DECC: Achieving Low Latency in Data Center Networks With Deep Reinforcement Learning

Yi Liu, Jiangping Han, *Member, IEEE*, Kaiping Xue, *Senior Member, IEEE*, Jian Li, *Member, IEEE*, Qibin Sun, *Fellow, IEEE*, and Jun Lu

***Abstract*—Data Center Networks (DCNs) suffer from synchronized bursts for network topology and parallel applications, leading to buffer overflows at switches and increasing network delay. To overcome this problem, some congestion control algorithms like DCTCP use Explicit Congestion Notification (ECN) to notify in-network congestion and reduce switch buffer occupancy. However, the traditional Additive Increase Multiplicative Decrease (AIMD) method causes high fluctuation of round-trip time (RTT) in DCNs. Some intelligent congestion control algorithms designed for Internet can achieve great flexibility, but are not applicable in DCNs for a lack of accurate congestion feedback. In this paper, we analyze the deficiencies of utilizing RTT as congestion signals and the applicability of learning algorithms in DCNs. Then, we propose DECC, a smart TCP congestion control algorithm for DCNs, which combines Deep Reinforcement Learning (DRL) with ECN to achieve high bandwidth utilization as well as low queuing delay. DECC fully utilizes precise in-network feedback and formulates several QoS requirements to a multi-objective function. Meanwhile, it decouples cwnd adjustment with DRL decision making to gradually learn the optimal congestion control policy in real-time. We evaluate the performance of DECC in various scenarios. Simulation results show that DECC can reduce the queue length at bottleneck switches by more than 50% compared to DCTCP, while maintaining high bandwidth utilization and reducing Flow Completion Time (FCTs) under burst traffic.**

***Index Terms*—Data center networks, TCP incast, deep reinforcement learning, congestion control.**

I. Introduction

With the rapid development of network applications, data centers have been established and flourished to meet strict requirements for computation and storage [1]. In large data centers, there are numerous servers interconnected by switches to carry complex and demanding applications, forming data center networks (DCNs). Complex application traffic and special topology cause many characteristics in DCNs different from traditional networks:

- *Highly synchronized bursts*. The request of one target host may be divided into multiple small tasks, such as super computing, Web search or distributed file requests, which need to be completed by several other servers. After finishing the task, they respond sub results to the target node almost at the same time, resulting in highly synchronized bursts. These synchronized bursts exceed the egress port capacity of switches, causing queue to build up and may lead to congestive packet drops [2].
- *High-bandwidth links and shallow-buffer switches*. he link bandwidth grows rapidly to achieve the high-speed transmission of throughput-sensitive flows in DCNs. By contrast, the buffer in commodity switches expands slowly for fast processing, not compatible with the growth of per port bandwidth [3].

High-speed links, shallow buffers, and high bursts make special demands and challenges to DCNs. High-speed links bring extremely low link delay, hence the queuing delay at the switches accounts for a large proportion of the total delay and influences the real Round Trip Time (RTT) significantly. The data transmission in DCNs is queuing delay sensitive, which makes the switch buffer occupancy a crucial factor. However, the micro-burst causes a persistent queue backlog in the switch when multiple concurrent flows send their packets to the same output port. The small buffer size cannot carry the packet occupancy. As a consequence, lots of packets are dropped due to the buffer overflow. This phenomenon is called TCP incast [4], [5]. In severe cases, a whole window of packets will be dropped continuously, unable to trigger fast retransmission and recovery, resulting in TCP timeout and large Flow Completion Time (FCT).

To reduce the queuing delay and solve the incast problem, congestion control algorithms must limit the queue length and buffer occupancy of switches. However, the state-of-art algorithms react slowly to in-network congestion and are not sensitive to buffer occupancy. To overcome this issue, some congestion control algorithms [6], [7], [8] are proposed to utilize Explicit Congestion Notification (ECN) mechanism to perceive in-network congestion at end hosts, so as to reduce the congestion window (cwnd) in advance. However, some defects make them insufficient to meet the growing requirements of applications in DCNs: 1) Traditional Additive Increase Multiplicative Decrease (AIMD) rule converges

Manuscript received 16 August 2022; revised 18 March 2023; accepted 5 June 2023. Date of publication 12 June 2023; date of current version 12 December 2023. This work is supported in part by the National Natural Science Foundation of China under Grant No. 61972371 and No. U19B2023, Youth Innovation Promotion Association of the Chinese Academy of Sciences (CAS) under Grant No. Y202093, and the Fundamental Research Funds for the Central Universities. The associate editor coordinating the review of this article and approving it for publication was J. Yang. *(Corresponding authors: Jiangping Han; Kaiping Xue.)*

Yi Liu, Jiangping Han, Kaiping Xue, Jian Li, and Qibin Sun are with the School of Cyber Science and Technology, University of Science and Technology of China, Hefei 230027, Anhui, China (e-mail: jphan@ustc.edu.cn; kpxue@ustc.edu.cn).

Jun Lu is with the School of Cyber Science and Technology and the Department of Electronic Engineering and Information Science, University of Science and Technology of China, Hefei 230027, Anhui, China.

Digital Object Identifier 10.1109/TNSM.2023.3285110

slowly, leading to bandwidth waste; 2) Traditional algorithms bring high and unstable buffer occupancy at switches, vulnerable to frequent bursts; 3) The rigid cwnd adjustment strategy cannot adapt to fast changing and uncertain traffic patterns. To improve adaptability, another class of learning-based congestion control algorithms [9], [10], [11], [12], [13], [14], [15] have been proposed successively. Unfortunately, all these learning-based solutions are designed to adjust sending rates at end hosts passively, or independently consider the in-network optimization. Moreover, they only depends RTT to roughly describe the network congestion state.

Motivated by the above observations, in this paper, we provide a DRL-based ECN-assisted congestion control algorithm, named DECC, to combine the explicit congestion degree feedback at the bottleneck with proactively learning-based congestion control decisions at end hosts. DECC adopts a new and efficient objective function including ECN marking probability, to obtain an optimal window adjustment strategy and proactively handle the congestion, meanwhile eliminating the complex modification on switches. DECC further decouples the cwnd adjustment with DRL decision making. It learns the cwnd adjustment parameter within a fine-grained action space through the DRL agent, achieving precise congestion control with high effectiveness in DCNs. Moreover, for high learning efficiency as well as great adaptability, the agents first learn a universal cwnd adjusting strategy offline, then interact with the realistic network and train the network with online learning, which is detailed in Section IV. DECC is verified to decrease the buffer occupancy at any bottleneck switch, achieving low queuing delay and high burst tolerance.

Our contributions are summarized as follows:

- We design a novel congestion control algorithm for DCNs, named DECC, to combine the explicit congestion degree information with proactively learning-based congestion control decisions at end hosts. DECC reveals the deficiencies of RTT measurement and takes several QoS requirements into the learning goal, thus obtaining the optimal congestion control strategy.
- DECC forms the objective function into a direct reward function and decouples cwnd adjustment with DRL decision making. While taking advantage of the intelligence and adaptability of DRL, DECC learns the fine-grained cwnd adjustment parameter to achieve high effectiveness in DCNs.
- We evaluate DECC with large-scale simulations in several network topologies and traffic patterns. The results show that DECC can shorten the queue length at bottleneck switches by more than 50% compared with traditional schemes. Even under serious bursts, DECC can decrease FCTs and reduce timeouts of all flows, providing high burst tolerance.

The rest of the paper is structured as follows. Section II briefly survey the background and related works. Section III explains the motivation of our work with a simulation result. The proposed DECC is designed in details in Section IV. Thereafter, the performance evaluation and analysis are shown in Section V, and conclusions are drawn in Section VI.

II. Background and Related Work

DCNs carry large-scale applications and services with diverse traffic patterns. Most of the traffic in DCNs is generated by the communication between internal servers while few traffic comes from clients in public networks [1]. Meanwhile, the links in DCNs have high bandwidth and low propagation latency. This traffic pattern results in very short RTT and sensitivity to queuing delay. Traditional TCP is initially designed for the public network transmission which utilizes packet loss to inform the senders of congestion and tends to deplete the switch buffer in exchange for higher network throughput. Therefore, it is unreliable to directly apply TCP to DCNs for better transmission performance. Considering different mechanisms, previous works of congestion control in DCNs can be divided into following categories.

