An Investigation on Data Center Congestion Control Algorithms

by

Kunlin Zhang

A thesis submitted in partial fulfillment of the requirements for the degree of

Master of Science

in

Software Engineering and Intelligent Systems

Department of Electrical and Computer Engineering
University of Alberta

Abstract

With the rapid growth of data-intensive applications, congestion control algorithms for datacenter networks under RDMA over Converged Ethernet protocol have become vital in managing various traffic patterns that demand ultra-low latency and high end-to-end throughput. Although many rule-based and learning-based algorithms have been proposed to enhance throughput and reduce latency, challenges still remain in ensuring fair and efficient control under dynamic, bursty, and variable-sized flows which are prevalent in datacenter networks. In this thesis, we analyze the traffic control mechanism existing in modern data center networks and propose the Fair Datacenter Congestion Control (FDCC) algorithm based on deep reinforcement learning that leverages Long Short-Term Memory (LSTM) and historical memory augmentation to enable predictive control. FDCC operates from a window-based perspective, and introduces a staged reward design to effectively learn from network states based on in-band network telemetry acquired from various sources along the flow path to achieve fair congestion control under dynamic traffic. We perform extensive experiments under a variety of datacenter traffic patterns, including incast flows, long-short flows, and real-world traffic flows. The results suggest that our approach significantly enhances fairness for dynamic flow patterns over a range of state-of-the-art rule-based and learning-based congestion control algorithms, while maintaining comparable flow completion time and goodput.

Preface

A portion of the content presented in this thesis, including Chapter 2, 3, 4, 5, and 6 have been submitted to IEEE INFOCOM 2024 for review.

Acknowledgements

I am profoundly grateful to Dr. Di Niu and Dr. Zhan Shu for their invaluable guidance throughout my research journey. Their mentorship was instrumental in navigating the challenges I faced, and their depth of knowledge and unwavering patience illuminated my path, instilling in me the ideals of rigorous research.

My heartfelt appreciation extends to my family, my girlfriend Xi, and friends Jerry, Ruiqing, Ruichen, Haoran, Yiming, Jianming, Weidong, Yuxuan, and other esteemed labmates. Your support and encouragement have been a pillar of strength throughout my graduate studies.

Table of Contents

List of Tables

List of Figures

List of Symbols

Latin

A	Set of all possible actions
a	Action
p	State transition probability distribution function
Q	Q-value function
r	Step reward
S	Set of all environmental states

Greek

γ	Discounted factor
π	Policy
ρ	Reward function
τ	Trajectory
θ	Parameter of state-action value function

Abbreviations

ACK Acknowledgment.

BW Link Bandwidth.

CC Congestion Control.

DCN Data Center Network.

DCQCN Data Center Quantized Congestion Notification.

DCTCP Data Center TCP.

DDPG Deep Deterministic Policy Gradient.

DQN Deep Q Learning Network.

DRL Deep Reinforcement Learning.

ECN Explicit Congestion Notification.

HPCC High Precision Congestion Control.

LSTM Long Short-Term memory.

NIC Network Interface Card.

PACC Proactive and Accurate Congestion Control.

PFC Priority Flow Control.

PPO Proximal Policy Optimization.

RCC Receiver-Driven RDMA Congestion Control.

RDMA Remote Direct Memory Access.

RL Reinforcement Learning.

RoCE RDMA over Converged Ethernet.

TD3 Twin Delayed DDPG.

TRPO Trust Region Policy Optimization.

w.r.t With reference to.

Chapter 1

Introduction

1.1 Problem Motivation

Serving as the backbone for large-scale data-intensive applications [1], data centers have adapted to the increasing demand for high throughput and ultra-low latency by continually ramping up their link speeds from 10 Gbps to well over 400 Gbps. Given the extremely high data rates, traditional transmission protocols like TCP/IP, which operate on the system's network stack, are now impractical for deployment since software-based data packet processing can lead to significant overhead [2]. To alleviate the intensive kernel load and conserve CPU cycles, modern data centers have adopted a bypassing technology known as Remote Direct Memory Access (RDMA). This technique enables direct memory access between the memory of one computer and another without involving the operating system. Moreover, to meet the requirements of high throughput and ultra-low latency, the RDMA domain necessitates a lossless and reliable network to mitigate and manage potential packet loss and network congestion.

The original solutions are two simple low-level mechanisms named Priority Flow Control (PFC) and Go-Back-N (GBN) [2], which were retained for port-level control at the lower network layer. However, PFC may lead to potential problems such as Head-of-Line blocking [3] or a PAUSE frame storm [4] since it broadcasts PAUSE packets back to all flow sources from a congested port, which is likely to degrade the

performance across the entire RDMA domain. To effectively prevent the activation of PFC, designing more concise and specialized congestion control (CC) mechanisms for RDMA-enabled data center networks (DCNs) has become an urgent need and motivated many research efforts in recent years.

Existing datacenter congestion control algorithms can be classified into three types depending on the control-driven point (i.e., host, switch, or receiver) in a datacenter network. Host-driven approaches, such as DCTCP[5], TIMELY[6], DCQCN[7], and HPCC[8], employ information carried in acknowledgment packets to trigger congestion control actions. This type of control is resource-saving and has already been deployed in real-world data centers. In addition, recently proposed learning-based congestion control methods such as RL-CC[9], DeepCC[10], and Pareto[11] are also host-based approaches. Switch-driven approaches, exemplified by PACC[12], aim to reduce the delay in the control loop by directly generating explicit control actions from switches to hosts. Receiver-driven approaches, e.g., RCC [13], operate on the assumption that most congestion occurs in the ToR downlink and estimate the number of flows transmitted to the receiver to regulate traffic. This type of approach enables one-step control in certain traffic scenarios and requires switching between the last-hop phase and the in-network congestion phase during the flow's life cycle. Although these prior studies have shown great promise under multiple traffic scenarios, challenges still remain in achieving fair and fast-responding congestion control, especially under incast, bursty and dynamic traffic flows which are prevalent in data centers.

