Reinforcement Learning based Load Balancing for Data Center Networks

Jiyoon Lim*, Jae-Hyoung Yoo*, and James Won-Ki Hong*

*Department of Computer Science and Engineering, POSTECH, Pohang, Korea
{limjiyoon, jhyoo78, jwkhong}@postech.ac.kr

2021 IEEE 7th International Conference on Network Softwarization (NetSoft) | 978-1-6654-0522-5/21/$31.00 ©2021 IEEE | DOI: 10.1109/NetSoft51509.2021.9492566

***Abstract*—Data center networks are designed with multi-rooted topologies to provide the large bisection bandwidth. These topologies provide high path diversity and need a load balancing algorithm to utilize bisection bandwidth efficiently. To distributes traffic efficiently, congestion-aware and fine-grained load balancing algorithms are suggested. However, these algorithms need to set parameters depending on the network states. This paper presents a reinforcement weight-cost multipath (RWCMP) load-balancing method on a data center network to balance the traffic. It uses reinforcement learning to learn the optimal traffic split ratio of egress ports for each node and determines the routes of multiple flows simultaneously. We compared the network utilization of our RWCMP load balancing algorithm with that of the LetFlow algorithm. In a fat tree topology, the RWCMP load balancing algorithm outperforms the traditional load balancing algorithm with 24-36% lower network utilization in three different traffic patterns. In the same topology with a link failure, the RWCMP load balancing algorithm outperforms the traditional load balancing algorithm with 12-27% lower network utilization. These results demonstrate that RWCMP learns traffic split ratio depending on underlying traffic patterns and topology of the data center.**

***Index Terms*—Datacenter Network, Load balancing, Reinforcement learning**

I. Introduction

Data center networks provide various services such as web services, streaming, social networking services. To provide a large number of services at once, data center networks require a large bisection bandwidth. Most data center networks uses multi-rooted topologies such as Clos topology [1] or fat-tree topology [2]. These topologies have multiple paths between each host pair and need a load balancing algorithm to utilize the multiple paths efficiently.

The Equal-Cost Multi-Path (ECMP) algorithm [3] is the canonical load balancing algorithm in data center networks. It distributes the flow evenly over each shortest path by using the static hashing of the packet headers. However, the ECMP algorithm can aggravate the degree of network congestion, as it distributes traffic regardless of network congestion. If the ECMP algorithm forwards several large flows to the same path upon the hash collisions, network congestion can occur. Congestion decreases the throughput and increases the latency.

The limitation of the ECMP algorithm can be addressed using two approaches. The first approach is using the fined scheduling granularity such as LetFLow [4] [5]. Per-flow load balancing is difficult to balance load optimally since it is known as an NP-hard problem. By contrast, Per-packet load balancing quickly choose an optimal path. However, packet reordering will occur and resulting in worse performance. In consideration of performance and avoiding packet reordering, load balancing algorithms with sub-flow level scheduling granularity such as a flowlet [5] are suggested. A flowlet is a sub-flow concept that divides a flow into sub-flow according to the flowlet timeout that is the threshold time interval to distinguish flowlet in a flow. The other approach is using network information with a centralized controller [6]. The centralized controller collects the global network information and reroutes the elephant traffic. These load balancing algorithms show higher throughput than the load balancing algorithms that do not use network information.

These approaches balance the traffic based on the snapshots of the network status. These algorithms can encounter two challenges. First, they react late to the mice flows because of the control plane delay, which is composed of both the transmission and processing delay between controllers and switches. For example, 7.5Mb flows can pass through a 1Gbps link [7] when the control plane delay is 60ms. Second, these load balancing algorithms need to be set parameters following the traffic patterns and topologies. Their performance decreases if parameters are mismatched. The PIAS load balancing algorithm [8] shows a 38.46% performance reduction upon such mismatch.