Traditional end-to-end congestion control: e.g., DCTCP, D2TCP [8], L2DCT [7]. DCTCP is one of the most commonly used end-to-end transmission protocols in DCNs, which enables ECN to estimate congestion probability and reduces cwnd in advance. The cwnd is adjusted by: $cwnd = cwnd \cdot (1 - \frac{\alpha}{2})$, where α refers to the ratio of total number of marked packets in a transmission window size, that is the frequency of packets being marked. After DCTCP, some improved algorithms for specific scenarios and requirements have been proposed successively. For example, D2TCP defines a deadline urgency e (the ratio of expected completion time to remaining deadline) for flows with specified deadlines, then cwnd is cut by $\frac{\alpha^e}{2}$ to reduce the deadline missing rate of flows; L2DCT allocates a weight function f inversely related to the data transmission volume for each flow, and cuts cwnd by $\frac{\alpha^f}{2}$. Besides, there are many congestion control algorithms based on switch scheduling [16], [17], [18] or driven by receivers [19], [20], [21], [22], [23].

Congestion control based on special network interface cards (NIC): e.g., DCQCN [24], HPCC [25], TIMELY [26], Swift [27]. DCQCN works like DCTCP depending on ECN to modulate sending rates in Remote Direct Memory Access (RDMA) transfers. TIMELY uses specialized NIC to measure RTT in DCNs with enough precision and utilizes the rate of RTT variation to predict in-network congestion. HPCC utilizes the in-network telemetry (INT) to obtain detailed switch and link information. Swift uses precise RTT measurements to adjust cwnd in packets with an AIMD algorithm, aiming to maintain the delay around a target delay. Swift decomposes the end-to-end RTT into NIC-to-NIC and endpoint delay components to respond separately to the congestion in fabric versus at hosts or NICs. These algorithms generally require high switch or NIC configuration and computing power to break through traditional performance bottlenecks, and general commercial switches cannot meet the demand.

Intelligent congestion control: Learning algorithms are widely applied in the network [28], [29], [30]. QTCP [9] algorithm applies Q-learning to TCP cwnd adjustment, aiming to make full use of bandwidth and adapt to the network environment. Nie etc. [10] use RL techniques to dynamically adjust initial windows of TCP and select appropriate congestion control schemes, but it is not suitable in DCNs as transmission

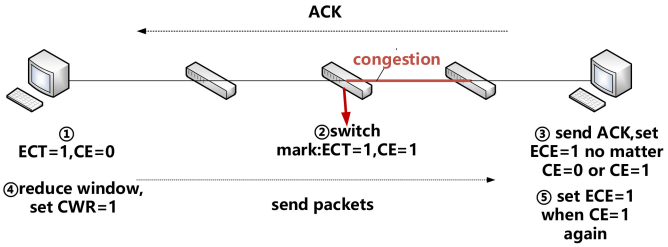


Fig. 1. The work process of ECN.

protocols in DCNs are fixed and controllable, not as complex and diverse as protocols in public networks. The above shows some applications of RL algorithms in traditional networks. For DCNs, AuTO [11] is proposed to shorten the completion time of short flows by learning the thresholds of each priority, and achieve high throughput of long flows by learning sending rates with DRL. To optimize the performance of ECN in large-scale RDMA networks, ACC [12] is designed to deploy a distributed DRL agent in each switch to tune the ECN thresholds independently. Moreover, Remy [13] and Indigo [14] learn to adjust the rate from pre-collected sampling network traffic. Aurora [15] uses DRL technique to update sending rate.

To express the congestion degree more precisely and control it more promptly, ECN is a standard mechanism described in RFC 3168 [31] for network devices to notify congestion at network bottlenecks, without resorting to packet drops. It utilizes ECN Capable Transport (ECT) and Congestion Experienced (CE) in IP header, Congestion Window Reduced (CWR) and ECN Echo (ECE) in TCP header to convey control signals [32]. The working process of ECN is showed in Fig. 1. At switches, if the queue length exceeds a threshold K, the exceeding packets will be marked through the CE code point in IP header. The threshold $K > \frac{RTT \times C}{7}$, where RTT is the round-trip time and C is the packet sending rate. When the receiver receives a marked data packet, the related ACK's ECE code point at TCP header will be marked and the ACK is sent without delay, but immediately after detecting the mark. Therefore, ECN can notify senders of the current congestion probability (queue occupancy) after the queue length exceeds the threshold.

III. Motivation

Traditional algorithms represented by DCTCP cannot adapt to the changeable network environment for the rigid cwnd adjustment and long convergence time. And long flows occupies the buffer, resulting in packet loss of short flows, especially in current high-speed shallow-buffer data centers. QTCP first breaks the rules of traditional congestion control algorithms and applies RL to TCP congestion control in public networks, which integrate reinforcement-based Q-learning framework with TCP design. Unfortunately, we find that in DCNs, QTCP performs even worse than DCTCP in some scenarios, like burst with background flows.

QTCP designed for Internet is not suitable for DCN scenarios: We illustrate this problem with a small ns-3 simulation experiment. As Fig. 2 shows, we connect 44 hosts to one

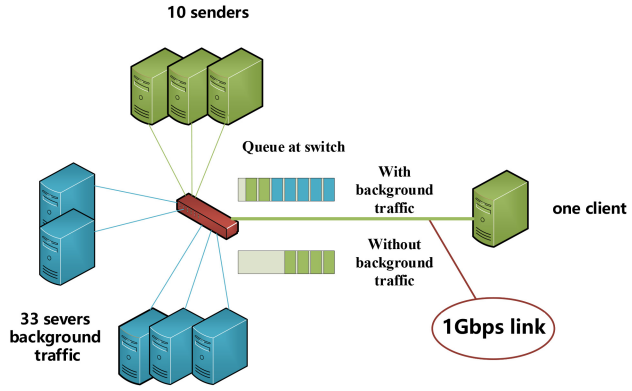


Fig. 2. The simulation topology.

TABLE I
The Query Delay Comparison

	without background traffic	with background traffic
DCTCP	9.32ms	10.11ms
QTCP	9.09ms	41.25ms

shared-memory switch with 4MB buffer and 1Gbps links. 11 of these hosts form a 10-1 incast pattern, with 10 servers synchronously sending data to the remaining client. The client requests 1MB data from servers for query, so each server need to send 100KB to the client almost synchronously. The query is finished until all ten flows are completed. Hence the query delay refers to the maximum completion time of all flows generated by the request. The client has requested 1000 times according to Poisson arrival. Setting DCTCP as a benchmark, we compare and analyze the query delay using QTCP with and without background traffic. To generate the background traffic, we make the remaining 33 servers to keep sending data in the network. Specifically, each host randomly selects other two hosts to send data, totally forming 66 long-lived flows as background traffic to occupy the shared buffer of the switch.

We repeat experiments and attain the average query delay with two algorithms showed in TABLE I. Without background traffic, there are only 10 synchronous short flows sharing 4MB buffer size. The minimum request time is around 8ms, and QTCP performs well in this scenario with no timeouts, as well as DCTCP. However, after we start background traffic to increase buffer pressure, QTCP flows suffer from serious timeouts and the request takes more than 40ms to complete, while DCTCP flows are not affected much. QTCP is designed for long-RTT public networks and utilizes RTT to perceive congestion, with generally 0.23 seconds decision time interval while RTT in DCNs is in microseconds. QTCP flows are not sensitive to the congestion caused by the queue building up at switches in very short time and cannot respond quickly. Hence with background traffic, long-lived flows deplete the shared buffer space, which leaves less headroom to absorb incast bursts. Therefore, the variation of RTT in DCNs is not enough to describe the congestion degree, we need more accurate and timely congestion feedback.

Accurate congestion perception is required: In congestion control algorithms, round-trip delay is an inaccurate congestion feedback especially in multi-hop networks. The total round-trip delay of a packet in the network can be expressed as

$$d_{total} = d_{node} + d_{prop} + d_{queue}, \tag{1}$$

where d_{node} is the processing delay (including end nodes and switches), d_{prop} is the propagation delay, and d_{queue} is the queuing delay at switches. In high-speed DCNs, d_{prop} is much smaller than d_{queue}. In multi-bottleneck networks, d_{queue} includes the delay at several bottleneck switches,

$$d_{queue} = d_{q_1} + d_{q_2} + \cdots + d_{q_n}, \tag{2}$$

where d_{q_n} is the queue delay at the bottleneck switch n. Suppose d_{prop} and d_{node} are unchanged, we have

$$\Delta d_{total} = \Delta d_{q_1} + \Delta d_{q_2} + \cdots + \Delta d_{q_n}. \tag{3}$$

Suppose the application service causes that the congestion of $bottleneck_1$ is relieved and the congestion of $bottleneck_2$ increases, finally d_{total} remain unchanged. Hence the sender perceives that the network congestion state is unchanged, but actually the overall congestion intensifies. The above argument demonstrates that RTT cannot accurately describe the degree of in-network congestion. The buffer occupancy at bottleneck switches is more accurate than the variation of RTT, which can be estimated by ECN.