1.2 Our Contribution

We propose the Fair Datacenter Congestion Control (FDCC) algorithm that learns to make predictive and fair congestion control decisions by fully utilizing historical information and network states along the flow path based on deep reinforcement

learning. In designing FDCC, our contributions can be summarized as follows:

- We propose a new DRL algorithm named FDCC to solve the partially observable Markov decision process in data center networks by leveraging LSTM and augmented memory to learn the traffic pattern to proactively and predictively control the congestion.

- We introduce a window-based control action, which achieves faster restrain upon congestion compared with the rate-based method, and multi-source in-band network telemetry to offer more effective control on congestion inspired by HPCC[8].

- We design a two-stage reward that takes advantage of the heuristic rule for the queue-draining stage to control congestion and encourages fairness and bandwidth utilization during the bandwidth-probing phase. We further employ curriculum learning to enable our agent to generalize from a small training traffic workload to other large-scale scenarios.

We conduct extensive experiments on NS-3 simulator[14] to evaluate the FDCC algorithm under various dynamic datacenter traffic types, including incast flows, long-short flows, and real-world traffic workloads (*Web_Search*[5] and *FB_Hadoop*[15]). The results suggest that FDCC outperforms a wide range of rule-based and learning-based congestion control algorithms in terms of fairness under multiple traffic patterns while maintaining comparable performance with these algorithms in terms of flow completion time and goodput.

1.3 Thesis Outline

The content of this thesis is organized in the following manner. Chapter 2 is a survey of related works. Chapter 3 provides a comprehensive background introduction and the main challenges of designing congestion control algorithms for data center

networks. Chapter 4 explains the choices of states, action and reward from a reinforcement learning perspective, the neural network architecture of our approach, and the training procedure. Chapter 5 details our experiment setup, benchmark, and test metrics. Chapter 6 presents the experimental results of FDCC(ours) with a range of state-of-the-art congestion control algorithms. Chapter 7 contains the conclusion and discusses potential future works.

Chapter 2

Related Work

2.1 Data Center Traffic Control in General

RDMA over Converged Ethernet (RoCEv2) relies on a lossless network guaranteed by two main mechanisms: congestion control part and loss recovery part. in this chapter, we briefly summarize the related works based on these two aspects over the past decades.

2.1.1 Loss Recovery Algorithms

Host-driven loss recovery. Recovering packet loss based on UNACK packets and retransmitting the packets in correct sequences have been a conventional rule since the TCP design. BGN[2] is a simple and straightforward way. It only needs to check the sequence number of the next frame it expects, and if it is not correct, the receiver will discard the following packets. IRN[16] maintains a bitmap to track which packets have been cumulatively and selectively acknowledged. When in the loss recovery mode, the sender selectively retransmits lost packets as indicated by the bitmap, instead of sending new packets, in such a way, that it can avoid duplicated retransmission. MELO[17] utilizes off-chip memory to store and re-order out-of-order packets and leverages an on-chip linked list to manage the bitmap to retransmit packets.

Switch-driven loss recovery. Switch-assisted loss recovery can benefit timeout avoidance. CutPayload[18] allows switches to send the notification to the destination

and echos it back to the source when packet drops occur. This echo-back process may aggregate congestion. On the contrary FastLane[19] directly sends back the notification to the source, which is similar to lightning[20], however, the latter one does not need the entire network (switches and NICs) to be upgraded at once, and can be deployed as a building block on programmable switches which is more convenient.

2.1.2 Congestion Control Algorithms

Host-driven Congestion Control. In these algorithms, the hosts primarily rely on the information conveyed directly by ACK packets or data that can be derived through basic mathematical operations. The main challenge for this type of method is to mitigate the side effects resulting from reliance solely on feedback from receivers. Factors such as delay and severe congestion can significantly compromise the effectiveness of host-based control methods. DCTCP[5] is the first congestion control for data center network, it utilizes ECN marking to adjust the rate within RTT. DCQCN[7] operates similarly to DCTCP, but incorporates ECN more accurately. TIMELY[6] and SWIFT[21] are both purely delay-based CCs. SWIFT mainly addresses the convergence issue observed in TIMELY. HPCC[8] leverages the in-band network telemetry (INT) collected from each hop to adjust both the rate and sending window. Despite their utility, these algorithms are encumbered by the long control loop problem and require further improvement in handling instantaneous bursty traffic adequately.

Switch-driven Congestion Control. Switch-based methods measure congestion and insert accurate control information into the packet header. Priority flow control (PFC) is the basic control mechanism in data center networks. Following a port-level dynamic threshold calculated by the usage of shared switch buffer size, PFC sends back PAUSE signal to sources to stop transmission until the congestion on this port is mitigated. Despite this approach being easy to deploy and almost resource-free, the performance degradation caused by head-of-line blocking, and PAUSE storm has

been a notorious issue. XCP[22] and RCP[23] adjust the window size and flow fairness proactively. TFC[24] use a token-based bandwidth allocation approach over a certain time interval. RoCC[25] proactively feedback fair rate based a queue-length-based PI on the switch. PACC[12] tracks the queue length based on a dynamic interval to distinguish bursty traffic and congestion and generate notifications directly from the switch. However, these methods might impose overhead on switch computational resources, as they require complex calculations and additional memory that most commodity switches lack.

Receiver-driven Congestion Control. Receiver-driven techniques are devised to alleviate incast traffic performance degradation, drawing on the conclusion that most congestion happens at the ToR downlink[26]. Solutions like pHost[27], and Express-Pass[28] send explicit control signals back to hosts, e.g., tokens. HOMA[29] and NDP[30] are credit-based solutions which is a significant shift from the state-of-the-art in practice since they have complex receivers. RCC[13] combines the control of last-hop congestion as fairness signal and in network congestion by PD controller on the switch as a two-stage process. However, PD controller will sacrifice bottleneck link throughput before entering into the fairness stage. The switching between these two stages might be nontrivial and require manual tuning.

