ACC: Automatic ECN Tuning for High-Speed Datacenter Networks

Siyu Yan[†], Xiaoliang Wang[⋆], Xiaolong Zheng[†], Yinben Xia[†]
Derui Liu[⋆], Weishan Deng[†]
[†] Huawei Technologies Co., Ltd [⋆] Nanjing University

ABSTRACT

For the widely deployed ECN-based congestion control schemes, the marking threshold is the key to deliver high bandwidth and low latency. However, due to traffic dynamics in the high-speed production networks, it is difficult to maintain persistent performance by using the static ECN setting. To meet the operational challenge, in this paper we report the design and implementation of an automatic run-time optimization scheme, ACC, which leverages the multi-agent reinforcement learning technique to dynamically adjust the marking threshold at each switch. The proposed approach works in a distributed fashion and combines offline and online training to adapt to dynamic traffic patterns. It can be easily deployed based on the common features supported by major commodity switching chips. Both testbed experiments and large-scale simulations have shown that ACC achieves low flow completion time (FCT) for both mice flows and elephant flows at line-rate. Under heterogeneous production environments with 300 machines, compared with the well-tuned static ECN settings, ACC achieves up to 20% improvement on IOPS and 30% lower FCT for storage service. ACC has been applied in high-speed datacenter networks and significantly simplifies the network operations.

CCS CONCEPTS

• **Networks** → **Transport protocols**; **In-network processing**.

KEYWORDS

ECN, AQM, Congestion Control, Datacenter Network

ACM Reference Format:
Siyu Yan[†], Xiaoliang Wang[⋆], Xiaolong Zheng[†], Yinben Xia[†], Derui Liu[⋆], Weishan Deng[†], [†] Huawei Technologies Co., Ltd [⋆] Nanjing University . 2021. ACC: Automatic ECN Tuning for High-Speed Datacenter Networks. In *ACM SIGCOMM 2021 Conference (SIGCOMM '21), August 23–28, 2021, Virtual Event, USA*. ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3452296.3472927

Xiaoliang Wang is the corresponding author.

SIGCOMM '21, August 23–28, 2021, Virtual Event, USA

ACM ISBN 978-1-4503-8383-7/21/08…$15.00
https://doi.org/10.1145/3452296.3472927

1 INTRODUCTION

Datacenters host a variety of applications like big data processing [18, 33, 34, 41, 54, 58], distributed storage [14, 32, 42, 53], high performance computing [30, 31, 64], etc. These applications desire high bandwidth and low network latency. For example, the total throughput of each storage node can exceed 100Gbps and the access delay of NVMe SSDs is at microsecond level [27, 60]. To meet the demand of applications, on one hand the link speed of current Datacenter Networks (DCNs) has grown from 25Gbps to 100Gbps, and beyond [27, 60]. On the other hand, new techniques like Remote DMA (RDMA) [9] and advanced congestion control mechanisms have been proposed and deployed in large-scale production datacenters to reduce the delay caused by host networking stack and in-network queuing [7, 27, 29, 35, 36, 62, 65].

Since Explicit Congestion Notification (ECN) is commonly supported by commodity switches, it has been widely adopted by the state-of-the-art congestion control mechanisms in DCN, e.g., DCTCP [7], DCQCN [65], and their enhanced schemes [36, 62]. They use the standard ECN [43] and RED [20] at switch and ECN-aware rate control at the end-host to throttle the injection rate upon congestion. Despite the success of ECN-based congestion control schemes at datacenter networks, with the increase of network speed and stringent requirement of low-latency operation, the datacenter network operators face many challenges:

- Determining the appropriate device configuration is challenging. Though the publications provide formulas to determine the marking thresholds [6, 65, 66], the parameters there are highly related to the real environments [27, 29]. With regard to the large number of switches and heterogeneity network environment, it usually takes network operators weeks or months to tune the ECN setting in practice. ECN Tuning is made more difficult when large-scale datacenter consists of legacy devices and/or products belonging to different vendors.
- In multi-tenant networks, the pre-determined static marking threshold is usually conservative to handle tenants with different traffic classes. It is hard to maintain persistent high transmission rate for the bandwidth-sensitive flows by setting up small queues in switch buffers (through high ECN marking rate), because flow rate increases conservatively after congestion. On the other hand, though the built-up queue is helpful to maintain high throughput (through low ECN marking rate), it introduces high queuing delay to latency-sensitive flows. The network operators need to make a trade-off between network utilization and tenants' performance.
- The nature of multi-tenant network leads to various traffic patterns spatially and temporally. Heterogeneous computing causes periodically large volume data and burst traffic with short messages co-exist in the network. Depending on the cluster allocation

policy, the spatial dynamics could result in workload dynamics at the same location as well. We have witnessed an overall degraded performance due to the incremental deployment of applications. Workload dynamics requires parameters tuning at different timescales. Operators may potentially tune it weekly or monthly. However, it is time-consuming and error-prone to renew the ECN marking thresholds in hand-crafted fashion for thousands of running switches.

To reconcile low latency and high bandwidth, HPCC [29] leverages the precise load information from In-network telemetry (INT) to compute accurate flow rates. Timely [35] and Swift [27] adjust flow rates based on accurate delay measurements with NIC timestamps instead of ECN-based signals. These clean-slate designs demonstrate tremendous improvements. Unfortunately, the deployability is a challenge in heterogeneous datacenters consisting of legacy devices, which do not support new features, e.g. INT. Moreover, it is notable that due to operational issues for the multi-tenant production datacenters with bare-metal servers and RDMA networking, it is not easy to revise the networking stack at end-host.

From the networking operational point of view, to optimize the performance with minimum impact on tenants, we target "zero-configuration" as the objective, i.e. building up the network by using the default setting of commodity devices, while enabling automatic and fine-grained parameters tuning over volatile datacenter traffic without human intervention.

In this paper, we introduce our operation experience in a high-speed datacenter network, which realizes automatic in-network optimization to maintain low queue length without compromising network utilization at run-time. With regard to the widely-deployed ECN-based congestion control, the choice of the marking threshold is the key as it directly affects the performance of throughput and latency. We introduce, ACC[1], which applies Deep Reinforcement Learning (DRL) for automatic ECN marking threshold tuning at intermediate switches. ACC obtains information of switch loading. The DRL agent generates a policy based on observed telemetry statistics and updates the ECN parameters through the control interface of switches. ACC requires no new features of commodity devices [11–13]. Specifically, ACC enables the usage of default setting of NICs belonging to different vendors e.g., Mellanox ConnectX-4, ConnectX-5 and Intel NICs, and does not need any modification of ECN-based rate control at the end servers. Thus, ACC is easy to integrate into current datacenters.

In detail, we propose the following optimization when deploying ACC. First, due to the fact that for the scale of production datacenters, it is time/bandwidth consuming to collect the information of buffers and flow states from all switches, we study the distributed design instead of applying a centralized approach. It deploys DRL agent at each switch to adjust the ECN marking threshold independently. Second, given the large state and action space associated with ECN parameters, we simplify the operation by discretizing the DRL's state and action values. The discretion function is carefully determined based on the characteristic of traffic. Third, to optimize the exploration-efficiency of online DRL, we train the DRL model offline using samples collected from various applications and traffic patterns. Then, we adopt a fast exponential decay of the exploration probability online to avoid the unstable exploring actions [46].

The performance and stability of ACC are verified in production datacenters to support the services of distributed storage and training. ACC allows to maintain low switch queuing delay with diverse workloads. It maintains line-rate in both 25Gbps and 100Gbps networks, while improving the IOPS of storage service by 20% for a DCN with 300+ servers. It provides short RPC completion time for intensive storage and analytic workloads [27]. Through both self-defined workloads for stress tests and realistic traffic loads, our experiments show that ACC achieves up to 20% lower average FCT and 60% lower 99-percentile FCT compared with the static ECN threshold setting, while maintaining high throughput and link utilization.

In summary, this paper makes the following contributions:

- ACC aims to achieve "zero-configuration" to simplify network operation by automatic in-network optimization. The proposed approach is appealing because it does not need any modification on ECN-based protocols. ACC is easy to implement in the commodity switches. Automatic deployment with minimum maintaining cost is important not only for large-scale public cloud, but also for private cloud service maintained by small companies or academic institutes.
- ACC achieves good performance in terms of both network-level measures and applications-level metrics in realistic datacenter. ACC is able to support various applications and diverse traffic simultaneously. At the network level, ACC delivers high utilization and sustains low latency at the same time. At the application layer, ACC provides short RPC completion time for intensive storage and analytics workloads.
- Learning-based network optimization has been recently studied for Internet, Cellular network, and datacenter, etc. [19, 25, 26, 45, 56, 59]. To the best of our knowledge, this is the first work that demonstrates the experience for successfully applying DRL in ultra-high speed, datacenter-scale networks for automatic ECN tuning in practice.

2 BACKGROUND AND REQUIREMENTS

In this section, we explain the trend and difficulty when operating datacenter networks.

2.1 Background

We run both TCP and RDMA[2] protocols in our datacenter. Recently the customers have increasingly acquired for the RDMA networking support to deploy their computing and storage services, e.g. the GPU clusters running high-speed machine learning applications and baremetal servers deploying large-scale storage services. To meet customers' demand, network operators maintain a large-scale (hundreds to thousands of servers), high bandwidth (25Gbps and beyond), low latency (microseconds) RDMA network running RoCEv2 protocol [9]. DCQCN [65] is the default congestion control mechanism integrated into hardware by RDMA NIC vendors. DCQCN applies the explicit congestion notification (ECN) on switches to inform the sender to update the injection rate. ECN marking

[1] ACC is named after Adaptive Cruise Control for network operation.

[2] Remote DMA technique, allowing directly access to the memory of remote server, greatly decreases the latency of packet processing at end-host [1].

