing in dynamically changing networks: A reinforcement learning approach. *Advances in neural information processing systems*, pages 671–671, 1994.

[10] H. Chen and T. Benson. The case for making tight control plane latency guarantees in sdn switches. In *Proceedings of the Symposium on SDN Research*, SOSR '17, pages 150–156, New York, NY, USA, 2017. ACM.

[11] H. Chen and T. Benson. Hermes: Providing tight control over high-performance sdn switches. In *Proceedings of the 13th International Conference on Emerging Networking EXperiments and Technologies*, CoNEXT '17, pages 283–295, New York, NY, USA, 2017. ACM.

[12] A. Das, C. Lumezanu, Y. Zhang, V. K. Singh, G. Jiang, and C. Yu. Transparent and flexible network management for big data processing in the cloud. In *Proceedings of ACM HotCloud 2013*.

[13] N. Farrington, G. Porter, S. Radhakrishnan, H. H. Bazzaz, V. Subramanya, Y. Fainman, G. Papen, and A. Vahdat. Helios: A hybrid electrical/optical switch architecture for modular data centers. In *Proceedings of ACM SIGCOMM 2010*.

[14] N. Foster, R. Harrison, M. J. Freedman, C. Monsanto, J. Rexford, A. Story, and D. Walker. Frenetic: A network programming language. *SIGPLAN Not.*, 46(9):279–291, Sept. 2011.

[15] S. Ghorbani, B. Godfrey, Y. Ganjali, and A. Firoozshahian. Micro load balancing in data centers with drill. In *Proceedings of ACM HotNets 2015*.

[16] A. Greenberg, J. Hamilton, D. A. Maltz, and P. Patel. The cost of a cloud: Research problems in data center networks. *SIGCOMM Comput. Commun. Rev.*, 39(1):68–73, Dec. 2008.

[17] A. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. A. Maltz, P. Patel, and S. Sengupta. Vl2: A scalable and flexible data center network. In *Proceedings of ACM SIGCOMM 2009*.

[18] D. Halperin, S. Kandula, J. Padhye, P. Bahl, and D. Wetherall. Augmenting data center networks with multi-gigabit wireless links. In *Proceedings of ACM SIGCOMM 2011*.

[19] N. Hamedazimi, Z. Qazi, H. Gupta, V. Sekar, S. R. Das, J. P. Longtin, H. Shah, and A. Tanwer. Firefly: A reconfigurable wireless data center fabric using free-space optics. In *Proceedings of ACM SIGCOMM 2014*.

[20] B. Heller, S. Seetharaman, P. Mahadevan, Y. Yiakoumis, P. Sharma, S. Banerjee, and N. McKeown. Elastictree: Saving energy in data center networks. In *Proceedings of USENIX NSDI 2010*.

[21] V. Heorhiadi, M. K. Reiter, and V. Sekar. Simplifying software-defined network optimization using sol. In *Proceedings of USENIX NSDI 2016*.

[22] X. Jin, J. Gossels, J. Rexford, and D. Walker. Covisor: A compositional hypervisor for software-defined networks. In *Proceedings of the 12th USENIX Conference on Networked Systems Design and Implementation*, NSDI'15, pages 87–101, Berkeley, CA, USA, 2015. USENIX Association.

[23] S. A. Jyothi, C. Curino, I. Menache, S. M. Narayanamurthy, A. Tumanov, J. Yaniv, R. Mavlyutov, I. Goiri, S. Krishnan, J. Kulkarni, and S. Rao. Morpheus: Towards automated slos for enterprise clusters. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, pages 117–134, GA, 2016. USENIX Association.

[24] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.

[25] H. Mao, M. Alizadeh, I. Menache, and S. Kandula. Resource management with deep reinforcement learning. In *Proceedings of the 15th ACM Workshop on Hot Topics in Networks*, pages 50–56. ACM, 2016.

[26] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International Conference on Machine Learning*, 2016.

[27] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[28] J. C. Mogul, A. AuYoung, S. Banerjee, L. Popa, J. Lee, J. Mudigonda, P. Sharma, and Y. Turner. Corybantic: Towards the modular composition of sdn control programs. In *HotNets*, 2013.

[29] C. Monsanto, N. Foster, R. Harrison, and D. Walker. A compiler and run-time system for network programming languages. In *Proceedings of ACM POPL 2012*.

[30] S. Nedevschi, L. Popa, G. Iannaccone, S. Ratnasamy, and D. Wetherall. Reducing network energy consumption via sleeping and rate-adaptation. In *Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation*, NSDI'08, pages 323–336, Berkeley, CA, USA, 2008. USENIX Association.

[31] S. Prabhu, M. Dong, T. Meng, P. B. Godfrey, and M. Caesar. Let me rephrase that: Transparent optimization in sdns. In *Proceedings of ACM SOSR 2017*.

[32] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*, 2015.

[33] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

[34] I. Stoica, D. Song, R. A. Popa, D. A. Patterson, M. W. Mahoney, R. H. Katz, A. D. Joseph, M. Jordan, J. M. Hellerstein, J. Gonzalez, K. Goldberg, A. Ghodsi, D. E. Culler, and P. Abbeel. A berkeley view of systems challenges for ai. Technical Report UCB/EECS-2017-159, EECS Department, University of California, Berkeley, Oct 2017.

[35] N. Usunier, G. Synnaeve, Z. Lin, and S. Chintala. Episodic exploration for deep deterministic policies: An application to starcraft micromanagement tasks. *arXiv preprint arXiv:1609.02993*, 2016.

[36] A. Valadarsky, M. Schapira, D. Shahaf, and A. Tamar. Learning to route. In *Proceedings of the 16th ACM Workshop on Hot Topics in Networks*, HotNets-XVI, pages 185–191, New York, NY, USA, 2017. ACM.

[37] G. Wang, D. G. Andersen, M. Kaminsky, K. Papagiannaki, T. Ng, M. Kozuch, and M. Ryan. c-through: Part-time optics in data centers. In *Proceedings of ACM SIGCOMM 2010*.

[38] R. J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.

[39] D. Zhuo, M. M. Ghobadi, R. Mahajan, K.-T. Forster, A. Krishnamurthy, and T. Anderson. Understanding and mitigating packet corruption in data center networks. page 14. ACM SIGCOMM, August 2017.