Here, we present a reinforcement learning-based weighted ECMP load balancing algorithm (RWCMP). It learns and adapts the traffic split ratios that minimize network utilization by using traffic and network information. In addition, it processes multiple flows at once to alleviate the problem of control plane delay. We validate our algorithm using three different traffic patterns on a regular fat tree topology and fat tree topology with a link failure. We compare the network utilization ratios of the LetFlow load balancing algorithm and RWCMP algorithm. We propose the following contributions:

1) We present a learning load balancing algorithm with reinforcement learning
2) Our method balances multiple flows simultaneously to handle mice and elephant flows both
3) Our method compares the network utilization of RWCMP with two different rewards and neural network models.

II. Background and Related Work

A. Background

Reinforcement learning is a machine learning paradigm that learns policies to maximize the cumulative reward. The reinforcement learning framework contains two components: agent and environment. The agent is a learner that decides the actions, whereas the environment is defined as anything that the agent cannot control. State is the set of variables in the environment. The agent observes the environment and estimates its current state. This is called observation. Then agent decides a policy that is the action probability of each state. Although state and observation are different terms, herein, we collectively refer them as a state in this paper.

Proximal policy optimization (PPO) [9] is approximate policy as policy parameters and update them with gradient method that constraints objective function for a stable policy update. It achieves good sample efficiency and high cumulative rewards.

B. Related work

LetFlow [4] is a simple load balancing algorithm with a flowlet-level scheduling granularity. LetFlow divides a flow into flowlets with a predefined flowlet timeout. Then, each switch randomly selects a path for each flowlet. This algorithm shows similar performance to other algorithms that use global network information even when they do not use network information. This study implies that the proper parameter setting significantly affects performance.

AuTo [7] presents the problem of control plane delay when reinforcement learning is applied for traffic optimization in data center networks. AuTo uses a different load balancing algorithm for mice flows and elephant flows to address this problem. They schedule smaller flows first using a multi-level feedback queue. Using a reinforcement learning algorithm, the parameters of the multi-level feedback queue are adjusted, and the optimal routes of elephant flows are determined. Compared to Quantized Shortest Job First (QSJF), AuTo lowers the average flow completion time by 27.95%.

Learn to Route with Deep RL [10] applies machine learning for intradomain routing. It demonstrates that traffic optimization with lower regularity is difficult to supervised learning. However, reinforcement learning shows a lower congestion ratio than traditional algorithms.

III. Design

In this section, we define the environment and agent for the RWCMP algorithm. Also, we describe the overall process of the RWCMP load balancing algorithm.

A. Environment

1) Network: When exploring the load balancing problem in the data center, we limited the network topology to the widely used fat-tree topology.

We define the network with the fat-tree topology $G = (V, E, c)$ where V, E, and C denote a node set, a link set, and mapping function, respectively. Further, n and m denote