2.2 Learning-Based Congestion Control

Pcc Vivace[31] is the first design that introduces online optimization into congestion control. Aurora[32] explores this idea by involving deep reinforcement learning. For internet CC, there are several works such as Remy[33] and Indigo[34], which perform optimization via offline learning. DeepCC[10] and MOCC[35] both proceed as online policy tuning after offline learning to achieve balance of different targets (i.e., throughput, delay and loss). Pareto[11] takes advantage of a state-machine reward design inspired by BBR[36] to achieve fairness for small fan-in flows. However, the internet network conditions may vary significantly depends on different types of com-

munication, and the initial rate of the network is not aligned with link capacity.

On the contrary, the data center network has a consistent hardware setup and the start rate always matches the link rate. Nevertheless, DCN faces design challenges due to rapidly changing flow numbers and bursty traffic patterns. As a consequence, algorithms trained for the internet, may not maintain the same performance if just simply migrate to DCN without fine-tuning. In DCN CC, RL-CC[9] benefits from simple end-to-end states and LSTM, but the action space is smaller and may potentially exceed the threshold of the underlying flow control algorithm.

Chapter 3

Background and Preliminaries

in this chapter, we explain the background of the data center congestion control problem and preliminary knowledge for designing a learning-based method.

3.1 Deep Reinforcement Learning

Reinforcement Learning (RL) is a learning approach that maps from environmental states to actions. The goal is to enable an agent to achieve the maximum cumulative reward during its interactions with the environment[37]. The Markov Decision Process (MDP) is commonly used to model RL problems. An MDP is typically defined as a quadruple (S, A, ρ, f), where:

1. S is the set of all environmental states. $s_t \in S$ denotes the state of the agent at time t.

2. A is the set of all possible actions the agent can take. $a_t \in A$ represents the action taken by the agent at time t.

3. $\rho : S \times A \to \mathbb{R}$ is the reward function. $r_t \sim \rho(s_t, a_t)$ represents the immediate reward received by the agent for taking action a_t in state s_t.

4. $p : S \times A \times S \to [0, 1]$ is the state transition probability distribution function. $s_{t+1} \sim p(s_t, a_t)$ denotes the probability of the agent transitioning to state s_{t+1} after taking action a_t in state s_t.

In reinforcement learning, the policy $\pi : S \to A$ is a mapping from the state space to the action space. This can be expressed as the agent selecting action a_t in state s_t, executing this action, transitioning to the next state s_{t+1} with probability $p(s_t, a_t)$, and receiving a reward r_t as feedback from the environment. If we assume that the immediate rewards received in all future time steps are discounted by a factor γ, the sum of the rewards from time t to the end of the episode at time T is defined as

$$R_t = \sum_{t'=t}^{T} \gamma^{t'-t} r_{t'},$$

where $\gamma \in [0, 1]$ serves to balance the influence of future rewards on the cumulative reward.

The state-action value function $Q^\pi(s, a)$ indicates the cumulative return that an agent receives by taking action a in state s and following policy π until the end of the episode. It is given by

$$Q^\pi(s, a) = \mathbb{E}[R_t | s_t = s, a_t = a, \pi].$$

For all state-action pairs, if a policy π^* has an expected return greater than or equal to all other policies, then π^* is deemed the optimal policy. While there might be multiple optimal policies, they all share a single state-action value function:

$$Q^*(s, a) = \max_\pi \mathbb{E}[R_t | s_t = s, a_t = a, \pi].$$

This is known as the optimal state-action value function and follows the Bellman optimality equation, which is

$$Q^*(s, a) = \mathbb{E}_{s' \sim S} \left[r + \gamma \max_{a'} Q(s', a') | s, a \right].$$

In conventional RL, the Q-value function is typically found by iteratively applying the Bellman equation:

$$Q_{i+1}(s, a) = \mathbb{E}_{s' \sim S} \left[r + \gamma \max_{a'} Q_i(s', a') | s, a \right].$$

As $i \to \infty$, $Q_i \to Q^*$. Iteratively applying this will ultimately converge the state-action value function and yield the optimal policy:

$$\pi^* = \arg\max_{a \in A} Q^*(s, a).$$

However, for practical problems, solving for the optimal policy using iteration as in Equation (5) is infeasible due to the computational cost, especially in large state spaces. To address this, RL algorithms commonly employ linear function approximators to represent the state-action value function, i.e.,

$$Q(s, a|\theta) \approx Q^*(s, a).$$

Additionally, nonlinear function approximators, such as deep neural networks, can be used to approximate either the value function or the policy.

However, conventional RL techniques can face challenges in high-dimensional or continuous state or action spaces. As the number of possible states or actions grows, it becomes computationally infeasible to represent and compute the value function or policy for every state-action pair. This is where Deep Reinforcement Learning (DRL) comes in. Many of the successes in DRL have been based on scaling up prior work in RL to high-dimensional problems. This is due to the learning of low-dimensional feature representations and the powerful function approximation properties of neural networks. By means of representation learning, DRL can deal efficiently with the curse of dimensionality, unlike tabular and traditional nonparametric methods[38, 39].

Among the plethora of strategies, value-based (e.g., DDPG[40], TD3[41]) and policy-based (e.g., TRPO[42], PPO[43]) approaches are of paramount importance due to their foundational nature. Value-based algorithms work by approximating the value of states or state-action pairs, and then make decisions based on these approximated values. The epitome of this category is the Deep Q-Network (DQN). For a

given state s, action a, next state s', and reward r, the Q-value update rule is:

$$Q(s,a) \leftarrow r + \gamma \max_{a'} Q(s',a').$$

The corresponding loss, fundamental to Q-learning and minimized during training, is defined as

$$L(\theta) = \mathbb{E}_{(s,a,r,s')\sim U(D)} \left[\left(r + \gamma \max_{a'} Q(s',a';\theta^-) - Q(s,a;\theta) \right)^2 \right],$$

where $U(D)$ is the process of drawing a transition from the replay buffer, θ are the parameters of the Q-network, and θ^- are the parameters of the target Q-network.