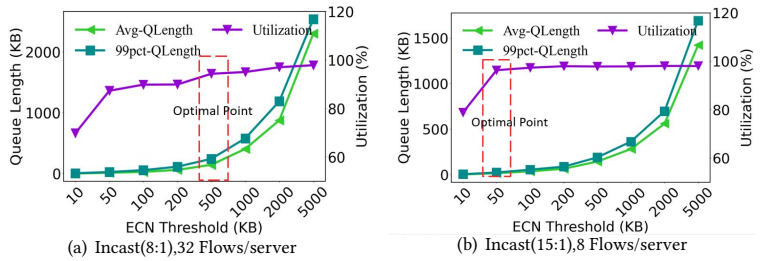


Figure 1: The optimal ECN settings under different workloads

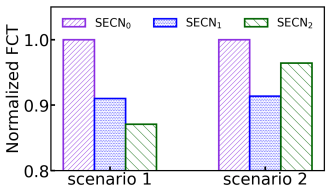


Figure 2: FCT under different DCQCN parameters:
SECN$_0$ (DCTCP): $K_{min} = K_{max} = 18KB$;
SECN$_1$ (DCQCN): $K_{min} = 5KB, K_{max} = 200KB$;
SECN$_2$ (HPCC): $K_{min} = 100KB, K_{max} = 400KB$

threshold is the key for maintaining short in-network queuing which has great impact on the latency and throughput performance of the running applications.

2.2 Motivation

When operating a large-scale RDMA network, we see multiple challenging issues to determine the ECN setting.

Observation 1: Various traffic patterns demand for different ECN settings. In large-scale production datacenter networks, the traffic model is easily affected by multiple factors, including the number of flows, load, scale of network, etc. The traffic pattern varies over time. For example, the delay sensitive Online Transaction Processing (OLTP) workload usually appears at day time. While the workload is dominated by the Online Analytical Processing (OLAP) traffic at night (and the end of each month). On the other hand, with the rise of network bandwidth in datacenter, the timescale of network events decreases accordingly, e.g. most bursts on the racks sustain for tens of microseconds [63]. The microburst traffic may result in serious performance degradation (e.g., Incast problem [44]). Through an experiment, we demonstrate that the fixed ECN parameters setting is not adaptive to satisfy the requirement of variable traffic patterns. We build a testbed with a small Clos network, which has 24 servers with 25 Gbps uplink, 2 leaf switches and 2 spine switches connected with 100 Gbps links. In the first case, we emulate the scenario of incast congestion by randomly selecting 8 servers as senders and 1 server as receiver. Each sender generates 32 flows to the receiver. In the second case, we select 15 senders, each of which generates 8 flows simultaneously. We measure the receiver's throughput and queue depth of the switch connecting to the receiver. As shown in Figure 1, in the first case, the optimal ECN threshold is $K = 500KB$ for maintaining a small queue in buffer with high throughput. For the second case, $K = 50KB$ is the optimal point with regard to the tradeoff between delay and throughput, which is much less than $K = 500KB$. We have tested other scenarios using different link loads, network scales and workloads ($WebSearch$, $DistributedStorage$), which have different optimal points to maintain high throughput and low latency (see Section 5.3).

Observation 2: Existing solutions of static ECN settings do not work well at run time. Through empirical studies on a representative ECN setting, we demonstrate and explain that the existing solutions of static ECN settings are not adaptive to the variable workloads. In the experiment, we use three kinds of ECN settings in switch, SECN$_0$ based on DCTCP paper [6], SECN$_1$ base on DCQCN paper [65] and SECN$_2$ based on HPCC [29]. We use DataMining [22] (Scenario-1) and WebSearch [7] (Scenario-2) traffic workloads in the above Clos network topology. The results are normalized according to the average FCT of SECN$_0$. As shown in Figure 2, SECN$_2$ achieves the lowest FCT in Scenario-1 but SECN$_1$ achieves the best result in Scenario-2. This is because SECN$_1$ aggressively triggers ECN notification to keep low queue length for latency sensitive traffic. It causes large variation on delay performance and long tail latency. On the contrary, SECN$_2$, using relatively high threshold, is more friendly to throughput sensitive traffic. Similar to the original DCQCN paper [65], the conservative setting of SECN$_0$ is not the optimal setting. When using the single threshold, i.e., $K_{max} = K_{min} = C \times RTT \times \lambda$, where RTT denotes the average round-trip time, C denotes link capacity, and λ is a parameter related to realistic network environment, we see the following problems: 1) it is hard to estimate the value of λ. Specifically, device vendors may have customized rate control in the hardware implementation. 2) the value of RTT is not stable in realistic multi-hop network with middleboxs. ECN# [62] shows that the actual RTT varies vastly in production environment (around 3×). It is hard to find an appropriate static threshold to balance queue occupancy and throughput.

Observation 3: Parameter tuning is complicated and time consuming. Operators always struggle to determine the appropriate value for CC parameters. For example, DCQCN mainly has 9 parameters at end-host and 3 ECN parameters at switch. DCTCP has 5 parameters, HPCC has 3 parameters and ECN# has 4 parameters, etc. It usually takes a few weeks or months to evaluate the performance of throughput and latency under the given sets of traffic demands, and perform stress testing to emulate all traffic patterns, like many to one traffic demands, various flow sizes, and different scheduling approaches, failure scenarios, etc. The experienced operators need to have a deep understanding of the requirements of applications and network behaviors at high-resolution. Additionally, production networks usually consist of devices from different NIC and switch vendors. For example, our servers are configured with NICs from two major NIC vendors. Hence, we need to tune the threshold for each port of the leaf switch. To ensure the stability, the operators usually adopt a conservative version of static ECN parameters. At run time, to guarantee the performance of low

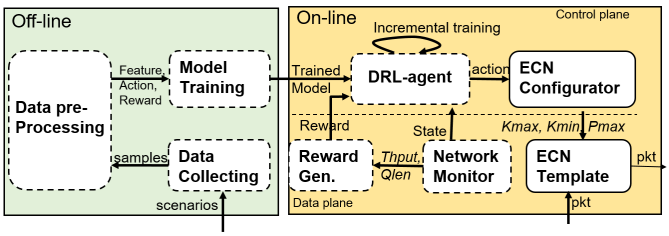


Figure 3: Overview of ACC: applying RL for the setting of ECN marking thresholds

latency traffic, the operators have to keep low network utilization and deploy new applications separately in new PoDs [29].

3 ACC

We aim to address the above issues with an effective "zero-configuration" approach. The proposed approach should be easy to deploy in production datacenters, be able to operate alongside existing equipments, require no modification to the existing network stack, and maintain good performance under diverse traffic loads and network scales. Learning-based approaches have great potential to dynamically adjust the ECN marking threshold. They avoid the costly procedure of manual tuning of parameters for a specific network environment. Thus, we propose a learning-based ECN tuning based on the operation experience of datacenter networks.

3.1 Overview

Reinforcement Learning. Tuning ECN dynamically can be formulated as an Reinforcement learning problem (RL). RL[46] is a learning setting that an agent learns from the interactions with the environment. Markov Decision Process is usually used as the mathematical formalization for reinforcement learning.

Definition 1 (Markov Decision Process) A Markov decision process (MDP) is a 4-tuple $\mathcal{M} =< \mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{P}, \gamma >$, where $\mathcal{S}$ is a state space, $\mathcal{A}$ is a set of actions, $\mathcal{P}(S_{t+1}|S_t, a)$ is the probability of action $a \in \mathcal{A}$ at state S_t will lead to state S_{t+1}, $\mathcal{R}(S, a)$ is the intermediate reward after executing action a from state S_t, and $\gamma \in [0, 1]$ is the discount factor that controls how much we favor immediate rewards over those from distant future.

The goal of reinforcement learning is to learn an optimal policy π^* that maps from states to actions and maximizes the discounted accumulated rewards $\pi^* = argmax_\pi E^\pi\{\sum_{t=0}^{\infty} \gamma^t r_t\}$. This goal is equivalent to finding the optimal state-action value $Q^*(S, a) = \max_\pi E^\pi\{\sum_{k=0}^{\infty} \gamma^k r_{k+1}|S_t = S, a_t = a\}$, which defines the value of taking an action a at state S.

The problem of automatically tuning ECN is formalized as an MDP. Specifically, we divide time into consecutive monitoring interval (Δt). At time slot t, the agent records the state (network statistics) S_t, takes an action (ECN configuration) a_t and receives a reward r_t.

ACC Framework. Figure 3 depicts our general framework for applying RL to set ECN marking thresholds. To speed up the training, we apply both offline training and online training. The offline training is performed to get the pre-trained model based on the collected traffic trace. During the online training, the agent interacts with the environment and adapts to optimize the action. In detail, the RL-agent is based on the pre-trained Neutral Network (NN) model. The network monitor collects data from switches and feeds the data to RL-agent and reward generator. The measurement includes queue depth, throughput and flow information (e.g. number of ECN-marked packets). Then the RL-agent makes an action to ECN configurator, which maps the ECN value into ECN template of forwarding chip. Meanwhile, RL-agent updates the NN model with the reward.

3.2 Design Choice

Centralized Design. The ideal ECN configuration should be able to provide high throughput and low latency for different applications. Besides, it is adaptive to the variable traffic patterns and different network scale. Intuitively, we can apply a centralized DRL agent like a centralized controller to implement the automatic ECN tuning. The topology information could be collected via the link layer discovery protocol and the network state could be collected via the control interface at each switch. All the collected data are transferred to a central controller. Unfortunately, when applying the centralized DRL-based automatic ECN tuning in the high-speed DCN, we found the following practical problems:

- **The large space of network states and actions.** In the centralized design, the agent has the information of topology and states of all switches, including throughput, queue length, number of packets transferred, PFC frames received, packets with ECN marking, etc. For example, a large-scale DCN with $1K$ switches and 48 ports in each switch. Assume two queues of each port are assigned for RDMA traffic which apply automatic ECN tuning. Thus the vector of network states consists of at least $1K \times 48 \times 2 \times NF$ elements, where NF is the number of collected features. Assume that each feature has 10 values. The state space is $10^{96K \times NF}$. On the other hand, the ECN configuration has 3 parameters $\{K_{max}, K_{min}, P_{max}\}$. The marking threshold varies from a few KB to tens of MB. Assume each parameter has 10 intervals, the action space is as large as $\{10 \times 10 \times 10\}^{96K}$. It is hard to make accurate decisions online for such large network states-actions space. Though the topology of datacenter is symmetric, the traffic is not equally distributed due to the spatial and temporal variation of tenants' traffic. With regard to the large amount of states and actions, the centralized model will take long time to converge and acquires large computation resources.
- **The long latency for collecting network state and updating ECN configuration.** In modern high-speed datacenters, the round-trip latency is about ~10 microseconds. Therefore, the update of ECN configuration should be in a few microseconds to minimize the influence of congestion on application performance. In the centralized design, though it can tolerate several milliseconds to update DRL model during online training phase, it has only a few microseconds to make decisions during DRL inference phase. In practice, the centralized controller takes several milliseconds to collect data from all switches, model inference and set actions to all switches. Therefore, the centralized design would suffer from the long delay to deal with network congestion.