Therefore, to break the limitations of traditional algorithms and make the new intelligent congestion control algorithm suitable for DCNs, we propose an ECN-assisted DRL-based congestion control algorithm, and design the algorithm composition according to the characteristics of DCNs, dedicated to reducing the queue length at switches and efficiently handling synchronized bursts meanwhile obtaining high bandwidth utilization.

IV. Design of DECC

In this section, we discuss the application of Deep Q-learning Network (DQN) [33] and introduce our algorithm DECC, including the design, description and theoretical analysis of state, action, objective function, reward mechanism and training algorithm, to describe the working process of the proposed system.

A. Problem Formulation

We formulate the congestion control in DCNs as a RL problem based on Markov Decision Process (MDP). MDP is a cyclic process in which an agent takes an action to interact with the environment, moving to the next state and obtaining rewards. Mathematically, a MDP $M = <S, A, R, P>$ is defined as follows:

- *State space S:* finite states which are sufficient to represent environment conditions.
- *Action space A:* a collection of limited actions that an agent can take to interact with the environment.
- *Reward R(s, a):* the reward attained after taking action $a \in A$ at state $\mathbf{s}$. It depends on the objective function which is composed of optimization goals. If the objective increases, the action taken is worth encouraging and gets positive rewards to increase the probability of being taken next time, otherwise gets negative rewards to reduce it.
- *Probability* $P(\mathbf{s}_{t+1}|\mathbf{s}_t, a)$*:* the probability of choosing action $a \in A$ at state $\mathbf{s}_t$ and move to state $\mathbf{s}_{t+1}$, where t is the decision time.

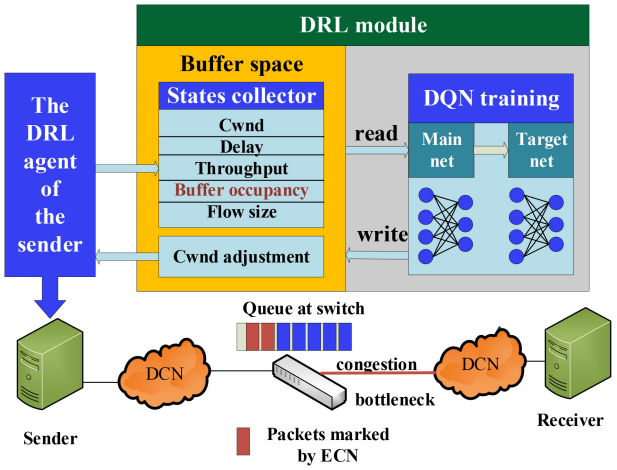


Fig. 3. DECC framework.

Let π denote a policy to select actions based on an given state $\mathbf{s}$: $a = \pi(\mathbf{s})$. The goal of RL is to learn an optimal policy π^* for selecting an action in a given state and maximize discounted accumulated rewards (γ denotes the discounting factor):

$$\pi^* = \arg\max_{\pi} E^{\pi}\left(\sum_{t=0}^{\infty}\gamma^t r_t\right). \tag{4}$$

This problem is equivalent to finding the maximum state-action value:

$$Q^*(\mathbf{s}, a) = \max_{\pi} E^{\pi}\left[\sum_{t=0}^{\infty}\gamma^t r_{t+1}|\mathbf{s}_t = \mathbf{s}, a_t = a\right], \tag{5}$$

which refers to the value of taking action a at state $\mathbf{s}$ and decision time t.

To guarantee the congestion control in DCNs a MDP, we define:

- *State space S:* cwnd size, round trip delay, throughput, switch buffer occupancy, and the length of the current flow.
- *Action space A:* sending window adjustment strategy.

Hence the next state of the network only depends on the current state and decisions, which satisfies the Markov property:

$$P(\mathbf{s}_{t+1}|\mathbf{s}_t, a_t, \mathbf{s}_{t-1}, a_{t-1}, \ldots, \mathbf{s}_0, a_0) = P(\mathbf{s}_{t+1}|\mathbf{s}_t, a_t). \tag{6}$$

Next, we introduce the design of DECC framework.

B. DECC Framework

The cwnd adjustment strategy in congestion control is actually a MDP. The network performance is directly transformed into an evaluation index, finally the optimal policy will be obtained through interacting with the DCN and learning. The overall framework of DECC is shown in Fig. 3. There

is a DRL agent working at, which collects network state information by interacting with the DCN environment, and caches it in the sender buffer. The DRL module takes out the latest data from the buffer every time interval to learn and adjust parameters, then outputs the current optimal window adjustment action, which is stored in the buffer and read by the sender to adjust cwnd.

1) States: State space consists of five observations from the DCN measured in a time interval, finally defined as $\mathbf{s} = \{s_1, s_2, s_3, s_4, s_5\}$, where s_1 represents the average cwnd size, s_2 represents the average RTT, s_3 represents the average throughput, which is calculated by dividing the sent bytes by the time interval, s_4 represents the average switch buffer occupancy, which is calculated by the percentage of ECN marked packets, and s_5 represents the current flow length (the remaining data waiting to be sent of a flow). As the states is observed every time interval (several RTTs), s_1, s_2, s_3 and s_4 are average values calculated in the time interval. s_5 is the current flow length at the beginning of the time interval. To calculate s_5, the total flow size is aware to the sender while generating applications. Then, s_5 is calculated by subtracting the sent bytes from the overall flow length.

Cwnd size and round trip delay can describe the transmission rate at the sender, and the throughput reflects the overall performance of the network. In general network, these three factors can fully describe a network's condition. However, there are much burst traffic in DCNs, hence we add the switch buffer occupancy to describe the congestion degree of current network. When the switch queue length exceeds a given threshold, the CE Code Points of packets exceeding will be marked, then the receiver marks the ACK corresponding to these packets with ECN. The sender counts the number of marked ACKs in one sending window to estimate the congestion degree of the network (switch buffer occupancy). The current flow length is also considered as our algorithm needs to treat long and short flows differently. In the same network state, it should converge to different window adjustment strategies for long and short flows.

2) Action: DECC aims to find the optimal window adjustment strategy, so the action space is designed as increasing cwnd, reducing cwnd or maintaining the original size. When designing the action space of DECC, we adopted finer granularity, which is compared to QTCP and most common learning algorithms in the public network. The action space is defined as 14 optional values so as to adjust the window accurately and find the optimal adjusting scheme:

$$cwnd = cwnd + \frac{x}{cwnd}, \tag{7}$$

where $x \in [-3, 10]$ & $x \in \mathbb{Z}$. More optional actions are provided for agents to make congestion control more flexible. The asymmetric interval is designed according to facts that the sender needs to increase the sending window in a larger-scale so as to detect the available bandwidth, especially at the early stage of sending packets, and reduce the window by a small margin to avoid congestion through its prediction. We define the interval boundary to -3 and 10 for some observations in experiments, that actions beyond this range will not improve the performance, but lead to greater storage spaces and longer convergence time. The value of x is updated every time interval and take effect in the next interval, while the cwnd is adjusted every ACK according to Eq. (7).

3) Reward Function: Reward function design mainly considers the optimization goal and the ability to judge whether the action is worth rewarding. In general, DECC's optimization goal is to achieve the maximum throughput, the minimum delay. In addition, to reduce the queuing delay and overflow, buffer occupancy at the bottleneck switches is expected to be the lowest. Hence we add an important optimization goal, congestion probability (buffer occupancy) attained by ECN. Finally the objective function is designed as:

$$U = \log\left(\frac{T_p}{B}\right) - \sigma_1 \log d + \sigma_2 \log(1 - F), \tag{8}$$

where T_p is the current throughput, B is the bottleneck bandwidth (for a real DCN, the bottleneck bandwidth can be detected in advance), d is the network delay defined as $d = RTT - RTT_{min}$, and $F \in [0, 1]$ refers to the proportion of ECN marked packets received in a cwnd. The $\log$ function ensures that the network can converge to a proportional fairness allocation of bandwidth resources when multiple users compete for the same bottleneck link. When no congestion occurs, the switch has no or slight queuing and the packet marking probability $F = 0$, so the function is revised to maximizing $\log(1-F)$. σ_1 and σ_2 represent weights of each objective in the multi-objective function, which are tunable. When σ_1 and σ_2 are adjusted to adapt to service requirements, the model needs no retrain but only requires a period of time to converge to the new strategy. Note that when there is no congestion, the parameter F makes no difference as no ECN marks are detected, hence throughput and delay are predominant to control the learning direction, which ensures that the flow can occupy the bandwidth to the greatest extent; When ECN marks are detected, F begins to affect the utility, and the sender adjusts cwnd in advance to restrict queues at switches. Therefore, the objective function makes DECC have better capability to deal with burst traffics.