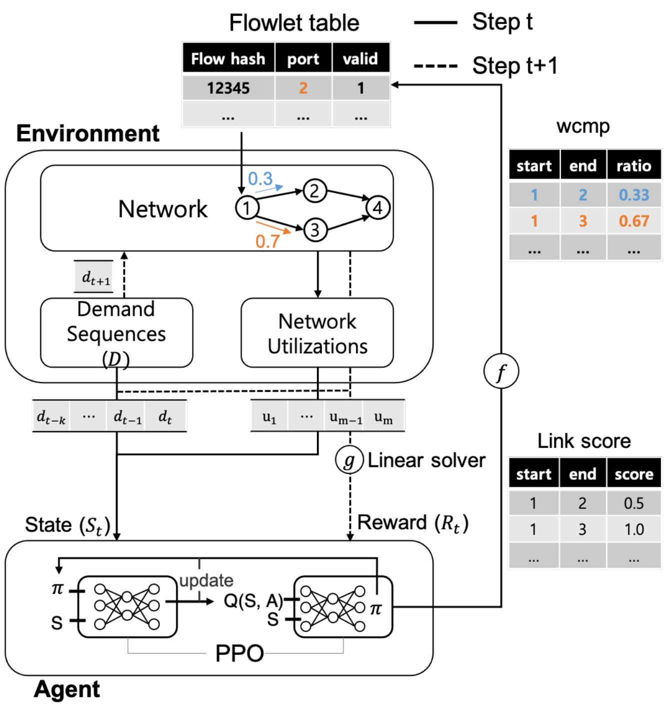


Fig. 1: Design of RWCMP

the number of nodes and links in G, respectively, while $\Gamma(v)$ denotes a neighbor node set of node $v \in V$.

2) States: States contains two factors: demand sequence and network link utilization. The demand sequence is the history of the traffic matrix. A traffic matrix constituted traffic demand sets for every node in the network. The traffic demand for each node is the outgoing flow sizes.

Network utilization is a link utilization set for every link in a given network. We normalize the traffic size and network utilization to a range [-1, 1] with the following equation.

$$2\frac{x - \min x}{\max x - \min x} - 1 \tag{1}$$

3) Reward: We define the two rewards and compare them. The first reward is defined as -1 * network utilization (Eq. 2). Network utilization is the square sum of link utilization for all link in the network. Since, cumulative reward should be maximized, we multiply the network utilization with -1.

$$\begin{aligned}\text{Network utilization} &= \sum_{(u,v)\in E} (U_(u,v))^2 \\ \text{where} \quad U(u,v) &= \frac{\sum_{i=1}^{K} f_i(u,v)\cdot d_i}{c(u,v)}\end{aligned} \tag{2}$$

The second reward can be defined as the -1 * maximum link utilization ratio (MLU_ratio) (Eq. 4). Maximum link utilization (MLU) is the maximum value of the link utilization (Eq. 3). MLU Ratio is the ratio of the MLU of the agent (U_{agent}) to the optimum MLU ($U_{optimum}$). We use the optimal MLU

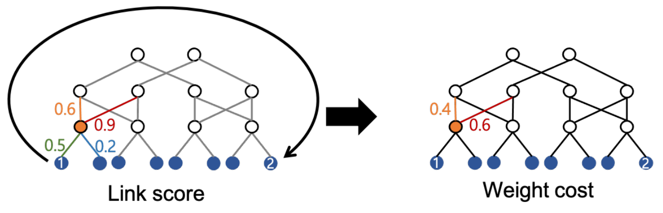


Fig. 2: Action

for giving hint the optimal routing decisions and expect faster and more accurate learning. The optimal MLU is calculated by a linear solver that computes the optimal MLU at packet-level routing using linear programming.

$$\text{MLU} = \max_{e \in E} U(e) \tag{3}$$

$$\text{reward} = -\frac{U_{agent}}{U_{optimum}} \tag{4}$$

B. Action

First, we consider action is the splitting ratios for each link. However, this requires an excessively large action space size ($O(n \times n \times m)$) where n and m indicates the number of nodes and the number of links, respectively. It makes agent difficult to learn optimal load balancing.

We set action as a selecting forwarding link at each node, and interpret traffic split ratio as a stochastic action. This method only require $O(m)$ action space size. The action probability of each link is the ratio of forwarding packets to each egress link of a given node. In this case, traffic can be reversed to the previous node, and the resulted in a routing loop problem. Thus we truncate action that can be loop-able links such as downlink at the edge or leaf node and uplinks at the core node. Suppose we should send traffic from node 1 to node2 (Fig. 2). At the orange node, we compute stochastic policy (left) and truncate the loop-able link such as green and blue links. Then we recalculate the probability of remain links, orange and red.

To process the flow with a fine-grained level, we divide the flows into flowlets. We use an aging method [11]. The aging method distinguishes new flowlet with valid bit. Periodically, the valid bit is set to 0 when there is no packet that has the same flow id in the flowlet timeout. We set flowlet timeout to 50μs.