- **The large bandwidth consumption to collect data.** If the ECN tuning period is 100 μs, and we only collect 4 features per queue (4 bytes per feature) and transfer data to a decision node by using UDP packet (header consumption is 46B), the total amount of data to be collected is $1K \times 48 \times 2 \times (4 \times 4B + 46B) = 5.952MB$. Hence, the bandwidth consumption for collecting data is $5.952MB/100\mu s = 476Gbps$, which is a huge overhead for the system. Besides, it is a hazard to introduce a large amount of in-band monitoring traffic in the datacenter network.

Distributed Design. To overcome the aforementioned issues, we introduce a distributed design to achieve dynamic ECN tuning at runtime. Each switch is associated with a DRL agent. These agents form a multi-agent system. Each DRL agent observes the local queuing states and chooses an action for ECN setting independently based on the reward function. We explain and perform a simulation-based study to demonstrate the effectiveness of distributed design compared with the centralized design (see Section 5.4).

- Distributed design takes only the local network states and makes decision to dramatically reduce the state-action space. Since fewer features are used and fewer decisions are made in the agent, the state-action space is much smaller compared to the centralized design. As a result, the convergence of the learning process is fast [37]. To further accelerate the convergence, we carefully select several key features to represent the network state when designing the state of the agent. (Section 3.3)
- Distributed design reduces transmission latency by avoiding inter-device communication. The ECN tuning process is completed within the switch, and takes only a few microseconds, which is at the time scale of RTT in the datacenter. DCQCN reacts on timescale of microseconds. Its control loop interacts with ACC well to get enough time to become stable. (Section 3.3)
- Distributed design avoids inter-device data transmission. Each local agent make decisions independently. The state information is transferred within the switch via the inter-chip connection lane. (Section 4)

3.3 Problem Formalization

A learning agent is associated with each switch. The agent collects local port's network information, and then makes and executes an ECN configuration decision. We apply Deep Q-learning (DQN) [52] model to design the DRL agent.

State: States represent the environment information that is applied as the input of an agent, i.e., the network congestion risk. Here, we represent states as collectible statistics which can be measured on the fly from each switch. To be compatible with the cloud datacenters which usually contain switches from different vendors, we choose the features commonly supported by major switch chip vendors [49–51] as candidates. Based on the operational experience, as a result, we consider four features, i.e. current queuing length ($qlen$), output data rate for each link ($txRate$), output rate of ECN marked packets for each link ($txRate^{(m)}$) and current ECN setting($ECN^{(c)}$). Instead of feeding the values of gathered statistics to the agent directly, we use normalized value since normalization helps the agent generalize different network environments. Specifically, to evaluate the variations of queue length and throughput in continuous time slots, we apply the queue state of past k monitoring slots as the state information for each tuning inference.

Action: The action at time slot t is defined as the ECN setting, i.e. high marking threshold (K_{max}), low marking threshold (K_{min}), and marking probability (P_{max}).

$$a_t = \{K_{max}, K_{min}, P_{max}\}_t$$

To reduce action space, we discretize ECN tuning action space to form the ECN configuration template at switch. We choose discretization for the ECN marking threshold. We have tested several settings with fine-granularity level. It demonstrates that the throughput is not sensitive to the high marking threshold when it is larger than $1MB$. Therefore, we choose coarse-grained settings here to minimize the action space, e.g. $\{1MB, 2MB, 5MB, 10MB\}$ since the maximal buffer size of each queue in the commodity switch chip is usually less than $10MB$. For the low marking threshold, setting multiple intervals in a short range is helpful to achieve fine-grained adjustment on marking packets during congestion. To formalize this characteristics, an intuitive way is to use the exponential function. We introduce an exponential function (1) to determine the discrete action value $E(n)$

$$E(n) = \alpha \times 2^n KB, n = 0, .., 9 \quad (1)$$

where α is 20 in our system, and K_{min} is no greater than K_{max}.

For the marking probability P_{max}, when the P_{max} adjustment interval is more than 5%, the network throughput or delay have more than 1% change. Thus the uniform discretization is recommended, i.e. $P_{max} \in \{1\%, j \times 5\%\}, \forall j \in [1, 20]$.

ACC collects statistic data, makes and executes an ECN configuration decision at each time interval Δt. If ACC's adjustment period Δt is at the same scale with the reaction time of congestion control, ACC can seriously affect the performance of the existing congestion control scheme. For example, DCQCN takes a few RTTs in the control loop to respond to the congestion signal. To avoid interference with CC rate control, we choose one order of magnitude more than RTT as the action period to adjust ECN marking thresholds.

Reward: Reward function indicates the effectiveness of the action. From the network operators' point of view, we usually select the key network performance metrics of latency and throughput as the reward function. However, for the distributed design, we can only achieve the local information from switch. To ensure high link utilization and low queue buildup, we define the reward function as follows. The average throughput of one egress queue is represented by $txRate$, i.e., the amount of data that have been delivered to the link during the time interval Δt. We normalize the $txRate$ by the link bandwidth BW to represent the link utilization. The latency is represented by the average queue length L to indicate the impact of queuing delay. We select the average value instead of the instantaneous queue depth because the instant queue length varies in a large range, which can make reward unstable. Consequently, we define the reward function as a trade-off between latency and throughput, i.e. a trade-off between high link utilization and low queue buildup for each switch:

$$r = \omega_1 \times T(R) + \omega_2 \times D(L) \quad (2)$$

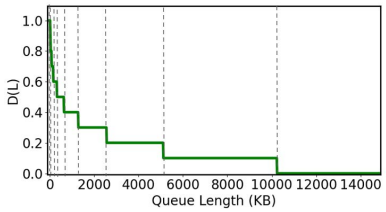


Figure 4: Mapping function of queue length reward

where $T(R) = txRate/BW$ denotes the utilization of link. Considering that applications are more sensitive to latency, we design $D(L) = 1 - n/10$, where $n = argmin_n(E(n) \geq L), \forall n \in [0, 9]$. L denotes the queue length. Here, $D(L)$ is a step mapping function as shown in Figure 4. The lower queue length the better. We provide an in-depth rational analysis of reward design in Appendix .1. ω_1, ω_2 are the weights to representing the utility-delay tradeoff, $\omega_1+\omega_2 = 1$. The network operator can easily set the reward parameters based on the requirement of running applications. For example, ω_1=0.7 and ω_2=0.3 are recommended in our storage system.

Markov Property: One requirement of the MDP is that a state is able to summarise past sensations compactly so that all relevant information is retained [46]. Here, we follow the assumption that the network statistics from the past k monitoring intervals is sufficient to summarise the variation of network statistics. k is usually determined via experimental results. More specifically, we have trained the model with different historical periods of network states (k=1, 3, 5) and evaluated the performance. $k = 3$ suffices to represent network congestion while avoiding introduce large state space. Hence, we use $4 \times 3 = 12$ features in total to represent the state of DRL agent.

3.4 DRL Algorithm:

We show that the optimization of ECN can be formulated as a DRL problem. Deep Q-network (DQN) [38] is a basic DRL algorithm, where $Q(S_j, a, \theta_j)$ is represented using a deep neural network with parameter θ_j (the evaluation network). The evaluation network is updated at the iteration j using the following loss function:

$$L_j(\theta_j) = E[(y_j - Q(S, a; \theta_j)]^2 \tag{6}$$

where $y_j = r_j + \gamma max_{a'} Q(S_{j+1}, a, \theta')$, θ' denotes the parameter of a separate *target network*. The target network is obtained by saving the evaluation network every n step (periodically updated target). The intuition is that if we know the optimal action-value of the next state S_{j+1} , then the best strategy is to push the current action-value function $Q(S_j, a, \theta_j)$ closer to the sum of the immediate reward and the discounted action-value of the next state ($Q(S_{j+1}, a, \theta')$.

At time step j, we perform an action a_j, observe reward r_j and next state S_{j+1}. The tuple $(S_j, a_j, r_j, S_j + 1)$ is called an experience. This experience is stored to a buffer D for experience replay. The network is then trained by sampling from D uniformly.

Together, *periodically updated target* and *experience replay* have been proved to greatly improve and stabilize the training procedure of Q-learning [38]. However, DQN is known to overestimate action values under certain conditions. Deep Double D-network (DDQN) [52] is able to reduce overestimation by decomposing the max operation in the target into action selection and action evaluation.

Algorithm 1 ACC's Learning Algorithm

Input: Local Replay Memory D, Batch size N

1: **for** t=1 to T **do**
2: // for each interval Δt
3: Agent pulls a queue state QS_t, and obtains the state
$$S_t = \{QS_{t-k+1}, QS_{t-k+2}, ..., QS_t\}$$
4: Select the action $a_t = \arg\max_a Q(S_t, a, \theta_i)$ and execute the action a_t
5: At the following time step $t + 1$, observe the queue state QS_{t+1}, monitor the throughput and queue length to obtain the reward r_t
6: Store the transition $\{S_t,a_t,r_t,S_{t+1}\}$ in D
7: Sample random minibatch of transitions $\{S_j,a_j,r_j,S_{j+1}\}$ from D and compute the loss:
$$y_j = r_j + \gamma \times Q(S_{j+1}, \arg\max_a Q(S_{j+1}, a, \theta); \theta') \tag{3}$$
$$L(\theta) = \frac{1}{N} \sum \{y_j - Q(S_j, a_j; \theta)\}^2 \tag{4}$$
8: Compute the gradient and update the evaluation network parameters θ with the gradient:
$$\nabla_\theta L(\theta) = \frac{1}{N} \sum \{y_j - Q(S_j, a_j; \theta)\} \nabla_\theta Q(S_j, a_j; \theta) \tag{5}$$
9: Replace the target network θ' with the evaluation network θ every n iterations
10: **end for**

So we use DDQN as agent, where the target y_j in equation (6) is replaced by equation (3).