On the other hand, policy-based methods directly model and optimize the agent's policy. The policy, $\pi(a|s)$, defines the probability distribution over actions given a state. The main objective of optimization is to maximize:

$$J(\theta) = \mathbb{E}_{\tau\sim\pi_\theta}[R(\tau)],$$

where τ represents a trajectory $(s_0, a_0, s_1, a_1, \ldots)$, and $R(\tau)$ is the cumulative return for that trajectory. The policy gradient, which indicates the direction in which to modify the policy parameters to improve performance, is given by:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau\sim\pi_\theta} \left[\left(\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) \right) R(\tau) \right].$$

3.2 Remote Direct Memory Access Technology

Remote Direct Memory Access (RDMA) is a technology that enables two networked computers to exchange data in main memory without relying on the processor, cache, or operating system of either computer. An RDMA-enabled NIC must be installed on each device that participates in RDMA communications. Today's RDMA-enabled NICs typically support one or more of the following two network protocols:

RDMA over Converged Ethernet. RoCE is a network protocol that enables RDMA communications over an Ethernet The latest version of the protocol – RoCEv2

– runs on top of User Datagram Protocol (UDP) and Internet Protocol (IP), versions 4 and 6. RoCEv2 is currently the most popular protocol for implementing RDMA, with wide adoption and support.

InfiniBand. InfiniBand provides native support for RDMA, which is the standard protocol for high-speed InfiniBand network connections. Because of its ability to speedily connect large computer clusters, InfiniBand has found its way into additional use cases such as big data environments, large transactional databases, highly virtualized settings and resource-demanding web applications.

3.3 Data Center Network Congestion Event

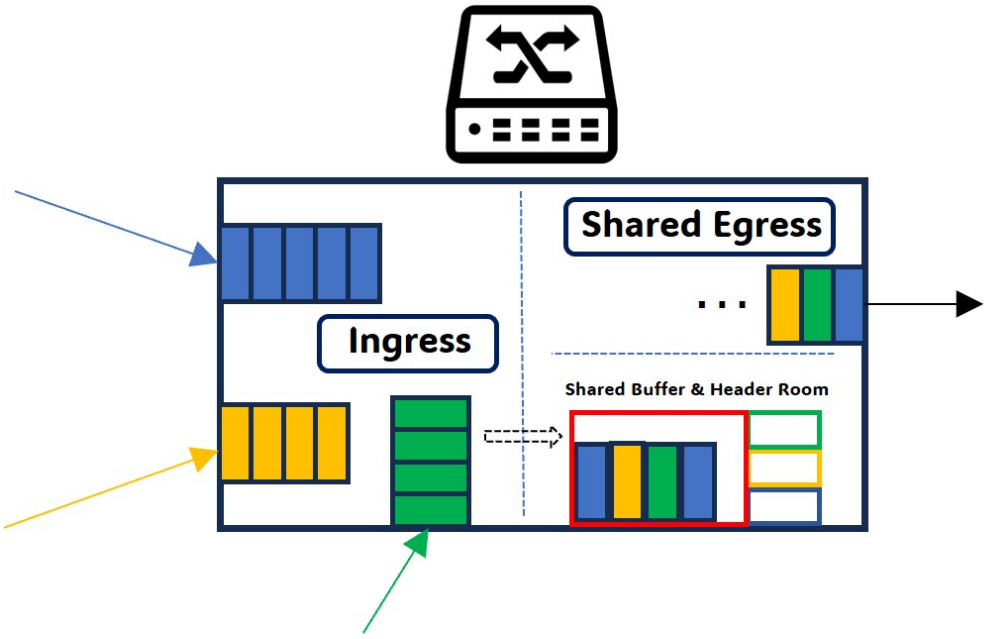


Figure 3.1: A high-level overview of Broadcom product switch used during our simulation

In the context of the dumbbell topology depicted in Fig. 3.4, an analysis of the congestion event on the left switch allows us to identify pivotal components indicative of congestion severity.

Fig. 3.1 illustrates that each port is assigned a distinct ingress queue of modest size, typically 4k bytes. Additional packets are accommodated in a communal buffer.

If a particular port's shared buffer utilization surpasses a defined limit, the header room associated with that port is leveraged to further buffer burst traffic. The priority flow control (PFC) threshold for a port, discussed in Chapter. 2.1.2, is determined by the equation:

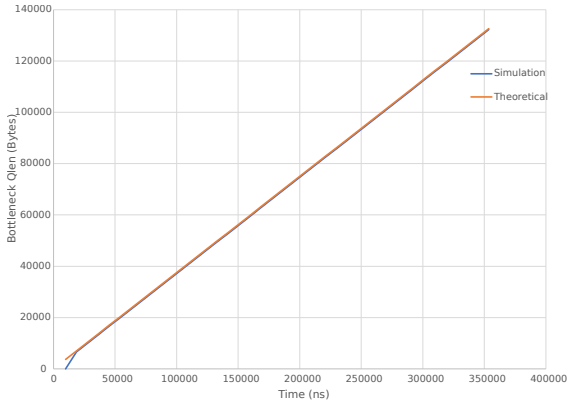
$$PFC_{threshold} = \frac{buffer\_size - total\_hdrm - total\_ingress - sharedBuffer\_size}{16},$$

where:

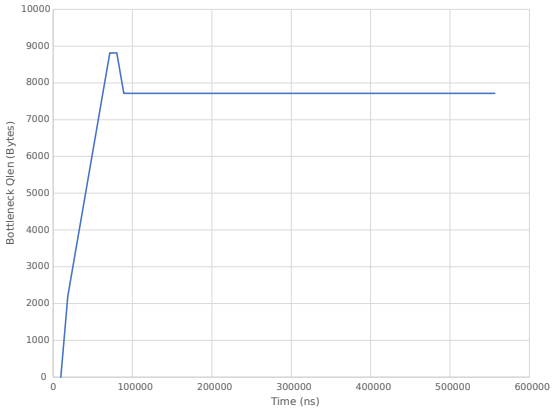
- $buffer\_size$ denotes the cumulative memory capacity of the switch.
- $total\_hdrm$ represents the aggregate headroom size across all ports.
- $total\_ingress$ signifies the collective ingress queue size for all ports.
- $sharedBuffer\_size$ measures the present shared buffer consumption across all ingress flows.