C. Reinforcement learning algorithm

To learn the action that maximizes the total sum of rewards, we use PPO learning algorithm. PPO uses an advantage function (Eq. 5) as a loss function to reduce the variance of the policy gradient method. The advantage function $A(s, a)$ indicates a change of the expected total sum of rewards after performing the action a when in state s.

$$A^{\pi_{\theta_t}}(s,a) = Q^{\pi_{\theta_t}}(s,a) - V^{\pi_{\theta_t}}(s) \tag{5}$$

To compute the advantage, the PPO uses two models: actor model, and critic model. The actor model calculates the policy based on the action-value function. The critic model evaluates the policy that is calculated by the actor model. They receive and update each other's output values. The actor and critic model are constructed with same structure, but have different parameters that generates actions. The objective function of PPO is bound to the policy change(Eq. 6) for stabler updates. A hyperparameter ϵ controls the bound range (Eq. 7). When the policy selects action a more frequently than the previous policy at state s, advantage $A(s, a)$ is positive. The minimum function limits the increase in the objective function. Similarly, when advantage A is negative, the minimum function limits the decrease in the objective function.

$$L(s,a,\theta,\theta_{old}) = \quad \min\left(\frac{\pi_\theta(a|s)}{\pi_{\theta_{old}}(a|s)} A^{\pi_\theta}(s,a), g(\epsilon, A^{\pi_\theta}(s,a))\right) \tag{6}$$

where,

$$g(\epsilon, A^{\pi_\theta}(s,a)) = \begin{cases} (1+\epsilon)A^{\pi_\theta}(s,a) & A^{\pi_\theta}(s,a) \geq 1 \\ (1+\epsilon)A^{\pi_\theta}(s,a) & A^{\pi_\theta}(s,a)(s,a) < 1 \end{cases} \tag{7}$$

Then the PPO-clip updates its policy using the following equation:

$$\theta = \arg\max_\theta \mathbb{E}_{s,a \sim \pi_\theta}[L(s,a,\theta,\theta_{old})] \tag{8}$$

D. Process

The RWCMP algorithm iterates following process and learns the optimal routing decisions.

Let assume that the current step is t. The agent obtains demand sequences and network utilization from the environment and calculates the traffic split ratio using the PPO algorithm. Then, agent calculates the entries of the Weight-cost multipath (WCMP) routing table by truncate harmful links and applies them to the network. When the traffic demand d_{t+1} arrives at the network, each node divides a flow into flowlets, and each flowlet is forwarded to each flowlet, using the WCMP. After the process of the traffic demand, and agent obtain a reward (R_{t+1}) from the environment, which updates the parameters in the PPO algorithm.

IV. Evaluation

A. Experimental setup

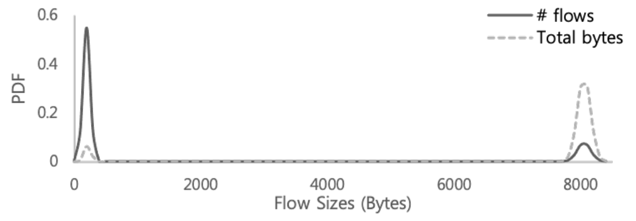
To evaluate the network utilization of RWCMP, we implemented the simulator using Python. We compare the network utilization of four load balancing algorithms for each traffic pattern in regular and link-failed fat tree topologies. Each load balancing algorithm has different function approximator and reward function. The hyperparameter settings in the experiments are listed in Table. I.