U is the true objective that DECC attempts to optimize, but it cannot be set as the reward function directly. Reward function indicates the effectiveness of the action, as the agent tends to choose the action with the largest cumulative reward. If the action taken is worth advocating, it gets a positive reward to increase the probability of being taken next time, otherwise gets a negative value as punishment to avoid being taken next time. However, if the agent chooses a wrong action, the value of U calculated by the sender in a short time may still be positive, which means that this action will be advocated. Then, U can not be used as the reward function since the network may be trained to an unsatisfactory direction. To correctly demonstrate the effectiveness of the action, DECC uses the difference of U to define the reward, as an increase of the objective function value definitely indicates an improvement and the corresponding action should be encouraged, regardless of the original value of U.

Hence reward function r is given as:

$$r = \begin{cases} a & \Delta U > \epsilon_r, \\ 0 & -\epsilon_r \leq \Delta U \leq \epsilon_r, \\ b & \Delta U < -\epsilon_r, \end{cases} \tag{9}$$

where ΔU refers to the difference of the objective function values, a is a positive constant meaning encouragement, b is a negative constant meaning punishment, and ϵ_r is the objective function error. ϵ_r can be adjusted in implementation to control systematic errors and computing errors. Specific values of these parameters are tunable in the experiments to achieve better performance.

4) Training Algorithm: For efficient generalization and prediction ability, we use DQN as the training algorithm. DQN is a combination of deep learning and Q-learning, which turns the Q table into a Deep Neural Network (DNN). DQN consists of two networks, the main network and target network. The input of the main network is the state $\mathbf{s}_t$ at time step t, and the output is the value $Q(\mathbf{s}_t, a_t; \theta)$ of each action a_t taken in state $\mathbf{s}_t$ with parameter θ.

At time step t ($t_{interval}$ is the time interval of each step), the agent decides an action through $\epsilon - Greedy$ algorithm, the network will enter next state $\mathbf{s}_{t+1}$, then reward r_t is fed back to the agent according to the difference of the objective function value. For every $t_{interval}$, it obtains a tuple $(\mathbf{s}_t, a_t, r_t, \mathbf{s}_{t+1})$ and cache the tuple in the replay memory D for experience replay. Then, the training approach sample a random minibatch of $\{(\mathbf{s}_j, a_j, r_j, \mathbf{s}_{j+1})\}$ from D, and calculate the loss function. The loss function of main network is defined as:

$$\begin{aligned} L(\theta) &= E\Big[\big(y_j - Q\big(\mathbf{s}_j, a_j; \theta\big)\big)^2\Big], \\ y_j &= r + \gamma \max_{a'} Q\big(\mathbf{s}_{j+1}, a'; \theta'\big), \end{aligned} \tag{10}$$

where θ and θ' respectively refer to parameters of the main network and target network. $L(\theta)$ indicates the difference between the theoretical maximum reward and the actual reward of action a_t in state $\mathbf{s}_t$. The learning goal is to minimize the difference.

The main network is updated every $t_{interval}$. To train the target network, supervised learning can be used to update the parameters until the network converges. Training data sets are randomly sampled from the experience pool to reduce correlation between samples, making the network easier to be trained. The target network is trained and updated in periodic intervals with tuples copied from the experience pool. More details are given in Algorithm 1.

Combination of online training and offline training: DECC first trains the DQN offline to make the parameters converge by simulating the real DCN scenarios, until the parameters converge (to a pre-trained model). After convergence, it applies the pre-trained model as the initial model for tests and experiments, and starts online training. The sender will train its local parameters online by using the realistic traffic to fit the actual flow size distribution, bandwidth and other changing factors. Online training is used to improve the model generalization and handle new network states.

Algorithm 1 Learning Algorithm of DECC

Initialize replay memory D and minibatch size N
Initialize state-action value function Q with random θ
Initialize target state-action value function Q' with $\theta' = \theta$
Initialize the time step $t \leftarrow 1$
while $t \leq T$ **do**
  The agent takes current network state $\mathbf{s}_t$ from the shared buffer at sender;
  Select a random action a_t with probability ϵ;
  Otherwise select $a_t = \max_a Q(\mathbf{s}_t, a; \theta)$;
  Execute action a_t and transfer to a new state $\mathbf{s}_{t+1}$ and get the reward with variation of throughput, round trip delay and switch buffer occupancy;
  Cache the tuple $(\mathbf{s}_t, a_t, r_t, \mathbf{s}_{t+1})$ in D;
  Sample random minibatch of $(\mathbf{s}_j, a_j, r_j, \mathbf{s}_{j+1})$ from D;
  Compute the target Q value:

$$y_j = r_j + \gamma \max_{a'} Q'(\mathbf{s}_{j+1}, a'; \theta');$$

  Compute the loss function:

$$L(\theta) = E\Big[(y_j - Q(\mathbf{s}_j, a_j; \theta))^2\Big];$$

  Update the parameters θ with a gradient descent:

$$\bigtriangledown_\theta L(\theta) = E\big[(y_j - Q(\mathbf{s}_j, a_j; \theta))\bigtriangledown_\theta Q(a_j, a_j; \theta)\big];$$

  Replace the target network parameters θ' with the main network parameters θ every periodic intervals;
  $t = t + 1$;
end while

V. Performance Evaluation

We implemented DECC in ns-3 [34] and evaluated its performance in DCNs. Our experiments are committed to highlighting that DECC can efficiently learn the optimal cwnd adjustment strategy, obtain low network delay by reducing the queue length at switches and high bandwidth utilization, better at handling burst traffic.

As most of congestion control algorithms in DCNs are based on ECN and AIMD rules, we select DCTCP as the baseline algorithm. Besides, to highlight the good performance of DECC, We add swift and QTCP as other two comparison algorithms in our experiments:

- *DCTCP:* the most commonly used congestion control algorithm based on ECN feedback in DCNs.
- *Swift:* Swift uses an end-to-end RTT measurements with special NIC timestamps to modulate a congestion window in packets, aiming at maintaining the delay around a target value.
- *DECC:* the AIMD algorithm we designed combining DRL and accurate feedback from intermediate nodes.
- *QTCP:* a smart algorithm designed for traditional networks under dynamic network bandwidth, based on TCP and the generalization-based Kanerva coding.

Our experiments are divided into three parts. First, we evaluate the throughput, network delay, switch queue length and convergence time with four algorithms under different

TABLE II
DRL ALGORITHM PARAMETERS SETTING

Parameters	Value
Weight σ_1	0.5
Weight σ_2	0.5
ϵ_r in reward function	1
Positive reward a	5
Negative reward b	-5
Discount factor γ	0.95
$Time_{interval}$	0.01seconds
Learning rate	0.95
Exploration rate ϵ	Initially 1.0, discounted by 0.999
Replay buffer size	10000
Minibatch size	32

TABLE III
NETWORK PARAMETERS SETTING

Parameters	Value
Bottleneck bandwidth	50Mbps
Initial window	10 segments
Data segmentsize	1448 bytes
SndBufSize/RcvBufSize	8192000 bytes
ECN threshold	variable
Enable Pacing	true
RTT	variable

topologies and network bottlenecks. Second, we verify DECC's performance under incast traffic through variable numbers of synchronous flows. Finally, we evaluate the burst tolerance of DECC with distributed queries.