Concurrently, flows directed towards a common subsequent hop utilize the same egress queue at the egress port. Based on the switch's memory architecture, we can scrutinize the egress queue size in conjunction with the ingress and egress data rates to elucidate the correlation between queue length and data rate. In a 4-to-1 dumbbell topology, each sender transmits a flow to the receiver at 1Gbps, matching the link rate of 1Gbps. Consequently, the switch's queue length growth rate will be 3Gbps. Post-congestion control, even if the aggregate ingress data rate aligns with the egress data rate, congestion—manifested as packet delays—is reduced. Nevertheless, the queue length persists at an elevated state. By further constraining the ingress data rate to be less than the egress data rate, the buffer transitions to a queue length draining phase, thereby further alleviating congestion, as depicted in Fig. 3.2.

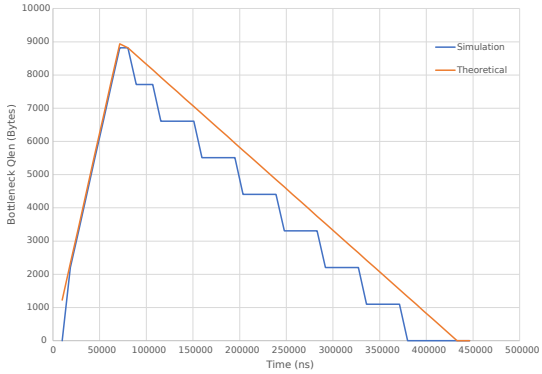
From this rudimentary modeling and validation, it becomes apparent that our focus should be on devising an algorithm that accelerates queue growth and also efficiently drains queue occupancy. Given that the egress rate is invariably set to the link rate



(a) $\sum Ingress\_rate > Egress\_rate$



(b) $\sum Ingress\_rate = Egress\_rate$



(c) $\sum Ingress\_rate < Egress\_rate$

Figure 3.2: 4-to-1 Dumbbell Topology Bottleneck Switch Queue Length

in the topology, adjusting the ingress rate from the flow sources emerges as the main viable strategy for congestion control.

3.4 Main Challenges for Data Center Congestion Control

Fairness of flows in burst traffic. Congestion algorithms may not be able to accurately estimate the available bandwidth or the degree of congestion as each flow departs from the switch buffer at different times, resulting in different observed information. As a result, they might under-react or over-react to the traffic bursts,

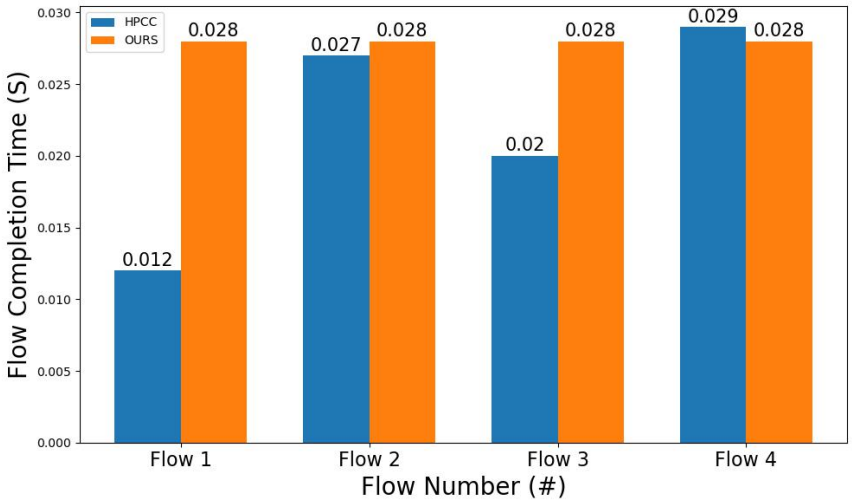


Figure 3.3: Flow Completion time comparison in 4-to-1 fan-in traffic. Flow completion time is the absolute time period of each flow.

leading to suboptimal performance and unfair allocation of network resources. For example, 4-to-1 fan-in traffic simultaneously starts in a dumbbell topology as Fig. 3.4, a fine-grained flow level CC like HPCC illustrates some performance degradation. As shown in Fig. 3.3, all four flows under HPCC control demonstrate different completion times. Different instantaneous observations in the burst scenario can lead to unfair flow pacing rates. Therefore, our goal is to enable our algorithm to predict global changes from local observations, in order to expedite the convergence process for fairness.

Table 3.1: Algorithm attributes comparison. BW denotes host link capacity, min_rate denotes the minimum threshold of pacing rate, hyperAI denotes the addictive factor, RTT denotes the round-trip time, Goodput denotes the correct throughput.

Algo	**Environment Provided States**	**Action Space**
HPCC	Inflight bytes of the most congested link	$(0, \frac{BW}{\text{min\_rate}}]$
DCQCN	ECN	$[0.5, \frac{BW-\text{min\_rate}}{2}]$
TIMELY	RTT	$(0, \frac{\text{hyperAI}}{\text{min\_rate}} + 1]$
RLCC	RTT, Current rate	$[0.8, 1.2]$
OURS	Inflight bytes of the most congested link, RTT, Goodput, Current rate	$[0.5, 2]$