Furthermore, in order to achieve stability of DDQN in multi-agent setting, we follow in principle the design of asynchronous deep reinforcement learning [37]. More specifically, we maintain a global replay memory beside local replay memory at each agent (switch). The experience tuples of local memory will be periodically sampled and added to the global memory, while some experience tuples from the global memory will also be periodically sampled to the local memory. The use of a global replay memory is a centralized way to store the history of agents' experiences with larger capacity. The periodic exchange interval can be several seconds. By doing so, agents at different switches can exchange experiences and explore different parts of the whole network environment. This strategy has been proved to make the learned model more stable and generalizable [4]. The training algorithm of each agent is given in Algorithm 1.

4 IMPLEMENTATION

In this section, we describe the implementation of ACC on commodity switch. To reduce CPU overhead we introduce two steps of optimization: (i) combining offline and online training to accelerate the convergence of DRL model; and (ii) applying parallel message processing to speed up state monitoring.

4.1 Hardware Architecture of ACC

Figure 5 shows ACC implementation on a commodity switch. The switch's SDK (Software Development Kit) provides basic telemetry

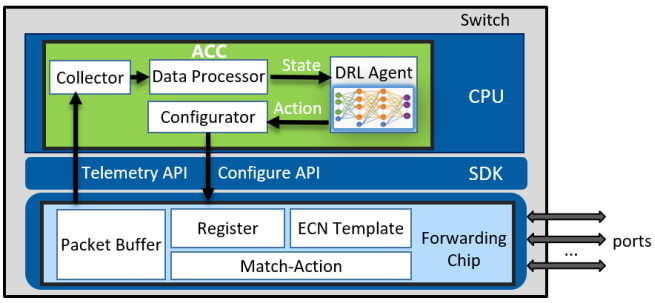


Figure 5: Hardware architecture of ACC in switch

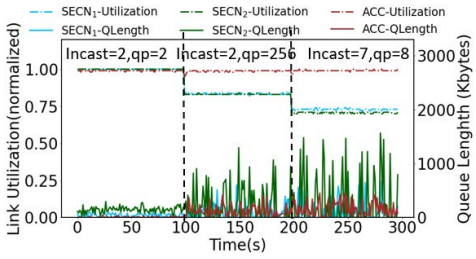


Figure 6: ACC dynamically adjusts ECN threshold to optimize queue size and link utilization

interface to read the state information from the switch chip and configuration interface to set up ECN parameters. We implemented ACC as a module in commodity switches (or a container in the programmable switches), which takes ~2400 lines of C code. The offline pre-trained DRL's NN model is loaded into the DRL agent module. The collector subscribes raw data from switch chip for features analysis, including the total bytes sent, number of ECN-marked packets and egress queue depth.

In detail, at each time interval Δt, the collector achieves the subscribed data from forwarding chips's registers. Then, data processor normalizes the row data as reward, extracts features

$$QS_t = (qlen, txRate, txRate^{(m)}, ECN^{(c)})$$

Then, QS_t is stored in memory as current states. Data processor obtains the history data (QS_{t-2}, QS_{t-1}) from memory and sends the state information S_t=$(QS_{t-2}, QS_{t-1}, QS_t)$ to DRL agent. DRL agent uses S_t as input to make inference and updates the DRL model. The new action a_t is generated and put to configurator. Finally, configurator maps the action into ECN template and sets new ECN marking threshold to the forwarding chip. Then it obtains the reward r_t and observes the next state S_{t+1}. For each time interval, the new transition $\{S_t, a_t, r_t, S_{t+1}\}$ is stored into replay memory for online training.

4.2 Optimization for Data Processing

ACC applies multithreading for parallel monitoring and data processing, and realizes thread granularity optimization for NN inference. One thread is responsible for monitoring a set of the outgoing port's queues in a round-robin manner. Given the large number of ports, especially when virtual queues are used, it leads to long monitoring period which can be longer than Δt and causes high CPU overhead. To optimize the process, we classify the queues into two categories: the busy queue and idle queue. If the queue length is less than K_{min} or the corresponding reward function does not change for continuous three time slots, we set it as "idle" queue and we can stop the inference task for the idle queue. Once the queue length of a idle queue is larger than K_{min}, this queue is identified as "busy" queue and we starts the inference on this queue. By so doing, we have observed ~10% decrease of CPU consumption.

4.3 Optimization for Training

In order to decrease the risk trial and error of online learning, we apply both offline training and online training. To guarantee the generalization of the offline training, we adopted a variety of typical traffic patterns generated by Perftest tools [40], including incast traffic and realistic traffic trace. The incast traffic is generated by randomly selecting $p \in [2, 64]$ senders forwarding traffic to one receiver. Each server randomly generates $q \in [1, 1000]$ flows with message sizes ranging from 10KB to 10MB. The traffic load varies from 10% to 90%. The realistic traffic traces are collected from prevailing RDMA applications, including distributed storage, high performance computing with LinkPack[16], Quantum Espresso[48], and distributed training with Tensorflow[34], Horovod[24]. After a model is trained offline by these training samples, we install the same offline training model for network switches. The switch will train its own local model online by using the realistic traffic to improve the model generalization. During the online training, the probability of choosing the exploration action is exponential decayed and the actions resulting large reward will be prioritised.

5 EVALUATION

We evaluate the performance of ACC based on the controlled experiments. We apply testbed experiments and large-scale simulation to validate the effectiveness of ACC in comparison with the alternative solutions.

5.1 Testbed Setup

Network Topology: The testbed is a two-layer Clos network, which mimics a small scale PoD (point-of-delivery) in the production datacenter. The testbed consists of two spine switches, four leaf switches and 24 servers. Each server is configured with two Mellanox ConnectX-5 cards (25Gbps) and connects to two leaf switches for high availability. Each leaf switch connects to spine switches via four 100Gbps links. The average RTT is under 2.32μs for inner-rack, and 6.13μs for inter-rack. PFC [3] is enabled at NIC and switches. Based on the NIC vendor's default setting of PFC, $X_{off} = \alpha(1+\alpha) \times Buffer_{free}$, where $\alpha = 1/8$, i.e., PFC is triggered when an ingress queue consumes more than 11.11% of the free buffer. ACC is deployed on both leaf and spine switches.

Workload: For micro-benchmark, RDMA incast traffic (N-to-1) is generated by using Mellanox PerfTest tool [40], which is widely used for performance evaluation in RDMA network. We change the number of QPs and senders to generate dynamic traffic. For macro-benchmark testbed, we use FIO benchmark tools [15] to generate traffic of distributed storage applications and use Tensorflow [34]

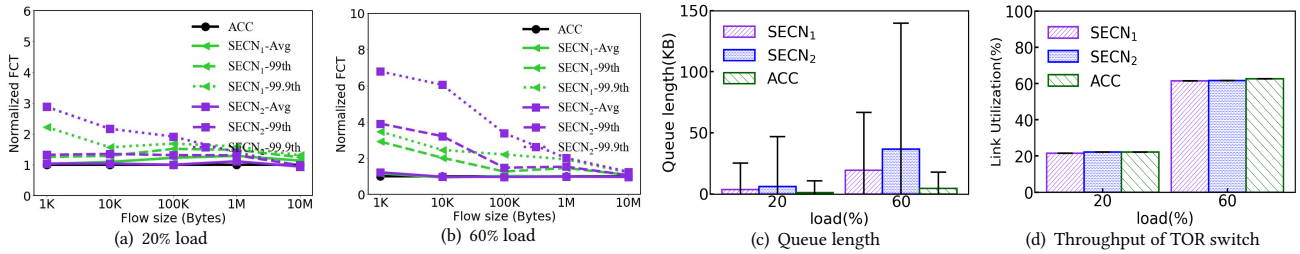


Figure 7: FCT, average queue length and switch port utilization of ACC and SECN during Incast

to generate traffic of distributed training for deep learning models. Furthermore, TCP traffic is generated by Iperf [47].

Benchmark: We compare ACC with two static ECN settings ($SECN_1$ and $SECN_2$). $SECN_1$ refers to the DCQCN paper [65], where K_{min} = 5KB and K_{max} = 200KB in our testbed. $SECN_2$ refers to the setting of a major cloud provider [29]. The ECN marking threshold is proportional to the link bandwidth (BW). $K_{min} = 100KB \times \frac{Bw}{25Gbps}$ and $K_{max} = 400KB \times \frac{Bw}{25Gbps}$. For example, K_{min} is 100KB and K_{max} is 400KB when the link bandwidth BW is 25Gbps. It is notable that the cloud provider version is usually conservative to avoid triggering PFC pauses during Incast event.

5.2 Micro-benchmark

We seek to understand: (1) can ACC adapt to the variable traffic? (2) how is the end-to-end performance of ACC? (3) Does ACC maintain fairness between RDMA and TCP traffic?

Heterogeneous Traffic. We randomly change the number of flows and the number of Incast senders every 100 seconds. We train ACC for different Incast and flows traffic for 4 hours and use the trained ACC model to adjust ECN thresholds online. The queue length and utilization of links are plotted against time in Figure 6. For the fixed ECN parameters ($SECN_1$, $SECN_2$), when the traffic characteristics match the ECN setting, they have good throughput and queue size performance as ACC (see the time interval of 0~100s). But when the missmatch occurs, the throughput drops or the queue builds up quickly (see the time intervals of 100s~200s and 200s~300s). This result shows that the fixed ECN parameter setting does not adapt to dynamic traffic. ACC learns and adapts across time varying traffic characteristics to reduce an order of magnitude of queue length and achieves 26.1% improvement of the average throughput.