- RWCMP-mlp : an RWCMP that uses an MLP function approximator and MLU_Ratio reward function
- RWCMP-lstm : an RWCMP that uses LSTM function approximator and MLU_Ratio reward function

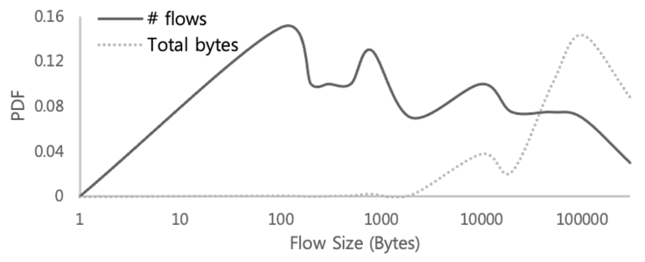
TABLE I: Hyperparameter of RWCMP

Parameter	Value	Description
History length	2	length of dm sequences
Cycle length	100	length of cycle
Sequence length	10	the number of cycle in a demand sequences
Batch size	32	the number of episodes in a single batch
iteration	100000	the maximum number of iteration
Maximum steps	10	the maximum steps in a iteration

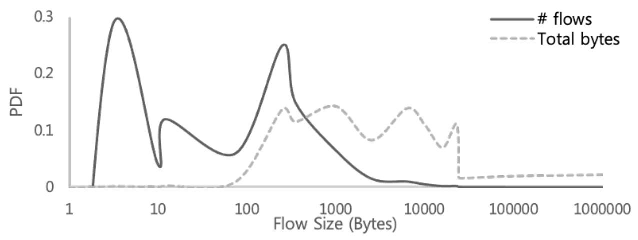
- RWCMP-lstm-nu : an RWCMP that uses LSTM function approximator and network utilization reward function
- LetFlow : a baseline algorithm



(a) Bimodal traffic pattern



(b) Web service traffic pattern



(c) Facebook traffic pattern

Fig. 3: Traffic pattern

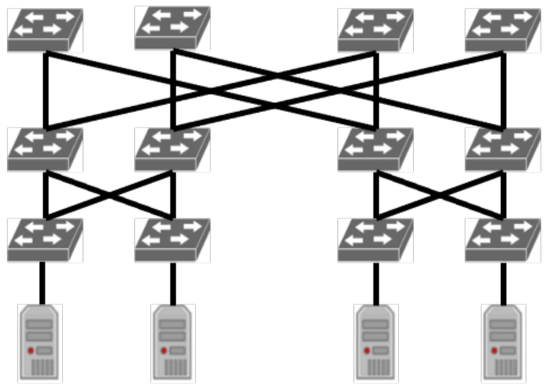
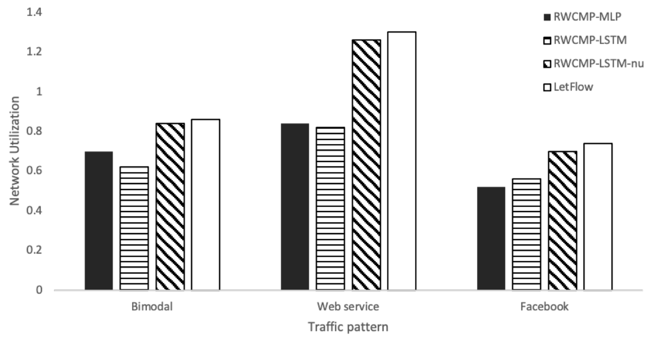
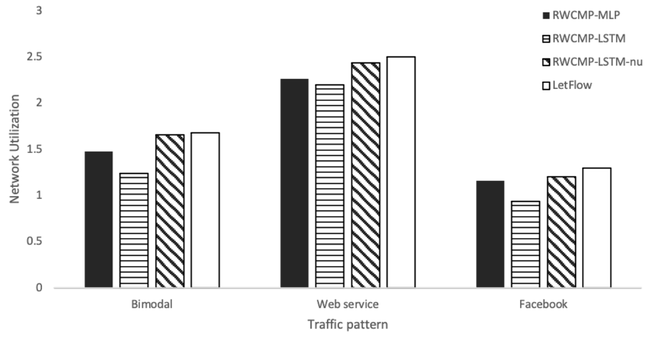


Fig. 4: Fat tree topology



(a) Fat tree topology



(b) Fat tree topology with a link failure

Fig. 5: Network utilization comparison

We simulated the RWCMP with three different traffic patterns: Bimodal pattern 3a), Web service pattern (Fig. 3b), and Facebook (Fig. 3c).