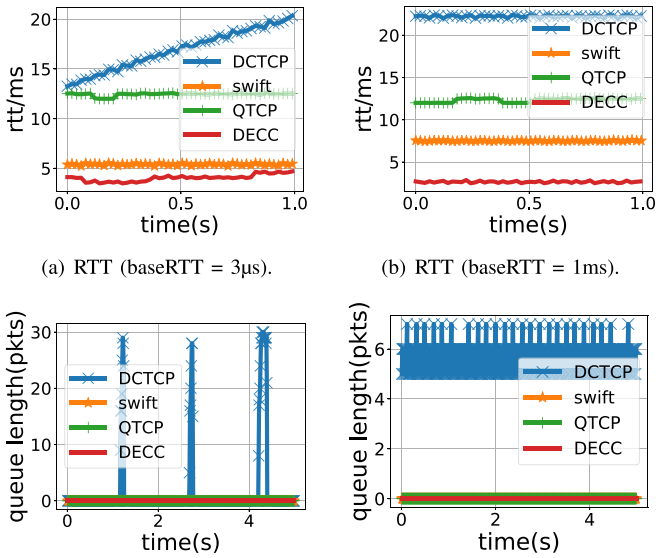
DRL algorithm parameters setting: TABLE II shows detailed parameter values of the DRL algorithm in simulation experiments. The reward is updated in every $time_{interval}$ and can be adjusted to get accurate measurements and appropriate convex condition. ϵ in $\epsilon - Greedy$ algorithm is initialized as 1.0, discounted by 0.999 per time step and no less than 0.01. This parameters vary in different topologies and are detailed in specific experiments.

A. Performance

1) Single-Hop Network Scenario: we use machines connected to one switch with 1Gbps links as Fig. 2 shows without background traffic. One host is a receiver; the others are senders. The senders establish long-lived connections to the receiver and send data as fast as they can. We repeat the experiment for DCTCP, swift, DECC and QTCP.

Simulation parameters: The available bandwidth of the network simulated by ns-3 is 1Gbps; the bottleneck bandwidth is 50Mbps; and the baseRTT (RTT without the queuing delay) is set to $3\mu s$ and 1ms respectively. During the transmission, we sample the instantaneous queue length at switch ports every 100ms. Parameters are summarized in TABLE III.

Considering one single flow: Fig. 4 shows the performance of DECC online. Fig. 4(c) and Fig. 4(d) show the real-time tracking of the queue length at switches. Setting baseRTT as 1ms, the queue length of DCTCP flows fluctuates around 6



(a) RTT (baseRTT = 3μs). (b) RTT (baseRTT = 1ms).

(c) Queue length (baseRTT = 3μs). (d) Queue length (baseRTT = 1ms).

Fig. 4. Comparison of RTT and queue length under a single-hop network.

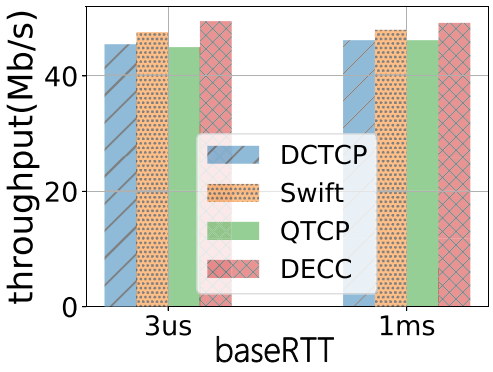


Fig. 5. Throughput comparisons under a single-hop network.

packets, showing that there are slight queuing on the switch, but no congestion. Similarly, the same experiments are carried out when baseRTT is 1ms, DCTCP flows have a short queue accumulation, reaching about 30 packets, resulting in short-term congestion, but quickly adjust and get out of congestion according to ECN feedback. Meanwhile, DECC can achieve zero queue length at the switch as well as swift and QTCP. Fig. 4(a) and Fig. 4(b) shows the variation of network delay with sending packets. The results show that DECC achieves the lowest network delay.

Obviously the switch queue with DECC does not exceeds the threshold and no ECN marks are detected, so the algorithm depends on the variation of throughput and delays to learn optimal cwnd adjusting policy. Fig. 5 shows the overall throughput measured after the network parameters basically converge. When the bottleneck bandwidth is 50Mbps, DECC achieves more than 49Mbps, highest in four algorithms. In general, DECC achieves higher bandwidth occupation and lower network delay.

Considering multiple flows: There are five senders and one receiver, and five TCP flows compete for 50Mbps bottleneck

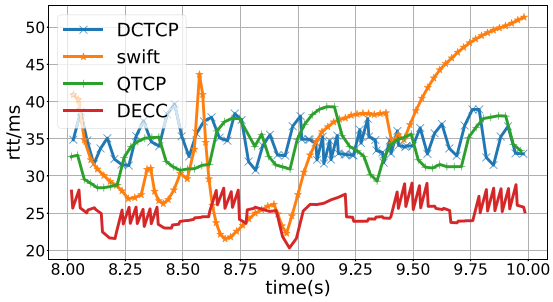


Fig. 6. Comparison of real-time RTT with multiple flows (baseRTT = 1ms).

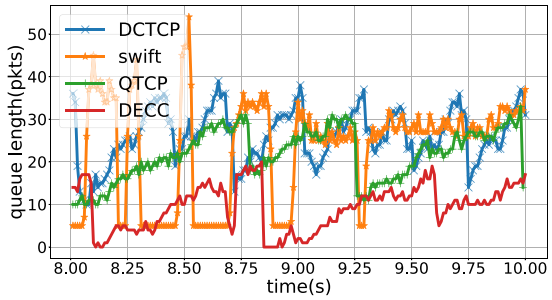


Fig. 7. Comparison of real-time queue length with multiple flows (baseRTT = 1ms).

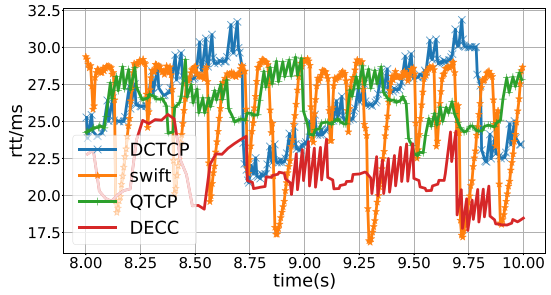


Fig. 8. Comparison of real-time RTT with multiple flows (baseRTT = 3 µs).

TABLE IV
COMPARISON OF THE FAIRNESS AND BANDWIDTH

Alg	DCTCP	swift	QTCP	DECC
Fairness	0.995	0.892	0.991	0.999
BW/Mbps	45.200	47.789	46.202	49.202

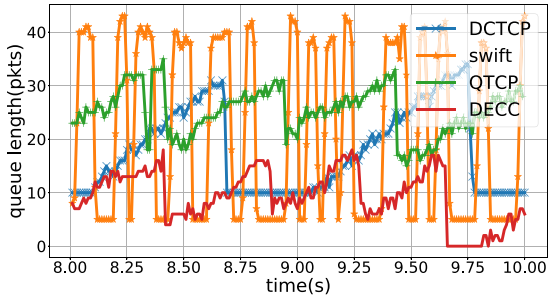


Fig. 9. Comparison of real-time queue length with multiple flows (baseRTT = 3 µs).

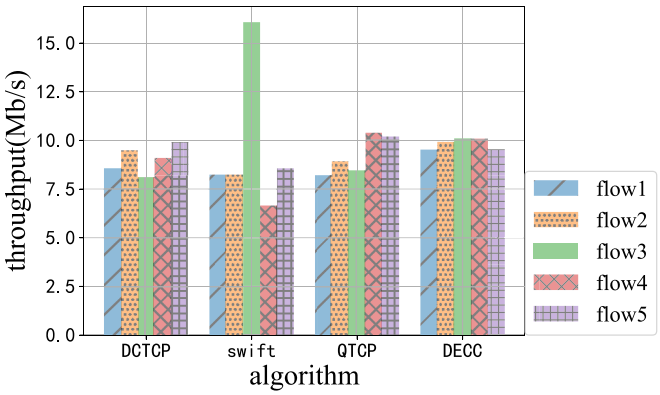


Fig. 10. Comparison of average throughput of five flows with different algorithms.

bandwidth. Fig. 7 shows the switch queue length, it can be seen that when five flows compete for the same bottleneck bandwidth, the four algorithms all suffer from the congestion. The DCTCP flows queue seriously at the switch, with the queue length fluctuating between 20 and 40 packets, and there are always marked packets in a cwnd. Swift adjusts the window according to the change of RTT, resulting in unstable queue length and large fluctuation.

Due to the lack of accurate congestion feedback, QTCP also has large queue fluctuations and has been in a congested condition. However, DECC controls the queue length to be less than 20 packets. Once the queue length exceeds 17 packets there will be marked packets by ECN. As can be seen from the result, DECC can quickly react when the ECN mark is generated, to make the queue length fall below 17 in a short time. Fig. 6 shows the same flow's RTT variation in 2 seconds. DECC achieves the lowest round-trip latency and highest stability due to well-limited switch queues. Fig. 8 and Fig. 9 are delay and queue length variation while baseRTT is setting as 3 µs. DECC still performs the best, and as the shorter RTT brings faster information updates, DECC can limit the queue to a lower value, while the queue of swift flows still fluctuates greatly, frequently overflowing and packet loss, which result in timeouts and large delay fluctuations.