End-to-end Performance. To evaluate the end-to-end performance, we keep on randomly sending messages of size {1KB, 10KB, 100KB, 1MB, 10MB} from two senders to one receiver at 20% and 60% loads. We evaluate the average FCT and the 99th/99.9th percentile FCT to represent the average end-to-end latency and the tail latency of flows, respectively. For ease of comparison, the results are normalized based on the FCT achieved by ACC. As shown in Figure 7 (a) and 7 (b), ACC achieves lower FCT compared to static ECN at different loads especially for small flows. For example, ACC cuts the tail latency of SECN by 1.5× ~ 3× for the mice flows with messages shorter than 10KB. The round trip latency of such messages is close to the RTT in the testbed. The gap increases with the increase of traffic loads. For example, at 60% load, ACC cuts the tail latency of SECN by 2× ~ 7× for the mice flows.

We then evaluate the link utilization and queue length of one leaf switch connecting to the receiver (Figure 7(c) and 7(d)), which provides more insight into the achieved performance gain. At 60% load, the average queue size of ACC is 5.6KB and standard-deviation is 13.3KB, whereas the average queue size and standard-deviation of $SECN_1$ are 20.4KB and 49.6KB, and for $SECN_2$ they are 37.5KB and 109.3KB. ACC achieves much lower queue size steadily, thus achieves much lower tail FCTs for short flows. These experiments confirm that the statically-configured ECN is not adaptive to the mixed traffic flows at different loads, which results in high queue length. While ACC can learn to approach the optimal ECN threshold and adapt to different traffic patterns.

Fairness between RDMA and TCP Traffic. Coexistence of TCP and RDMA traffic is common in datacenter. Generally, RDMA and TCP traffic are isolated from each other by using different traffic classes [1], but they usually use the shared physical buffer of the switch. However, when operating the network, we notice that TCP is not RDMA friendly in some scenarios, which is not well addressed in the literature. TCP and RDMA uses different transport protocols. Once the congestion occurs, it takes longer time for TCP traffic to decrease rate than RDMA traffic because of the long TCP RTT feedback interval (TCP 25.4us v.s. RDMA 1.7us [65]). In this case, TCP traffic will occupy more buffer and bandwidth than expected. Specifically, in some datacenters, the drop-tail mechanism is used for TCP, it becomes more greedy and may occupy the whole buffer of the port. A straightforward solution to this problem is applying a high ECN marking threshold, which leads to low ECN marking rate and maintains a balance between RDMA traffic rate and TCP traffic rate. However, by doing so the queue will build up which increases the latency of RDMA traffic. We use an experiment to demonstrate this problem. To evaluate the bandwidth sharing between RDMA and TCP, we set up 8 servers with 100Gbps RDMA NIC connected by one switch. At the switch, the bandwidth allocation, 70% for RDMA traffic, 30% for TCP traffic, is configured through the deficit weighted round robin [1]. We select 2 or 7 servers sending messages to one receiver. The concurrent RDMA queue pair connections randomly change from 1 to 32 at each sender. We use ECN setting introduced in DCQCN paper [65] as the static setting ECN (SECN).

As shown in Figure 8, with static ECN TCP occupies 10% to 20% more bandwidth than allocated bandwidth, i.e. the actual throughput of RDMA is 10% to 20% lower than expected. In contrast, ACC

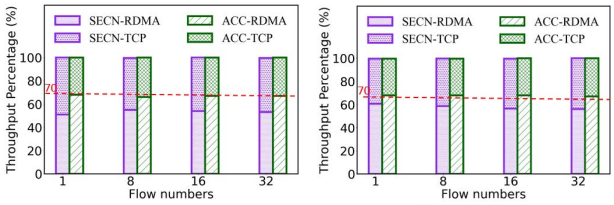


Figure 8: The average throughput ratio of RDMA and TCP traffic over 100Gbps network

can significantly improve the fairness for weighted fair sharing between TCP and RDMA traffic. This is because ACC can iteratively adjust ECN marking threshold to maintain high throughput of RDMA traffic for making full use of the allocated bandwidth resources. This result also works for network with 25G links. Specifically, ACC achieves up to 65.1% (average RTT) and 25.3% (99th RTT) lower latency than SECN, i.e. ACC can achieve the low latency while maintaining high throughput in coexistence with TCP traffic.

5.3 Macro-benchmark

To evaluate the performance of ACC under realistic application, we test ACC under distributed SSD-storage system and distributed GPU training system.

5.3.1 Distributed storage system. In the cluster of distributed storage, servers are divided into computing nodes and storage nodes, which are deployed in a ratio of 3:1. The macro-benchmark consists of 18 servers as computing nodes and 6 servers as storage nodes. Computing nodes send IO requests (Read Data/Write Data) to storage nodes, while storage nodes backup data, and response to the request of computing nodes. We measure the performance based on the IOPS (I/O operations per second) metric, which is affected by both network throughput and latency. We compare ACC with fixed ECN setting, $K_{max} = 270KB, K_{min} = 30KB, p = 10\%$, which are suggested by the device vendor.

Table 1: Traffic loads in distributed storage system[3]

Traffic Pattern	Read-Write Ratio	Blocksize(IO size)
OLTP	5:5	512B-64KB
OLAP	5:5	256KB-4MB
VDI	2:8	1KB-64KB
Exchange Server	6:4	32KB-512KB
Video Streaming	2:8	64KB
File Backup	4:6	16KB-64KB

Traffic Loads. We use FIO [15] to produce realistic workloads of distributed storage (Table 1), which is monitored and abstracted from the trace of large scale cloud storage system for the last five years. The traffic models are identified based on the following characteristics: read-write ratio, block size distribution, IOdepth concurrency[4]. For example, OLTP includes fast transaction query of data less than 64 KB. OLAP includes complex data analysis, the blocksize of which varies from hundreds of kilobytes to several megabytes.

Summary of Results. As illustrated in Figure 9, ACC improves by up to 30% application performance of the distributed storage cluster. Take *VDI* as an example (Figure 9(c)). ACC achieves better IOPS performance especially for large IOdepth. For IOdepth of 16, ACC increases 5% IOPS. For IOdepth of 128, ACC increases 15.3% IOPS. The gap between SECN and ACC increases with the increase of IOdepth. For the *FileBackup* (Figure 9(f)), the improvement of IOPS for ACC is as large as 30% compared to the static ECN setting.

We notice that there are some cases where ACC achieves little performance gain in comparison with SECN, e.g. the *OLTP*, *VDI*, and *FileBackup* with low IO depth and small IO size. We observe almost no ECN-marked packets. It happens due to the low probability of collision in switch when senders have low concurrency and flow rate. We observe the performance gain with the increase of IO depth. With more IO depth and tasks, the storage system becomes overload and leads to degrade performance (Figure 9(b) and (d)).

5.3.2 Distributed training system. To illustrate the efficiency of ACC in GPU clusters, we use 8 servers with GPU P100 (7 servers as workers and 1 server as parameter server) to train AlexNet and ResNet-50 models (batchSize=64), respectively. Meanwhile we use the training speed as metric to evaluate the performance of ACC. As shown in Figure 10(a), ACC outperforms $SECN_1$ and $SECN_2$ for distributed training. For example in ResNet-50, ACC achieves up to 7% and 12% higher training speed. Besides, Figure 10(b) shows that ACC achieves the low round trip latency, which benefits to the small messages, such as control packets. Note that ACC also improves the link utilization for better communication of big messages.

In distributed training, the traffic patterns of communication repeatedly occurs in each training iteration. ACC's NN has the experience memory. In the following iterations it can conserve quickly to adjust the ECN when traffic with similar patterns arrive and achieve good performance.

5.4 Large Scale Simulation

In this section, we use NS3 [2] simulation to evaluate ACC's performance in large-scale DCNs.

Setup. We use a 288-host leaf-spine topology with 12 leaf switches and 6 spine switches. Each leaf switch has 24 25Gbps links connecting to the servers and 6 100Gbps links connecting to the spine switches. We generate traffic based on two realistic workloads, as shown in Figure 11: Web Search [7] and Data Mining [22]. Both workloads are heavy tailed.

Overall. The average FCT of all flows is calculated, which is expressed as overall average FCT. As shown in Figure 12(a), compared to $SECN_1$, ACC achieves 5.8% lower overall average FCT at 90% load. Compared to $SECN_2$, ACC achieves 16.6% lower overall average FCT at 90% load. This is because ACC can maintain high throughput

[3]OLTP: Online transaction processing, OLAP: Online Analytical Processing, VDI: Virtual desktop infrastructure, Exchange server: Mail reading/writing processing, Video Streaming: Upload and download of videos, File Backup: Large file backup

[4]Read traffic is mainly from storage nodes to computing nodes and write traffic is in the reverse path. Block size indicates the size of the data read or write by one IO request (equal to the flow size). IO depth means the number of outstanding I/O requests

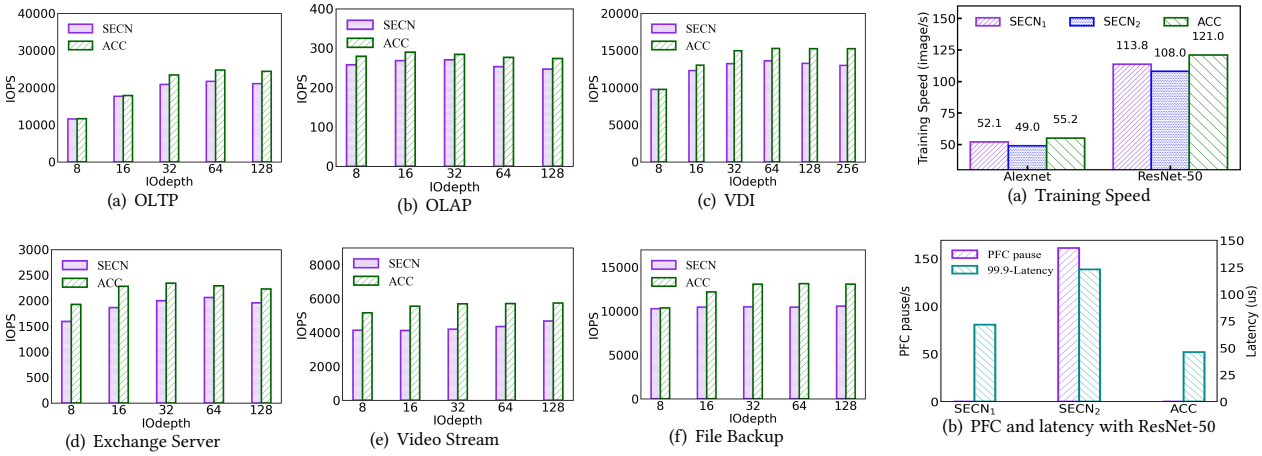


Figure 9: Comparisons in different distributed storage application

Figure 10: Distributed training

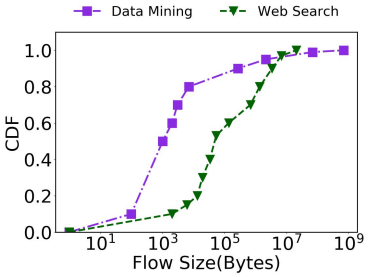


Figure 11: Traffic distributions in large scale simulation

for elephant flows while effectively guaranteeing latency for mice flows.