The Bimodal patterns is a traffic pattern that modelled data center network traffic with a bimodal distribution. This model comprise two normal distributions (Eq. 9) which refers to mice flows and elephant flows, respectively. The Web service pattern represents web service workloads composed of large update flows that copy fresh data to the workers and time-sensitive short message flows that update control state on the workers [12]. The Facebook pattern represents a mix of data mining workloads and web service workloads in the Facebook data centers [13].

$$d = \begin{cases} x \sim \mathcal{N}(150, 20) & \mathcal{U}(0,1) > 0.8 \\ x \sim \mathcal{N}(8000, 20) & \text{Otherwise} \end{cases} \tag{9}$$

B. Results

We generated 1000 flows in fat tree topology (Fig. 4) and the same fat tree topology with a link failure. Then, we calculated the network utilization for each load balancing algorithm in three different traffic patterns. We iterated this process 10 times and compared the average network utilization (Fig. 5).The RWCMP algorithm shows relatively lower network utilization than the traditional load balancing algorithm, regardless of traffic patterns and link failure. At the fat tree topology, the RWCMP-LSTM yields 24-36% lower network utilization than the traditional load balancing algorithm. In addition, the RWCMP-LSTM shows 28% lower average network utilization than RWCMP-LSTM-nu and 3% lower average network utilization than RWCMP-MLP. At the link failutre topology, the network utilization of RWCMP-LSTM is 12-27% lower than that of the traditional load balancing algorithm. In addition, the network utilization of RWCMP-LSTM is 17% and 10% lower than RWCMP-LSTM-nu and RWCMP-MLP, respectively.

These observations imply that the RWCMP algorithm learns a better load balancing routing strategy following the traffic patterns and topologies. In addition, using LSTM as a neural network model shows better performance than using MLP. Since LSTM has the ability to capture longer sequence information than MLP, RWCMP-LSTM learns traffic patterns more accurately than the RWCMP-MLP. Furthermore, using MLU-Ratio as a reward shows better convergence speed and lower utilization than using network utilization as a reward. This implies giving a hint to the reward accelerates the learning in the proper direction.

V. Conclusion

In this paper, we proposed a reinforcement-learning-based load balancing algorithm to set parameter automatically in data center networks. This algorithm interprets traffic split ratio as the stochastic policy, and learns stochastic policy that minimize network utilization. In addition, it uses flowlet-level scheduling granularity to avoid packet reordering problems. We implement a python-based simulator to evaluate our algorithm in three different traffic patterns with regular and link-failed topologies. We demonstrated that the RWCMP shows lower network utilization, showing better performance than the existing load balancing algorithm for processing the sequence of the demand matrix. Moreover, RWCMP using LSTM function approximator and MLU_Ratio reward function shows the lowest network utilization.

In our future work, we aim to extend the testbed to a real network to evaluate the real-time performance. To build a real network testbed, we should use programmable switches and P4 language [14] to implement the required functionalities for the RWCMP algorithm. Further, we aim to design a robust algorithm that works as the topology changes and a stable learning algorithm. We use a graph neural network as a function approximator to robust at topology changes. For stable learning, we will modify actions from the link score to link score changes. In addition, Finally, we will validate the proposed method with various communication delays and other metrics, such as throughput and flow completion time.

Acknowledgment

This work was supported in part by Institute of Information communications Technology Planning Evaluation (IITP, Development of virtual network management technology based on artificial intelligence) under Grant 2018-0-00749 and in part by the MSIT(Ministry of Science and ICT), Korea, under the ITRC(Information Technology Research Center) support program(IITP-2020-2017-0-01633) supervised by the IITP(Institute for Information communications Technology Planning Evaluation).