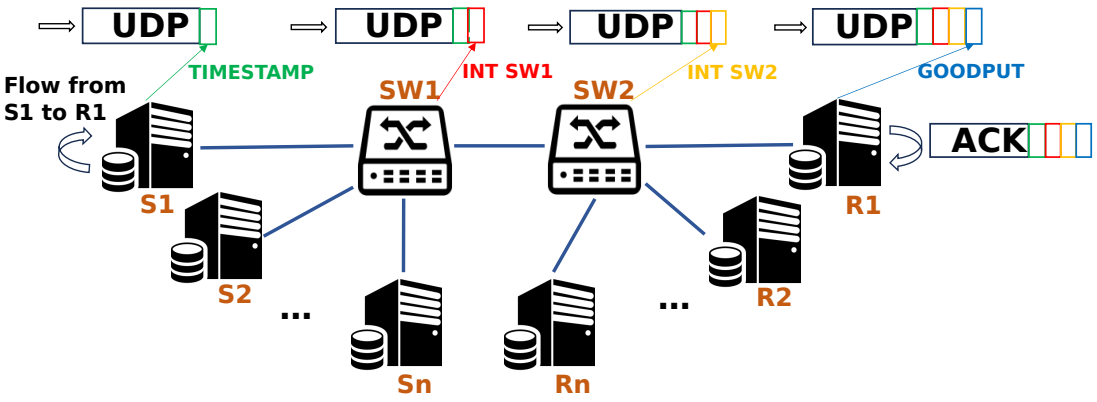


Figure 3.4: The overview of network state perception in an n-to-n dumbbell topology within a data center network.

Feedback and control delay due to previous congestion. Feedback delay has long posed a significant challenge in congestion control. This disadvantage becomes evident when the flow source receives an acknowledgment from the receiver (i.e., $qlen_t$ for the buffer occupation at time t), and the current buffer occupation may have changed to $qlen_{t+1}$ such that $qlen_{t+1} \gg qlen_t$ or $qlen_{t+1} \ll qlen_t$. This scenario can occur in incast situations. In such cases, a component capable of learning the dependencies of feature changes over time could prove valuable in addressing the congestion control problem. Furthermore, our analysis of previous learning-based congestion control methods for data center networks indicates a significantly smaller action space for rate manipulation, compared to heuristic rule-based algorithms, as demonstrated in Table 3.1. We estimate the action space by calculating the maximum potential rate magnitude change after a single update step. The result may not be sufficiently accurate, as the rate increase stage is typically designed as an additive process. Nonetheless, the action space remains considerably larger than that of a learning-based method. As both types of algorithms are event-triggered, a smaller action space could potentially result in less efficient control at the same step size. Adjusting the action space size could enhance the performance of the learning-based algorithm.

Information utilization across the flow path. Upon comparing previous algorithms, it is evident that the design of these algorithms often utilizes only a subset of available network statistics. Specifically, Table 3.1 enumerates the different features each algorithm considers. Given that different network features can contribute to an algorithm's convergence, it would be beneficial to consider all these features collectively. This would allow us to leverage the specialized feature representation learned by a deep neural network to develop an enhanced control policy.

3.5 In-band Network Telemetry

In-band network telemetry (INT) technology[44] collects hop-by-hop network status information through business packets to achieve end-to-end visualization of network services. In-band network telemetry uses the data plane to directly drive the network measurement process, subverting the research idea of traditional network measurement that treats network switching devices as an intermediate black box. It also has the advantages of flexible programming, strong real-time, less noise and path-level network status perception, etc. In our design, we follow the INT structure of HPCC[8], add an additional flow number information, and abandon the pathID during the training phase.

3.6 Reinforcement Learning for Datacenter Congestion Control

First, we reformulate the problem from a reinforcement learning perspective. Considering a dumbbell topology (as depicted in Fig. 3.4), multiple senders are transmitting packet flows to the receivers via a single bottleneck link, connected by two switches. It's worth noting that no specialized scheduler exists on each host to ensure fairness, such as round-robin scheduling between each flow. Instead, rate adjustment is an

event-triggered process, i.e., it reacts upon receiving an ACK. Therefore, each flow is assigned an agent to control its packet pacing rate, and these agents are expected to achieve optimal control wherein all flows traversing the bottleneck link at a given time can evenly share the bandwidth and fully utilize the link capacity. More specifically, as an agent is allocated per flow, it can only pace the packets based on the ACK packets it receives from its corresponding receiver. Based on the above preliminaries, the problem here can be defined as a partially observable Markov decision process (POMDP) under multi-agent control, as each flow perceives the network states according to local observation and cannot communicate with each other.

To address the issue more formally, comparing the tuple description of MDP $\langle S, A, p, \rho, \gamma \rangle$ and POMDP $\langle S, A, p, \rho, \omega, O, \gamma \rangle$, where S is the states set, A is action set, p is the state-transition functions, which defined as $P(s_{t+1} \in S | s_t \in S, a_t \in A)$, ρ is the reward function, ω is the observation transition function $\omega(o_{t+1} \in O, s_{t+1} \in S)$, i.e., the probability of obtaining an observation o from global state s, O is the the set of observations of one agent, γ is the discounted factor. In a POMDP, an agent only gets to observe a certain representation of the state, which may not capture all the relevant aspects. This uncertainty introduces significant challenges when it comes to finding optimal policies.

There are several existing approaches to tackle this kind of problem, e.g., recurrent neural network[45] and memory-augmented neural network[46, 47].

Inspired by the previous work[48], we believe that leveraging the recurrent component of neural network can help our agents learn the representation of the state and be capable of predicting the congestion status. Also, for the multi-agent settings, regarding the homogeneous property of each agent (i.e., identical observation and action space), we utilize parameter sharing, i.e., all the agents share both the policy and value function parameters, for which proven work has illustrated promise in learning efficiency [49, 50].

Based on the aforementioned problems and potential solutions, we have designed

a deep reinforcement learning algorithm that utilizes historical sequences and long-short term memory to directly modify the inflight window size of each flow through control signal generation. The algorithm incorporates explicit reward guidance for both queue draining and maintaining fairness. The following chapters will detail our method and present evaluation results for various traffic patterns across different topologies.