Mice Flows. As shown in Figure 12(b) and 12(c), ACC outperforms the static ECN benchmarks for mice flows. Compared to $SECN_1$, ACC reduces the average and 99th percentile FCT for mice flows by up to 5.7% and 15.8% at 90% load, respectively. Compared to $SECN_2$, ACC achieves 17.3% and 47.5% lower average FCT and 99th percentile FCT, respectively. The reason is that SECN 1 sacrifices throughput for better latency. This indicates that ACC can effectively reduce the FCT for mice flows.

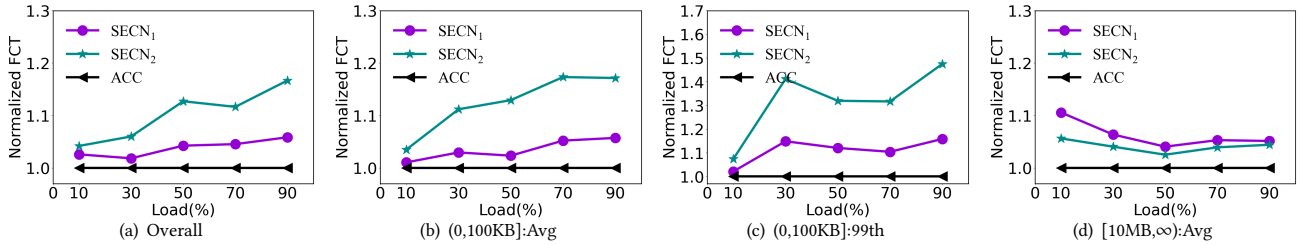
Elephant Flows. $SECN_1$ achieves comparable performance as $SECN_2$. However, it is slightly worse than ACC. As shown in Figure 12(d), ACC outperforms $SECN_2$ at high loads. For example, compared to $SECN_2$, ACC presents 4.4% lower average FCT for elephant flows at 90% load.

Temporally & Spatially Heterogeneous Traffic. To illustrate that ACC can adapt to the temporal and spatial changes of traffic, we use Web Search and Data Mining workloads based on the distribution given in Figure 11. The traffic load is chosen from {60%, 70%, 80%, 90%}, and the source and destination of each flow are randomly chosen from the servers. We run experiments ten times and report the average value. For example, as shown in Figure 13(a), compared to $SECN_1$, ACC reduces the average and 99th percentile FCT for mice flows by up to 8.7% and 24.3% respectively, while achieving 8.6% lower average FCT for elephant flows. Compared to $SECN_2$, ACC outperforms the average and 99th percentile FCT for mice flows by up to 28.6% and 58.3% respectively while achieving 21.1% lower overall average FCT. As shown in Figure 13(b), ACC always has better performance than $SECN_1$ and $SECN_2$. These results have verified that ACC are adaptive to temporal and spatial traffic variation.

Simulation Study on Centralized Design and Distributed Design. We compare the distributed and centralized design through simulation. We use a 96-host leaf-spine topology with 4 leaf switches and 2 spine switches. As discussed in Section 3.2, due to large space of network actions $\{55 \times 20\}^{\{96+4+4\}}$ ($\approx 10^{312}$), the centralized DRL can not converge. Thus, it is impossible to deploy the centralized design in the realistic system. To simply the design, we apply the same setting for all uplink ports or downlink ports because of the symmetric topology. Besides, we sampled some of the actions to further reduce action space. By doing so, we reduce the large action space from $\{55 \times 20\}^{\{96+4+4\}}$ to hundreds of actions in this simulation. As shown in Figure 14, compared with $SECN_1$, C-ACC achieves 16% and 25% lower average FCT and 99th percentile FCT. Compared with $SECN_2$, C-ACC achieves 52% and 70% lower average FCT and 99th percentile FCT. It is notable that C-ACC has higher FCT in comparison with D-ACC. This is because that C-ACC assigns the same ECN configuration to switches at the same layer, which leads to improper ECN setting during congestion.

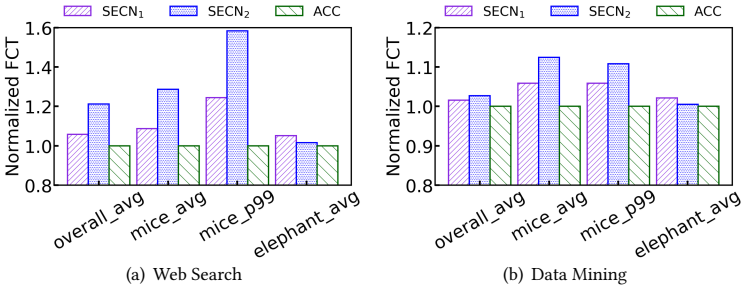
6 OPERATION EXPERIENCE AND DISCUSSION

ACC in Production Datacenters. We have deployed ACC in production datacenters for one year. It supports finance services including business transformation (latency-sensitive), financial analyst jobs (IOPS-intensive, throughput-intensive), and the secure cloud storage services. One datacenter consists of ~300 machines configured with NICs of 25Gbps. Notice that the application and machines



(a) Overall (b) (0,100KB]:Avg (c) (0,100KB]:99th (d) [10MB,∞):Avg

Figure 12: FCT statistics with Web Search workload



(a) Web Search (b) Data Mining

Figure 13: FCT statistics across different workloads and flow sizes in temporally & spatially heterogeneous traffic

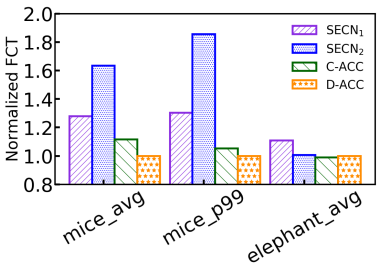


Figure 14: FCT of distributed design (C-ACC) and centralized design (D-ACC)

were incrementally deployed in the datacenter, which causes the aggregation of storage and computation service. We use RDMA for the communication of storage nodes. The communication between storage and computing nodes is TCP. Traffic flow into and out of datacenters is also TCP. Originally, the network applied the static ECN setting suggested by device vendors. After updating the switch with ACC, the IOPS [5] of storage services have improved by ~20%. More important, ACC reduces the burden of tuning the parameters during service migration. Besides switches, ACC is compatible with NICs from different vendors in datacenters located in different area.

Resource Consumption. ACC is deployed in the switch which is resource limited. We estimate the resource consumption of ACC in the switch. Assume the sampling interval is 500μs and the switch has 48 ports. The RDMA data traffic uses one priority queue for each port. Thus, it will takes 48KB/s bandwidth for one port and 2MB/s in total to collect data on PCIe bus. We use a four-layer NN in practice, the number of nodes in each layer is {20,40,40,20}. It requires 14M Flops for one port and 1G Flops in total for computation. The memory consumption for learning model is 30KB. Thus, the cost of ACC is acceptable for most off-the-shelf commodity switches which consists of multiple CPUs and 100Gbps bus[49–51].

Deep Dive of ACC. ACC works by achieving high throughput while keeping low queue length. To illustrate how ACC optimizes queue occupancy, we sampled the queue length of a switch when the burst traffic arrives, as shown in the magnified portion of Figure 15. When the queue length increases, if the current ECN threshold does not change, queue will build up quickly. Notice that ACC reacts to the increasing queue length and high link utilization by using a low ECN threshold to generate more ECN marked packets. When the queue length is approaching to low value, ACC applies a higher ECN threshold to avoid starving which guarantee the throughput performance. Hence, ACC always maintains short queues by adjusting ECN marking threshold based on the state of the environment dynamically.

Stability with Unseen Traffic Pattern. ACC applies learning-based technique which is adaptive to the variable traffic patterns. However, it has met the challenge about serious performance issues over unseen traffic pattern. First, from the operational point of view, operators care the long term benefit achieved from network optimization. It can tolerate the variance in very short period. It is notable that in production environment, RDMA applications usually run for a long time and have the similar traffic patterns. For example, distributed training task takes minutes or hours. The periodic communication leads to the same traffic patterns.

To address this concern, we design an experiment by using a pathological traffic pattern with two completely new traffic flows. We show the FCT statistics of training and compare its results with the recommended ECN configuration as shown in Figure 16. We use WebSearch [7] (P1) and DataMining [22] (P2) as two different traffic workloads and collect performance statistics once every 500 milliseconds. Actually the NN model is an aggressive version without offline-training. At the time of 4.5 second, we switch the traffic immediately from WebSearch to DataMining. ACC causes a high FCT in short time (1 second) and converges to achieve a better performance than that of static ECN. Then at the time of 8.5 second

[5]IOPS is the critical measurement for customers to evaluate the service-level performance, which is highly related to the networking delay and throughput.

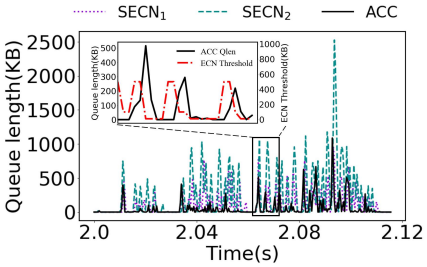
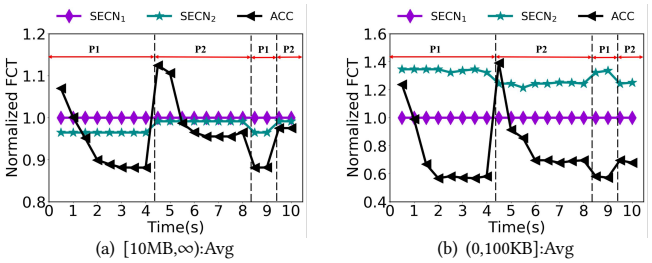


Figure 15: Runtime queue occupancy



(a) [10MB,∞):Avg (b) (0,100KB]:Avg

Figure 16: FCT under different workloads during training

and 9.5 second, we exchange the traffic flows. It is interesting to see that since the model has learned the pattern of arrival traffic, ACC can adapt to the traffic and maintain good performance. In short, ACC achieves 31.1% and 56.2% reduction of the average FCT compared to $SECN_1$ and $SECN_2$, respectively. Besides, it is notable that, ACC does not obviate PFC, which is the last guard to avoid loss and poor performance.