References

[1] C. Clos, "A study of non-blocking switching networks," *The Bell System Technical Journal*, vol. 32, no. 2, pp. 406–424, 1953.

[2] C. E. Leiserson, "Fat-trees: Universal networks for hardware-efficient supercomputing," *IEEE Trans. Comput.*, vol. 34, no. 10, p. 892–901, Oct. 1985.

[3] D. Thaler and C. Hopps, "Multipath issues in unicast and multicast next-hop selection," November 2000, rFC 2991. [Online]. Available: https://www.microsoft.com/en-us/research/publication/multipath-issues-unicast-multicast-next-hop-selection/

[4] E. Vanini, R. Pan, M. Alizadeh, P. Taheri, and T. Edsall, "Let it flow: Resilient asymmetric load balancing with flowlet switching," in *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*. Boston, MA: USENIX Association, Mar. 2017, pp. 407–420. [Online]. Available: https://www.usenix.org/conference/nsdi17/technical-sessions/presentation/vanini

[5] S. Kandula, D. Katabi, S. Sinha, and A. Berger, "Dynamic load balancing without packet reordering," *SIGCOMM Comput. Commun. Rev.*, vol. 37, no. 2, p. 51–62, Mar. 2007. [Online]. Available: https://doi.org/10.1145/1232919.1232925

[6] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat, "Hedera: Dynamic flow scheduling for data center networks," in *Proceedings of the 7th USENIX Conference on Networked Systems Design and Implementation*, ser. NSDI'10. USA: USENIX Association, 2010, p. 19.

[7] L. Chen, J. Lingys, K. Chen, and F. Liu, "Auto: Scaling deep reinforcement learning for datacenter-scale automatic traffic optimization," in *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, ser. SIGCOMM '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 191–205. [Online]. Available: https://doi.org/10.1145/3230543.3230551

[8] W. Bai, L. Chen, K. Chen, D. Han, C. Tian, and H. Wang, "Information-agnostic flow scheduling for commodity data centers," in *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*. Oakland, CA: USENIX Association, May 2015, pp. 455–468. [Online]. Available: https://www.usenix.org/conference/nsdi15/technical-sessions/presentation/bai

[9] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms." *CoRR*, vol. abs/1707.06347, 2017. [Online]. Available: http://dblp.uni-trier.de/db/journals/corr/corr1707.htmlSchulmanWDRK17

[10] A. Valadarsky, M. Schapira, D. Shahaf, and A. Tamar, "Learning to route," in *Proceedings of the 16th ACM Workshop on Hot Topics in Networks*, ser. HotNets-XVI. New York, NY, USA: Association for Computing Machinery, 2017, p. 185–191. [Online]. Available: https://doi.org/10.1145/3152434.3152441

[11] Alizadeh *et al.*, "Conga: Distributed congestion-aware load balancing for datacenters," *SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 4, p. 503–514, Aug. 2014. [Online]. Available: https://doi.org/10.1145/2740070.2626316

[12] ——, "Data center tcp (dctcp)," *SIGCOMM Comput. Commun. Rev.*, vol. 40, no. 4, p. 63–74, Aug. 2010. [Online]. Available: https://doi.org/10.1145/1851275.1851192

[13] A. Roy, H. Zeng, J. Bagga, G. Porter, and A. C. Snoeren, "Inside the social network's (datacenter) network," in *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, ser. SIGCOMM '15. New York, NY, USA: Association for Computing Machinery, 2015, p. 123–137. [Online]. Available: https://doi.org/10.1145/2785956.2787472

[14] Bosshart *et al.*, "P4: Programming protocol-independent packet processors," *SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 3, p. 87–95, Jul. 2014. [Online]. Available: https://doi.org/10.1145/2656877.2656890