Fairness: Fig. 10 shows comparisons of respective throughput of five flows with the four schemes, and TABLE IV lists the concrete fairness and bandwidth occupancy. The fairness is calculated by Jain's fairness [35]. From TABLE IV, even if five flows compete for the bottleneck, DECC has higher bandwidth utilization close to the full bandwidth and shows great fairness, no less than DCTCP, while swift is very terrible. Hence DECC has good inter-flow fairness.

Learning curve and convergence time: Fig. 11 shows the learning curves of offline training. The throughput is low before convergence and fluctuates greatly. With training and learning, parameters gradually converge and the throughput tends to be stable and close to full bandwidth. As DECC flows make decisions to select an optimal action and update the reward to optimize DQN parameters every $t_{interval}$, $t_{interval}$ significantly influences the convergence time as well as learning efficiency. To explore the sensitivity of DECC to decision time interval and determine an efficient value of $t_{interval}$ in experiments, we respectively set $t_{interval}$ as 0.01s, 0.05s

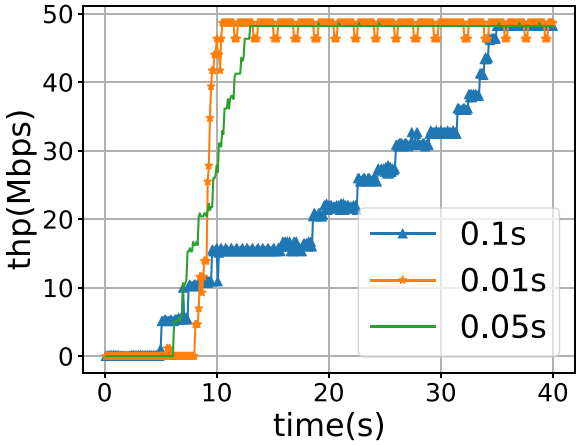


Fig. 11. Learning curves with different decision time intervals.

and 0.1s to observe their learning curves. In addition, we set the RTT as 1ms in single-hop networks. Intuitively, with the decision interval increasing, the convergence becomes slower. When $t_{interval}$ is set as 0.01s (10 RTTs), the throughput converges around 10s but fluctuates violently around 48Mbps. When $t_{interval}$ is set as 0.1s, the agent achieves high and steady throughput though needs around 35s to converge. Shorter time intervals and more frequent decision making allow agents to observe more information and learn faster, but also more sensitive to the variation of network states, leading to fluctuation.

DECC performs better when the interval is 0.05s, which can achieve relatively fast convergence time as well as stability.

Complexity and deployability: DECC compresses the number of states and actions of utilizing the average values of measurements (cwnd, throughput, delay, ECN and remaining flow size) to make decision in each time interval, instead of keeping all the measurements during the time. And DECC conservatively adopts a three-layer fully connected neural network for training, with 5 input nodes for states, 14 output nodes for actions and tunable node number for the hidden layer, which is easy to converge. Such design reduces the storage space and computational complexity of DECC, meanwhile ensuring the effectiveness of learning [28]. Moreover, we set $t_{interval}$ as 0.01s, whose resource consumption is acceptable for servers in DCNs which consists of multiple CPUs and high-bandwidth bus [12]. In addition, DECC is trained with online as well as offline training. DECC first learns from the historical data of the DCN offline to make the model converge at some general network scenarios, then adjusts the model online in the real network, which greatly improves the online training efficiency and reduce the convergence time.

2) Multi-Hop network Scenario: Fig. 12 shows the multi-bottleneck network topology in the experiment. There are a total of 40 flows competing for a 10Gbps bottleneck and 30 flows competing for a 1Gbps bottleneck.

Simulation parameters: available bandwidth of the DCN is 1Gbps. The bottleneck bandwidth T1-T2 is 10Gbps and T2-R1 is 1Gbps. The initial window is set as 10 data segments and each data segment is 1448bytes. Middle switches are configured with ECN. Switch T1's queue threshold is set

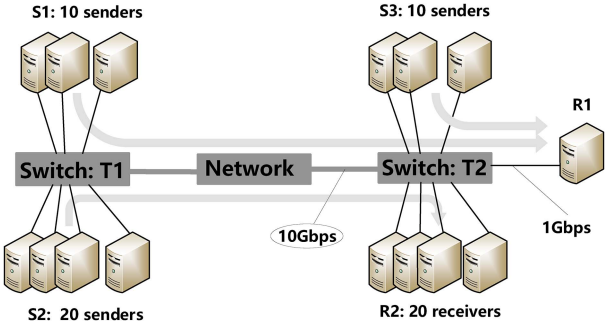


Fig. 12. A multi-hop network topology in the experiment.

TABLE V
COMPARISON OF THE AVERAGE FLOW THROUGHPUT OF THREE PATHS AND BOTTLENECK BANDWIDTH

BW/Mbps	S1-R1	S2-R2	S3-R1
DCTCP	22.249	465.368	73.072
swift	39.630	462.451	56.782
QTCP	21.374	464.262	72.115
DECC	24.263	473.952	74.960

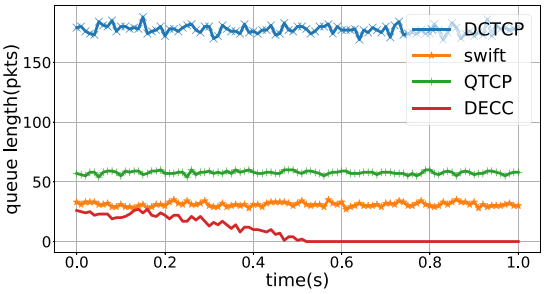


Fig. 13. Queue length of t1.

to 20-60 data packets and T2's queue threshold is set to 50-150 data packets. The RTT is set to $10\mu s$.

In this scenario, link T1-T2 and T2-R1 are two bottleneck links, so there are queues built up on switch T1 and T2. TABLE V shows comparisons of the average flow throughput of three paths S1-R1, S2-R2 and S3-R1. It can be seen that with DECC the average throughput under the three paths is higher. Moreover, according to TABLE V, with DCTCP, the total bandwidth of bottleneck T1-T2 is 9.5296Gbps and T2-R1 is 0.9532Gbps while with DECC, the total bandwidth of bottleneck T1-T2 is 9.7217Gbps and T2-R1 is 0.9922Gbps. Hence, bandwidth occupancy of DECC is higher than DCTCP. Fig. 13 and Fig. 14 show comparisons of the queue length at switches T1 and T2 with four congestion control algorithms tested during 1s after convergence.

The queue length of DECC is much shorter than DCTCP, almost cut by 70% and even more, which significantly reduces the queuing delay and relieves the switch pressure in faced with burst traffic. At switch T1, DECC limits the queue length below the minimum threshold of ECN, and can quickly restore queues after a slight congestion occurs, making queues shorter and shorter by learning from the network information. The queue with DCTCP always exceeds the maximum threshold of

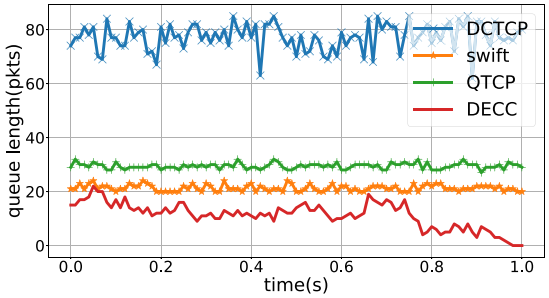


Fig. 14. Queue length of t2.

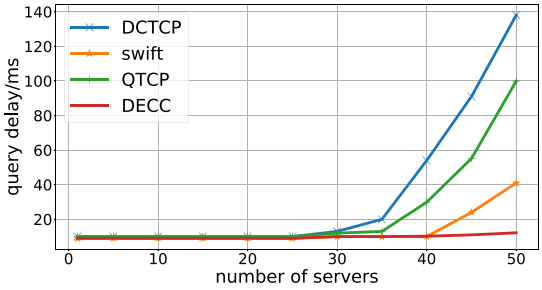


Fig. 15. Query delay comparison.