ACC minimizes human intervention for ECN configuration atop standard switches. The fact that the approach has been deployed in a small-scale production setup and has been evaluated using macro-benchmarks is commendable as it also shows the application-layer gains through automatic ECN optimization. In this paper, we focus on the choice of the ECN threshold for fast RDMA network deployment because RoCE processing is fully offloaded to the NIC hardware. An optimal solution may be hybrid: the RL model inference and ECN update is decentralized for quickest response, while online training/RL model update is done by a centralized controller. A global view from the centralized controller would help further improve the system. Furthermore, this work can be extended to a broader scope by using deep learning for software based congestion control with or without PFC [27, 29] and Optimization of the entire set of parameters (transport, ECN, PFC).

7 RELATED WORK

Congestion Control for High-speed DCN. Many congestion control algorithms have been proposed for emerging high-speed datacenter networks. PDQ[23], D3[55], pFabric[8], and HPCC[29] rely on precise in-network state information of switches and update transmission rate for each flow. Switches in PDQ and D3 allocate bandwidth for each flow based on the available link capacity. pFabric achieves near-optimal FCT by using (infinity) priority queues on switches. HPCC adopts the switch loading information directly from INT. However, it remains an open question on how to tune the ECN threshold there. pHost[21], Homa[39] rely on the receiver to send credit packets to determine the sending rate of each flow. TIMELY[35] and Swift[27] are RTT-based schemes to adjust the flow rate at end-host. These approaches achieve remarkable performance. However they require modification of networking stack which is not easy to implementation for RDMA hardware-based implementation.

Tuning ECN in Datacenter. Extensive studies have been proposed to optimize the delay and throughput performance by properly determining the ECN marking threshold. Adaptive AQM mechanisms have been introduced in the traditional TCP/IP network, which update the virtual queue capacity based on the arrival rate [28, 61]. For the modern datacenter networks, ECN* [57] shows that if the instant queue length based ECN threshold is properly tuned, it is possible for RED-like probabilities marking to achieve optimal incast performance. It is notable that though ECN switches accept two threshold parameters, the low and high thresholds, prior pioneer works usually set the two thresholds to the same value, i.e., they only consider one single threshold in the studies. TCN [10] applies the sojourn time, i.e., the amount of time a packet spends in the queue, to mark packets. ECN# [62] studies RTT variation in DCN and marked packets based on both instantaneous and persistent congestion states. ACC considers the operational challenges and uses dynamic ECN on commodity switches.

Learning-based Network Optimization. Learning-based approaches have been applied to handle flow-level traffic optimization [17], and parameters setting for congestion control at end-host [26, 45, 56]. Remy[56] and Indigo [59] learn to adjust the rate from pre-collected sampling network traffic. Vivace [19] utilizes an online algorithm and Aurora [25] uses DRL technique to update sending rate. To avoid performance issue to un-predictable traffic, Orca [5] uses conventional TCP Cubic combined with learning methods. However, most of the learning-based approaches are designed to adjust sending rates at end-host according to the feedback passively. None of them studies in-network optimization on the feedback like ECN. ACC is compatible with ECN-based solutions and adaptive to traffic variations.

8 CONCLUSION

We introduced our operational experience for automatic in-network optimization. ECN is the key to achieve low latency, high throughput communications with the state-of-the-art congestion control schemes. We propose ACC, an pragmatic approach which allows automatic ECN parameters tuning at each switch. By leveraging the deep reinforcement learning, ACC can greatly improve the flow completion time for small messages and maintain high throughput for large messages. Without any modification at end-host, ACC has been quickly deployed in the production datacenter to support storage and computing services stably.

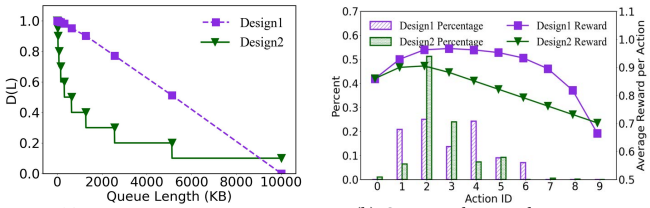
This work does not raise any ethical issues.

ACKNOWLEDGMENTS

The authors thank the anonymous reviewers and our shepherd, Keon Jang, for providing valuable feedback. We would also like to thank Yashar Ganjali, Camtu Nguyen and the teams at Huawei for their contributions to the work.