Chapter 4

Method Designs

4.1 Overview

Our FDCC aims to achieve better fairness in bursty traffic scenarios. Compared with conventional learning-based algorithms or heuristic rule-based algorithms that only take specific network features into algorithm design, we consider all available observations during one control loop, i.e., the round-trip of a packet. The workflow of FDCC can be concluded as two stages: 1) observations collection: When NIC is able to deliver a new packet, the sender inserts the current timestamp into the packet header, then upon each time the packet gets dequeued from a switch, the switching ASIC inserts some meta-data that reports the current load of the packet's egress port, which is the same as HPCC[8]. Once the packet arrives at the receiver, the receiver migrates the UDP packet header to the ACK packet header and inserts the corresponding timestamp and payload received so far into the header for further goodput calculation. 2) Event triggering control: The agent deployed on the sender will adjust the inflight bytes window and data pacing rate per ACK after data pre-processing of observations stored in ACK header. The details will be discussed in Chapter. 4.2.

Besides, traditional transmission rate modulation such as AIMD[51], is not efficient enough in data center bursty traffic scenarios. The success of HPCC[8] has proven a combination of AIMD and MIMD can effectively control the congested link within

one RTT. Also, a window-based control can be more proactive as it directly restrains inflight bytes within one Bandwidth-delay product, thus in our approach, the policy will generate a factor to manipulate inflight bytes window size as in MIMD modulation and calculate packet pacing rate based on the window as $Rate_t = WIN_t/baseRTT$. Another problem that mainly obstacles learning-based algorithm to work efficiently in DCN is the generalization problem. Agents need to be trained in different traffic patterns to learn a set of Pareto-optimal or near-optimal policies. In the training of FDCC, we conduct a curriculum learning process and design stage reward to facilitate our DRL agent to be able to generalize to different bursty traffic and achieve better fairness. In the following chapters, we explain the details of our method design.

4.2 State-transition Pair Components Definitions

We elaborate on the choices in the design of the action space, observations, neural network model, reward function and optimization objective.

Observations. The input of our model consists of observations available from the sender, switches, and receiver.

- `RTT`: Round-trip time calculated by subtracting the current timestamp and timestamp stored in ACK header, normalized by standalone RTT.
- `QLEN`: Buffer occupation obtained while UDP packet dequeue from switch. Here we use the in-band network telemetry(INT) same as paper[8], and only consider the most congested link, normalized by the bandwidth-delay product of the corresponding link.
- `SWITCHRATE`: Throughput of the most congested link, we compared the `QLEN` of current and previous timestamp to estimate the bytes on fly of the link, normalized by the bandwidth of corresponding link.
- `GOODPUT`: The data received at Receiver following the correct sequence between

each action interval, normalized by the bandwidth of the receiver's link.

- $\triangle$`RTT`: The derivative of delay change between two steps, calculated as $\frac{(RTT_t - RTT_{t-1})}{dt}$, dt is the time interval in milliseconds.
- $\triangle$`QLEN`: The derivative of `QLEN` change between two steps, calculated as $\frac{(QLEN_t - QLEN_{t-1})}{dt}$.
- `INSRATE`: Current packet pacing rate of the flow.
- `last_action`: The action took at last step.

Actions. In HPCC, the control mechanism can be divided into two parts: rate-pacing adjustment based on the inflight bytes of the most congested link and host-sending restriction based on the inflight bytes of the flow's source link. The relationship between these two parts can be formulated as: $WIN_t = rate_t \cdot baseRTT$, where $baseRTT$ stands for the longest standalone RTT of a single flow's round-trip time, WIN stands for the size of inflight bytes of a flow, which is restricted by bandwidth-delay product, $rate$ represents the pacing rate of packets. We found that with only the rate modification based on the most congested link, the buffer can easily become full, as shown in Fig. 4.1, and may even trigger PFC signals in small fan-in traffic. The key insight here is that the inflight bytes of the host can be considered as the only feedback signal in congestion control, one that experiences no delay. Therefore, we believe that using window-based control to calculate the corresponding pacing rate can be a more reasonable and robust approach for handling bursty traffic. Upon obtaining observation o, the DRL agent will choose action a. The output of neural network will be rescaled by following formula to reduce oscillation and ensure algorithm can converge under different circumstances. The rescale function denotes as below:

$$\text{action}_{t+1} = \begin{cases} \text{action}_t \cdot (1 + \alpha \cdot output) & \text{if } output >= 0 \\ \text{action}_t / (1 - \alpha \cdot output) & \text{if } output < 0 \end{cases}$$

where α is a scaling factor. In this thesis we set it equal to 0.5. The new inflight

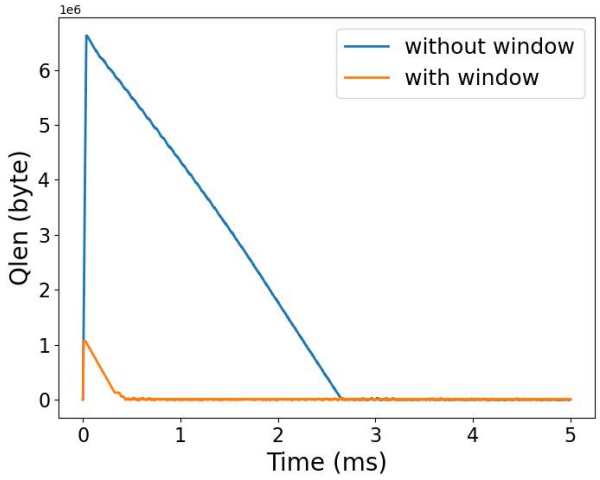


Figure 4.1: Compare the buffer occupation change w and w/o host inflight bytes window. 64-to-1 incast traffic with flow initialized within $1\mu s$ in a dumbbell topology.

window on host will be updated as $WIN_{t+1} = WIN_t \cdot \text{action}$, and the new packet pacing rate is derived as $\texttt{INSRATE}_{t+1} = WIN_{t+1}/baseRTT$

Rewards. Inspired by Pareto[11], our reward is designed for two phases. As the default initial rate in DCN is set to link rate, we do not need to consider a startup phase, instead, we only consider queue-draining and bandwidth-probing phases. As we have collected INT information from the switches along our flow path, we first compare the sum of `SWITCHRATE` and `QLEN` to identify the most congested link, then we compare it with a tolerant threshold to decide which stage our flow is on. For queue-draining phase, we use round-trip time with a tolerance threshold to help the agent mitigate congestion. For the bandwidth probing phase, we want to force flow to fully utilize the bottleneck link while remaining fairness if there is flow competition. The details of reward calculation can be found in Algo. 1. We take the reciprocal of normalized `RTT` to ensure a similar influence of latency and throughput in reward contribution. In addition to available observations, we add a flow number counter at each egress port of the switch to calculate the number of competing flows on the most congested link. We enhance the INTheader by incorporating an additional field, denoted as the *opcode*, derived from the InfiniBand Basic Transport Header[52].