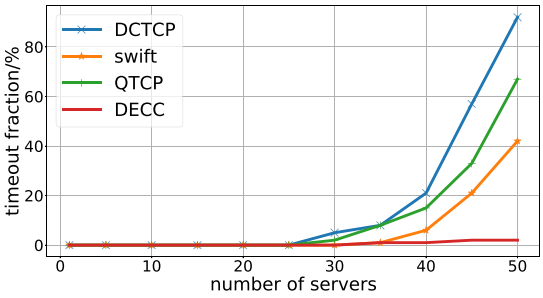


Fig. 16. Timeout fraction comparison.

ECN, which makes the network congested all the time, and the queue length fluctuates greatly. Swift and QTCP flows are also slightly congested, and have high queue occupancy. Moreover, at switch T2, the queue length of DECC becomes close to zero with time while DCTCP's queue fluctuates greatly in a high value. Since DECC is designed to minimize buffer occupancy, the queue of DECC becomes shorter with the agent learning.

It can be seen that in the complex multi-bottleneck network, especially when the number of concurrent flows is large, DECC has more obvious advantages, which can significantly reduce the switch buffer occupancy while maintaining high bandwidth.

B. Solving the Incast Issue

We compare the ability of DECC and the other three algorithms in dealing with the incast problem in the scenario with burst synchronous flows.

We implement an experiment to repeat the network conditions in [6]. Many-one machines are connected to a switch with 1Gbps links. One machine acts as a client, others act as servers. The client requests ("queries") $1MB/N$ bytes from N different servers, and each server responds immediately with the requested amount of data. The client waits until it receives all the responses, and then issues another similar query, repeatedly. We set the baseRTT as 10ms. In our experiments, the minimum query completion time is around 8ms. The incoming link at the receiver is the bottleneck, and it takes 8ms to deliver 1MB of data over a 1Gbps link. We carry out experiments for DCTCP, swift, QTCP and DECC. Fig. 15 shows the average query delay. When the number of synchronous flows is no more than 25, all algorithms perform well with around 10ms query delay. But when the number increases, DCTCP suffers from the incast problem and the query delay is largely extended. When there are 50 synchronous flows, the query delay with DCTCP even reaches more than 150ms, which is deadly in high-speed DCNs. QTCP and swift flows also perform poorly as the number of simultaneous flows increases. DECC has always controlled the query delay to around 10ms as the incast degree increases.

Fig. 16 shows the fraction of queries that suffered at least one timeout. When the number of servers gradually increases (to about 30), DCTCP begins to suffer timeouts slightly. When the number of servers reaches 35 or 40, DCTCP suffers more serious timeouts, while the performance of DECC always remains good. It can be seen DECC handles synchronous short flows well after convergence, and can avoid packet loss by limiting buffer occupancy, thereby avoiding request timeouts.

C. Burst Tolerance

Modern DCNs carry many types of services, with long and short flows mixed. There are many background flows, so switch buffers may be occupied at all times, which make higher demands for the burst tolerance of congestion control algorithms. In previous experiments, we confirmed that the queue length of DECC is shorter than that of DCTCP when sending data packets continuously. If the burst traffic arrives when the switch is highly occupied, the available space of the switch will be too small to avoid the overflow. Theoretically, it shows that DECC has better burst tolerance than the traditional algorithm by limiting queue length. Also, DECC has more advantages than QTCP with accurate congestion feedback.

To evaluate this, we connected 59 hosts to a switch with 1Gbps links. 26 of these hosts participate in a $25-1$ incast pattern, with 1 host acting as a client, and 25 hosts acting as servers. The client requested a total of 1MB data from the servers (40KB from each). In Fig. 12, the switch can easily handle 25:1 incast with DCTCP and DECC, without inducing any timeouts. Next, like the simulation in the motivation, we use the remaining 33 hosts to start "background" traffic of long-lived flows to consume the shared buffer. All background flows also use the corresponding congestion control algorithms. TABLE VI shows the query delay.

We see that without background traffic, DCTCP, QTCP and swift all achieve query delay not exceeding 10ms. Judging from the performance of swift in the previous experiment, swift has violent queue length fluctuations, which will lead to the timeout of data packets in a short period of time and extends query delay. But in presence of long-lived flows that occupy the buffer as the background traffic, DCTCP flows are

TABLE VI
THE COMPARISON OF THE QUERY DELAY

	without background traffic	with background traffic
DCTCP	9.26ms	37.34ms
swift	13.11ms	29.57ms
QTCP	10.31ms	41.76ms
DECC	9.02ms	9.19ms

subjected to timeouts and QTCP flows perform even worse, while DECC's performance is almost unchanged.

If there is no background flows, the buffer is empty and has high burst tolerance. When there are background flows, the buffer is already occupied by long and stable background flows, which may easily cause sudden short flows losing packets, as they cannot compete with long flows for the buffer. When there is a burst of traffic, the DECC flow makes an accurate and optimal response to the network congestion, quickly limits the queue length, which reduce the packet loss rate of short flows and make the query delay reasonable.

VI. CONCLUSION

In this paper, we proposed a DRL-based ECN-combined congestion control algorithm DECC for DCNs, to precisely perceive the congestion degree at the bottleneck link, and learn cwnd adjusting strategies proactively. We designed the objective function with ECN to reduce the queue length at bottleneck switches, and took a fine-grained action space to improve the throughput. Through simulation experiments, it is confirmed that DECC achieves higher bandwidth occupancy, lower queuing delay, and lower query delay under burst synchronous traffic. Accurate buffer occupancy information at bottleneck switches and appropriate DRL algorithm design are efficient for the congestion control in DCNs.

REFERENCES

[1] W. Xia, P. Zhao, Y. Wen, and H. Xie, "A survey on data Center networking (DCN): Infrastructure and operations," *IEEE Commun. Surveys Tuts.*, vol. 19, no. 1, pp. 640–656, 1st Quart., 2017.

[2] Y. Chen, R. Griffith, J. Liu, R. H. Katz, and A. D. Joseph, "Understanding TCP Incast throughput collapse in datacenter networks," in *Proc. 1st ACM Workshop Res. Enterprise Netw.*, 2009, pp. 73–82.

[3] A. M. Abdelmoniem and B. Bensaou, "T-RACKs: A faster recovery mechanism for TCP in data center networks," *IEEE/ACM Trans. Netw.*, vol. 29, no. 3, pp. 1074–1087, Jun. 2021.

[4] W. Chen, F. Ren, J. Xie, C. Lin, K. Yin, and F. Baker, "Comprehensive understanding of TCP Incast problem," in *Proc. IEEE Int. Conf. Comput. Commun. (INFOCOM)*, 2015, pp. 1688–1696.

[5] S. Zou, J. Huang, J. Wang, and T. He, "Flow-aware adaptive pacing to mitigate TCP Incast in data Center networks," *IEEE/ACM Trans. Netw.*, vol. 29, no. 1, pp. 134–147, Feb. 2021.

[6] M. Alizadeh et al., "Data center TCP (DCTCP)," in *Proc. ACM SIGCOMM Conf. Appl. Technol. Archit. Protocols Comput. Commun. (SIGCOMM)*, 2010, pp. 63–74.

[7] A. Munir et al., "Minimizing flow completion times in data centers," in *Proc. IEEE Int. Conf. Comput. Commun. (INFOCOM)*, 2013, pp. 2157–2165.

[8] B. Vamanan, J. Hasan, and T. N. Vijaykumar, "Deadline-aware datacenter TCP (D2TCP)," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 42, no. 4, pp. 115–126, 2012.

[9] W. Li, F. Zhou, K. R. Chowdhury, and W. Meleis, "QTCP: Adaptive congestion control with reinforcement learning," *IEEE Trans. Netw. Sci. Eng.*, vol. 6, no. 3, pp. 445–458, Jul.–Sep. 2019.

[10] X. Nie et al., "Dynamic TCP initial windows and congestion control schemes through reinforcement learning," *IEEE J. Sel. Areas Commun.*, vol. 37, no. 6, pp. 1231–1247, Jun. 2019.

[11] L. Chen, J. Lingys, K. Chen, and F. Liu, "AuTO: Scaling deep reinforcement learning for datacenter-scale automatic traffic optimization," in *Proc. ACM SIGCOMM Conf. Appl. Technol. Archit. Protocols Comput. Commun. (SIGCOMM)*, 2018, pp. 191–205.

[12] S. Yan, X. Wang, X. Zheng, Y. Xia, D. Liu, and W. Deng, "ACC: Automatic ECN tuning for high-speed datacenter networks," in *Proc. ACM SIGCOMM Conf. Appl. Technol. Archit. Protocols Comput. Commun. (SIGCOMM)*, 2021, pp. 384–397.

[13] K. Winstein and H. Balakrishnan, "TCP ex machina: Computer-generated congestion control," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 43, no. 4, pp. 123–134, 2013.