REFERENCES

[1] InfiniBand Architecture Volume 1 and released specification Volume 2. 2015. https://cw.infinibandta.org/document/dl/7859.
[2] Network Simulator 3. 2019. https://wwwnsnam.org.
[3] IEEE. 802.11Qau. 2011. Priority based flow control (PFC).
[4] S. Abbasloo, C. Y. Yen, and H. J. Chao. 2020. Classic Meets Modern: a Pragmatic Learning-Based Congestion Control for the Internet. In *SIGCOMM '20: Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication.*
[5] Soheil Abbasloo, Chen-Yu Yen, and H Jonathan Chao. 2020. Classic meets modern: A pragmatic learning-based congestion control for the Internet. In *ACM SIGCOMM*. 632–647.
[6] Mohammad Alizadeh, Javanmard Adel, and Balaji Prabhakar. 2011. Analysis of DCTCP: stability, convergence, and fairness. In *ACM SIGMETRICS Performance Evaluation Review*. 73–84.
[7] Mohammad Alizadeh, Albert Greenberg, David A. Maltz, Jitendra Padhye, Parveen Patel, Balaji Prabhakar, Sudipta Sengupta, and Murari Sridharan. 2010. Data Center TCP (DCTCP). In *ACM SIGCOMM*.
[8] Mohammad Alizadeh, Shuang Yang, Milad Sharif, Sachin Katti, Nick McKeown, Balaji Prabhakar, and Scott Shenker. 2013. pFabric: minimal near-optimal datacenter transport. In *ACM SIGCOMM*.
[9] InfiniBand Trade Association. 2014. Supplement to InfiniBand architecture specification volume 1 release 1.2.2 annex A17: RoCEv2 (IP routable RoCE).
[10] Wei Bai, Kai Chen, Li Chen, Changhoon Kim, and Haitao Wu. 2016. Enabling ECN over generic packet scheduling. In *ACM CoNEXT*.
[11] In band Network Telemetry in Barefoot Tofino. 2019. https://www.barefootnetworks.com/use-cases/ad-telemetry.
[12] In band Network Telemetry in Broadcom Tomahawk 3. 2019. https://www.broadcom.com/company/news/product-releases/2372840..
[13] In band Network Telemetry in Broadcom Trident 3. 2019. https://www.broadcom.com.
[14] Claude Barthels, Simon Loesing, Gustavo Alonso, and Donald Kossmann. 2015. Rack-scale in-memory join processing using RDMA. In *ACM SIGMOD*.
[15] Flexible Input/Output benchmark. 2017. http://github.com/axboe/fio.
[16] Linpack benchmark. 2018. http://www.netlib.org/benchmark/hpl/.
[17] Li Chen, Justinas Lingys, Kai Chen, and Feng Liu. 2018. Auto: Scaling deep reinforcement learning for datacenter-scale automatic traffic optimization. In *ACM SIGCOMM*.
[18] Tianqi Chen, Mu Li, Yutian Li, Min Lin, Naiyan Wang, Minjie Wang, Tianjun Xiao, Bing Xu, Chiyuapn Zhang, and Zheng Zhang. 2015. MXNet: A Flexible and Efficient Machine Learning Library for Heterogeneous Distributed Systems. *Statistics* (2015).
[19] M. Dong, Tong Meng, Doron Zarchy, E. Arslan, Y. Gilad, B. Godfrey, and M. Schapira. 2018. PCC Vivace: Online-Learning Congestion Control. In *USENIX NSDI*.
[20] Sally Floyd and Van Jacobson. 1993. Random early detection gateways for congestion avoidance. *IEEE/ACM Transactions on networking* 1, 4 (1993), 397–413.
[21] Peter X Gao, Akshay Narayan, Gautam Kumar, Rachit Agarwal, Sylvia Ratnasamy, and Scott Shenker. 2015. phost: Distributed near-optimal datacenter transport over commodity network fabric. In *ACM CoNEXT*.
[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 *ACM SIGCOMM*.
[23] Chi-Yao Hong, Matthew Caesar, and P Brighten Godfrey. 2012. Finishing flows quickly with preemptive scheduling. *ACM SIGCOMM Computer Communication Review* 42, 4 (2012), 127–138.
[24] Horovod. 2018. https://github.com/horovod/horovod.
[25] Nathan Jay, Noga Rotman, Brighten Godfrey, Michael Schapira, and Aviv Tamar. 2019. A deep reinforcement learning perspective on internet congestion control. In *International Conference on Machine Learning (ICML)*. 3050–3059.
[26] Lavanya Jose, Lisa Yan, Mohammad Alizadeh, George Varghese, Nick McKeown, and Sachin Katti. 2015. High speed networks need proactive congestion control. In *ACM Workshop on Hot Topics in Networks*.
[27] Gautam Kumar, Nandita Dukkipati, Keon Jang, Hassan M. G. Wassel, Xian Wu, Behnam Montazeri, Yaogong Wang, Kevin Springborn, Christopher Alfeld, Michael Ryan, David Wetherall, and Amin Vahdat. 2020. Swift: Delay is Simple and Effective for Congestion Control in the Datacenter. In *ACM SIGCOMM*.
[28] Srisankar Kunniyur and Rayadurgam Srikant. 2001. Analysis and design of an adaptive virtual queue (AVQ) algorithm for active queue management. In *ACM SIGCOMM*.
[29] Yuliang Li, Rui Miao, Hongqiang Harry Liu, Yan Zhuang, Fei Feng, Lingbo Tang, Zheng Cao, Ming Zhang, Frank Kelly, Mohammad Alizadeh, and et al. 2019. HPCC: High Precision Congestion Control. In *ACM SIGCOMM*.
[30] Xiaoyi Lu, Nusrat S Islam, Md Wasi-Ur-Rahman, Jithin Jose, Hari Subramoni, Hao Wang, and Dhabaleswar K Panda. 2013. High-performance design of Hadoop RPC with RDMA over InfiniBand. In *International Conference on Parallel Processing (ICPP)*. IEEE, 641–650.
[31] Xiaoyi Lu, Md Wasi Ur Rahman, Nahina Islam, Dipti Shankar, and Dhabaleswar K Panda. 2014. Accelerating spark with RDMA for big data processing: Early experiences. In *Annual Symposium on High Performance Interconnects*.
[32] Xiaoyi Lu, Dipti Shankar, Shashank Gugnani, and Dhabaleswar K DK Panda. 2016. High-performance design of apache spark with RDMA and its benefits on various workloads. In *International Conference on Big Data (Big Data)*. IEEE.
[33] Youyou Lu, Jiwu Shu, Youmin Chen, and Tao Li. 2017. Octopus: an RDMA-enabled distributed persistent memory file system. In *USENIX ATC*.
[34] Jianmin Chen Zhifeng Chen Andy Davis Jeffrey Dean Matthieu Devin Sanjay Ghemawat Geoffrey Irving Michael Isard Manjunath Kudlur Josh Levenberg Rajat Monga Sherry Moore Derek G. Murray Benoit Steiner Paul Tucker Vijay Vasudevan Pete Warden Martin Wicke Yuan Yu MartÃŋn Abadi, Paul Barham and Xiaoqiang Zheng. 2016. Tensorflow: A system for large-scale machine learning. In *USENIX OSDI*.
[35] Radhika Mittal, Vinh The Lam, Nandita Dukkipati, Emily Blem, Hassan Wassel, Monia Ghobadi, Amin Vahdat, Yaogong Wang, David Wetherall, and David Zats. 2015. TIMELY: RTT-based Congestion Control for the Datacenter. In *ACM SIGCOMM*.
[36] Radhika Mittal, Alexander Shpiner, Aurojit Panda, Eitan Zahavi, Arvind Krishnamurthy, Sylvia Ratnasamy, and Scott Shenker. 2018. Revisiting Network Support for RDMA. In *ACM SIGCOMM*.
[37] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. 2016. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*. PMLR, 1928–1937.
[38] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. 2015. Human-level control through deep reinforcement learning. *nature* 518, 7540 (2015), 529–533.
[39] Behnam Montazeri, Yilong Li, Mohammad Alizadeh, and John Ousterhout. 2018. Homa: A Receiver-driven Low-latency Transport Protocol Using Network Priorities. In *ACM SIGCOMM*.
[40] Mellanox Perftest Package. 2017. https://community.mellanox.com/docs/DOC-2802.
[41] SparkRDMA Shuffle Manager Plugin. 2018. https://github.com/Mellanox/SparkRDMA.
[42] Haonan Qiu, Xiaoliang Wang, Tianchen Jin, Zhuzhong Qian, Baoliu Ye, Bin Tang, Wenzhong Li, and Sanglu Lu. 2018. Toward Effective and Fair RDMA Resource Sharing. In *Asia-Pacific Workshop on Networking (APNet)*.
[43] K Ramakrishnan, Sally Floyd, and D Black. 2001. RFC3168: The addition of explicit congestion notification (ECN) to IP.
[44] Danfeng Shan and Fengyuan Ren. 2018. ECN Marking With Micro-Burst Traffic: Problem, Analysis, and Improvement. *IEEE/ACM Transactions on Networking* 26, 4 (2018), 1533–1546.
[45] Anirudh Sivaraman, Keith Winstein, Pratiksha Thaker, and Hari Balakrishnan. 2014. An experimental study of the learnability of congestion control. *ACM SIGCOMM Computer Communication Review* 44, 4 (2014), 479–490.
[46] Richard S Sutton and Andrew G Barto. 2018. *Reinforcement learning: An introduction*. MIT press.
[47] Iperf: The tcp/udp bandwidth measurement tool. 2005. *dast. nlanr. net/Projects* (2005), 38.
[48] Quantum Espresso test suite. 2019. https://www.quantum-espresso.org/.
[49] Barefoot Tofino. 2019. https://www.barefootnetworks.com/.
[50] Broadcom Tomahawk. 2019. https://www.broadcom.com/company/news/product-releases.
[51] Broadcom Trident. 2019. https://www.broadcom.com.
[52] Hasselt Van, Guez Hado, Arthur, and Silver David. 2016. Deep reinforcement learning with double q-learning. In *AAAI conference on artificial intelligence*.
[53] Yandong Wang, Li Zhang, Jian Tan, Min Li, Yuqing Gao, Xavier Guerin, Xiaoqiao Meng, and Shicong Meng. 2015. HydraDB: a resilient RDMA-driven key-value middleware for in-memory cluster computing. In *International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*. IEEE, 1–11.
[54] Xingda Wei, Zhiyuan Dong, Rong Chen, and Haibo Chen. 2018. Deconstructing RDMA-enabled Distributed Transactions: Hybrid is Better!. In *USENIX OSDI*.
[55] Christo Wilson, Hitesh Ballani, Thomas Karagiannis, and Ant Rowtron. 2011. Better never than late: Meeting deadlines in datacenter networks. *ACM SIGCOMM Computer Communication Review* 41, 4 (2011), 50–61.



(a) Queue length reward (b) Converged action decisions

Figure 17: Reward function designs and the comparison of converged action decisions

[56] Keith Winstein and Hari Balakrishnan. 2013. Tcp ex machina: Computer-generated congestion control. *ACM SIGCOMM Computer Communication Review* 43, 4 (2013), 123–134.

[57] Haitao Wu, Jiabo Ju, Guohan Lu, Chuanxiong Guo, Yongqiang Xiong, and Yongguang Zhang. 2012. Tuning ECN for data center networks. In *ACM CoNEXT*.

[58] Ming Wu, Fan Yang, Jilong Xue, Wencong Xiao, Youshan Miao, Lan Wei, Haoxiang Lin, Yafei Dai, and Lidong Zhou. 2015. GRAM: scaling graph computation to the trillions. In *ACM Symposium on Cloud Computing (SoCC)*.

[59] Francis Y Yan, Jestin Ma, Greg D Hill, Deepti Raghavan, Riad S Wahby, Philip Levis, and Keith Winstein. 2018. Pantheon: the training ground for Internet congestion-control research. In *USENIX ATC*.

[60] Lingbo Tang Yongqing Xi Pengcheng Zhang Wenwen Peng Bo Li Yaohui Wu Shaozong Liu Lei Yan Fei Feng Yan Zhuang Fan Liu Pan Liu Xingkui Liu Zhongjie Wu Junping Wu Yixiao Gao, Qiang Li, Jinbo Wu Jiaji Zhu Haiyong Wang Dennis Cai Zheng Cao, Chen Tian, and Jiesheng Wu. 2021. When Cloud Storage Meets RDMA. In *USENIX NSDI*.

[61] Honggang Zhang, Don Towsley, C. V. Hollot, and Vishal Misra. 2003. A Self-Tuning Structure for Adaptation in TCP/AQM Networks. *SIGMETRICS Perform. Eval. Rev.* 31, 1 (june 2003).

[62] Junxue Zhang, Wei Bai, and Kai Chen. 2019. Enabling ECN for datacenter networks with RTT variations. In *ACM CoNEXT*.

[63] Qiao Zhang, Vincent Liu, Hongyi Zeng, and Arvind Krishnamurthy. 2017. High-resolution measurement of data center microbursts. In *ACM Internet Measurement Conference (IMC)*. 78–85.

[64] Yiwen Zhang, Juncheng Gu, Youngmoon Lee, Mosharaf Chowdhury, and Kang G. Shin. 2017. Performance Isolation Anomalies in RDMA. In *KBNets*. ACM.

[65] Yibo Zhu, Haggai Eran, Daniel Firestone, Chuanxiong Guo, Marina Lipshteyn, Yehonatan Liron, Jitendra Padhye, Shachar Raindel, Mohamad Haj Yahia, and Ming Zhang. 2015. Congestion control for large-scale RDMA deployments. In *ACM SIGCOMM*.

[66] Yibo Zhu, Monia Ghobadi, Vishal Misra, and Jitendra Padhye. 2016. ECN or Delay: Lessons Learnt from Analysis of DCQCN and TIMELY. In *ACM CoNEXT*.

APPENDIX

Appendices are supporting material that has not been peer-reviewed.

.1 Impact of Reward Design

The design of reward function has a great impact on the performance of DRL agent. To achieve a deep understanding on the design of reward function, we demonstrate two typical reward function designs here. As shown in Figure 17(a), design-1 applies linear function, i.e., $D(L) = 1 - L/Qmax$ in Equation (2) (Q_{max} is the value of allocated buffer in one service pool $Qmax = 10MB$ in the testbed). Design-2 is our mapping function of queue length reward, which maps the queue depth stepwise in Figure 4. To evaluate the performance of two designs, we choose ten levels of high ECN thresholds. Under the scenario of incast congestion, the action decisions made by two designs are shown in Figure 17(b). We can see that ACC with reward Design-2 achieves the expected action. To reveal the reason, we review the reward functions. For Design-1, if queue length is directly mapped to reward by a linear function, the rewards are similar for different actions. To introduce differentiation of different actions, we propose the step function. it uses fine-grained intervals for shallow queue depth and coarse-grained intervals for large queue size. The reasons for this piecewise mapping design lie in: (1) Most network congestion happens at a small queue size (<1MB). So it is necessary to differentiate rewards at small queue depth. (2) A large queue length value (>1MB) always implies a long queue latency (>300us at 25Gbps link), which is two orders of magnitude larger than transmit latency and it suffices requirement of the design.