[14] F. Y. Yan et al., "Pantheon: The training ground for Internet congestion-control research," in *Proc. USENIX Annu. Tech. Conf. (ATC)*, 2018, pp. 731–743.

[15] N. Jay, N. H. Rotman, B. Godfrey, M. Schapira, and A. Tamar, "A deep reinforcement learning perspective on Internet congestion control," in *Proc. Int. Conf. Mach. Learn. (ICML)*, 2019, pp. 3050–3059.

[16] N. Mckeown et al., "pFabric: Minimal near-optimal datacenter transport," *ACM Sigcomm Comput. Commun. Rev.*, vol. 43, no. 4, pp. 435–446, 2013.

[17] L. Chen, K. Chen, W. Bai, and M. Alizadeh, "Scheduling mix-flows in commodity datacenters with Karuna," in *Proc. ACM SIGCOMM Conf. Appl. Technol. Archit. Protocols Comput. Commun. (SIGCOMM)*, 2016, pp. 174–187.

[18] W. Bai, L. Chen, K. Chen, D. Han, C. Tian, and H. Wang, "PIAS: Practical information-agnostic flow scheduling for commodity data centers," *IEEE/ACM Trans. Netw.*, vol. 25, no. 4, pp. 1954–1967, Aug. 2017.

[19] P. X. Gao, A. Narayan, G. Kumar, R. Agarwal, S. Ratnasamy, and S. Shenker, "pHost: Distributed near-optimal datacenter transport over commodity network fabric," in *Proc. 11th ACM Conf. Emerg. Netw. Exp. Technol. (CoNEXT)*, 2015, pp. 1–12.

[20] I. Cho, K. Jang, and D. Han, "Credit-scheduled delay-bounded congestion control for datacenters," in *Proc. ACM SIGCOMM Conf. Appl. Technol. Archit. Protocols Comput. Commun. (SIGCOMM)*, 2017, pp. 239–252.

[21] M. Handley et al., "Re-architecting datacenter networks and stacks for low latency and high performance," in *Proc. ACM SIGCOMM Conf. Appl. Technol. Archit. Protocols Comput. Commun. (SIGCOMM)*, 2017, pp. 29–42.

[22] B. Montazeri, Y. Li, M. Alizadeh, and J. K. Ousterhout, "Homa: A receiver-driven low-latency transport protocol using network priorities," in *Proc. ACM SIGCOMM Conf. Appl. Technol. Archit. Protocols Comput. Commun. (SIGCOMM)*, 2018, pp. 221–235.

[23] S. Hu et al., "Aeolus: A building block for proactive transport in datacenters," in *Proc. ACM SIGCOMM Conf. Appl. Technol. Archit. Protocols Comput. Commun. (SIGCOMM)*, 2020, pp. 422–434.

[24] Y. Zhu et al., "Congestion control for large-scale RDMA deployments," in *Proc. ACM SIGCOMM Conf. Appl. Technol. Archit. Protocols Comput. Commun. (SIGCOMM)*, 2015, pp. 523–536.

[25] Y. Li et al., "HPCC: High precision congestion control," in *Proc. ACM SIGCOMM Conf. Appl. Technol. Archit. Protocols Comput. Commun. (SIGCOMM)*, 2019, pp. 44–58.

[26] R. Mittal et al., "TIMELY: RTT-based congestion control for the datacenter," in *Proc. ACM SIGCOMM Conf. Appl. Technol. Archit. Protocols Comput. Commun. (SIGCOMM)*, 2015, pp. 537–550.

[27] G. Kumar et al., "Swift: Delay is simple and effective for congestion control in the datacenter," in *Proc. ACM SIGCOMM Conf. Appl. Technol. Archit. Protocols Comput. Commun. (SIGCOMM)*, 2020, pp. 514–528.

[28] Y. Kong, H. Zang, and X. Ma, "Improving TCP congestion control with machine intelligence," in *Proc. Workshop Netw. Meets AI (ML NetAI)*, 2018, pp. 60–66.

[29] A. Yu, H. Yang, K. K. Nguyen, J. Zhang, and M. Cheriet, "Burst traffic scheduling for hybrid E/O switching DCN: An error feedback spiking neural network approach," *IEEE Trans. Netw. Service Manag.*, vol. 18, no. 1, pp. 882–893, Mar. 2021.

[30] Y. Jiang, M. S. Kodialam, T. V. Lakshman, S. Mukherjee, and L. Tassiulas, "Resource allocation in data centers using fast reinforcement learning algorithms," *IEEE Trans. Netw. Service Manag.*, vol. 18, no. 4, pp. 4576–4588, Dec. 2021.

[31] K. K. Ramakrishnan, S. Floyd, and D. L. Black, "The addition of explicit congestion notification (ECN) to IP," IETF, RFC 3168, 2001. Accessed: Jun. 20, 2022. [Online]. Available: https://www.rfc-editor.org/rfc/rfc3168.html

[32] V. A. Jain, T. R. Henderson, K. S. Shravya, and M. P. Tahiliani, "Data center TCP in NS-3: Implementation, validation and evaluation," in *Proc. Workshop NS-3*, 2020, pp. 65–72.
[33] V. Mnih et al., "Playing Atari with deep reinforcement learning," 2013, *arXiv:1312.5602*.
[34] H. Yin et al., "NS3-AI: Fostering artificial intelligence algorithms for networking research," in *Proc. Workshop NS-3*, 2020, pp. 57–64.
[35] R. Jain, D. Chiu, and W. Hawe, "A quantitative measure of fairness and discrimination for resource allocation in shared computer systems," 1998, *arXiv:cs/9809099*.

Yi Liu received the B.S. degree from the Department of Electronic Engineering and Information Science, USTC in July, 2021, where she is currently pursuing the graduate degree with the School of Cyber Science and Technology. Her research interests include data center network and transmission optimization.

Jiangping Han (Member, IEEE) received the B.S. and Ph.D. degrees from the Department of Electronic Engineering and Information Science, USTC in 2016 and 2021, respectively, where she is currently a Postdoctoral Fellow with the School of Cyber Science and Technology. From November 2019 to October 2021, she was a Visiting Scholar with the School of Computing, Informatics, and Decision Systems Engineering, Arizona State University. Her research interests include future Internet architecture design and transmission optimization.

Kaiping Xue (Senior Member, IEEE) received the bachelor's degree from the Department of Information Security, University of Science and Technology of China (USTC) in 2003, and the Ph.D. degree from the Department of Electronic Engineering and Information Science, USTC in 2007. From May 2012 to May 2013, he was a Postdoctoral Researcher with the Department of Electrical and Computer Engineering, University of Florida. He is currently a Professor with the School of Cyber Science and Technology, USTC. His research interests include next-generation Internet architecture design, transmission optimization, and network security. He serves on the editorial board of several journals, including the IEEE TRANSACTIONS ON DEPENDABLE AND SECURE COMPUTING, the IEEE TRANSACTIONS ON WIRELESS COMMUNICATIONS, and the IEEE TRANSACTIONS ON NETWORK AND SERVICE MANAGEMENT. He has also served as a (Lead) Guest Editor for many reputed journals/magazines, including IEEE JOURNAL ON SELECTED AREAS IN COMMUNICATIONS, *IEEE Communications Magazine*, and IEEE NETWORK. He is an IET Fellow.

Jian Li (Member, IEEE) received the bachelor's degree from the Department of Electronics and Information Engineering, Anhui University in 2015, and the Ph.D. degree from the Department of Electronic Engineering and Information Science, University of Science and Technology of China (USTC) in 2020. From November 2019 to November 2020, he was a Visiting Scholar with the Department of Electronic and Computer Engineering, University of Florida. From December 2020 to December 2022, he was a Postdoctoral Researcher with the School of Cyber Science and Technology, USTC, where he is currently an Associate Researcher. His research interests include wireless networks, next-generation Internet, and quantum networks.

Qibin Sun (Fellow, IEEE) received the Ph.D. degree from the Department of Electronic Engineering and Information Science, University of Science and Technology of China in 1997, where he is currently a Professor with the School of Cyber Science and Technology. He has published more than 120 papers in international journals and conferences. His research interests include multimedia security, network intelligence, and security.

Jun Lu received the bachelor's degree from Southeast University in 1985, and the master's degree from the Department of Electronic Engineering and Information Science, University of Science and Technology of China in 1988, where he is currently a Professor. His research interests include theoretical research and system development in the field of integrated electronic information systems. He is an Academician of the Chinese Academy of Engineering.