Consequently, we are able to concurrently retrieve routing information and monitor the first and last packet of any flow traversing the egress port as shown in Algo. 2. However, this operation is not enabled during the experiments to reduce computation overhead. Also, considering the delay of feedback, we use n-step discounted cumulative reward instead of instant one for each step, i.e., $r_t = \sum_{i=0}^{i<n} \gamma^i \cdot r_{t+i}$.

Algorithm 1 Reward Calculation Demonstration

for Each step in one episode **do**
  if $\texttt{SWITCHRATE} + \texttt{QLEN} - 1 > 0.5$ **then**
    reward $= -\texttt{RTT}$
  else
    reward $= \texttt{THROUGHPUT} + \frac{1}{\texttt{RTT}} - (\texttt{THROUGHPUT} - \frac{1}{\text{Total\_flow\_num}})^2$
  end if
end for

Algorithm 2 Flow number count on switch

Input: *pkt.sip*, *pkt.dip*, *pkt.sport*, *pkt.dport*, *opcode*
Output: Total_flow_num

flow_id = hash32(sip,dip,sport,dport)
if $flow\_id$ in $Egress\_port\_flow$ **then**
  if *opcode* is not `Write Last` **then**
    pass
  else
    del $Egress\_port\_flow[flow\_id]$
  end if
else
  $Egress\_port\_flow[flow\_id] = True$
end if
return $Egress\_port\_flow$.size()

4.3 Neural Network Architecture and Optimization Objective

Neural Network Architecture. We use a modified Twin Delayed DDPG (TD3) as our deep reinforcement learning algorithm based on memory-based TD3[53]. As

shown in Fig. 4.2, the memory buffer will not only store observation-action pair $\langle o_t, a_t, r_t, o_t + 1\rangle$[1] but also a historical batch of observations before time step t and $t + 1$, the historical batch length $k \in K$, where $\max K \leq 8$. It happened that the current observation is the end of an episode, thus the length of history may not align with each other, and padding is mandatory for the input of LSTM layer. Therefore, the lengths of historical observations will be an additional input and be used as an index for *torch.gather* operation to retrieve the non-zero output of LSTM. During the inference phase, the input consists solely of a sequence of observations from $t - k$ to t. The representation of the observations thus far is established by default as the last vector derived from the LSTM layer.

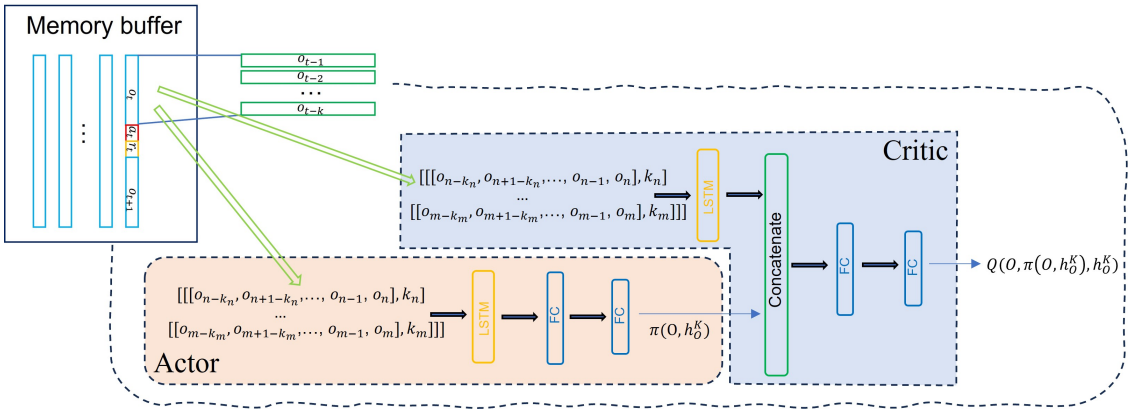


Figure 4.2: Actor-critic architecture of FDCC at training phase. For simplicity only one pair of actor-critic network is demonstrated here, the other parts' implementation follows standard TD3 architecture[41]. The memory buffer stores each observation-action pair incorporating a maximum sequence length of k historical observations.

Policy Optimization. Twin delayed DDPG is an off-policy reinforcement learning algorithm that learns a Q-function (action-value function). TD3 extends the DDPG[40] algorithm by introducing twin critics and delayed policy updates to mitigate the overestimation bias common in Q-learning algorithms. The actor network approximates a policy as $a = \pi_\phi(s)$, and the critic network gives the value function

[1]The subsequent references to observations and states in this thesis pertain to the identical set of features accessible to each agent, given that none of them have access to the global states.

regarding policy as $Q^{\pi_\phi}(s,a|\theta_i), i=1,2$. The optimization objective of the actor network is to maximize the expected return of value function $J(\phi)=\mathbb{E}[Q^{\pi_\phi}(s,a|\theta_i)]$. In order to approximate the optimal action-value function, which satisfies the Bellman equation:

$$Q^{\pi_\phi}(s,a|\theta_i)=r(s,a)+\gamma\mathbb{E}[Q^{\pi_\phi}(s',a'|\theta_i')]. \tag{4.1}$$

The critic network will be updated by minimizing TD error:

$$L=argmin_{\theta_i}N^{-1}\sum(y-Q^{\pi_\phi}(s',a'|\theta_i)), \tag{4.2}$$

where $y=r(s,a)+\gamma\min_{i=1,2}Q^{\pi_\phi}(s',a'|\theta_i')$. Then the actor network will be updated by deterministic policy gradient:

$$\nabla_\phi J(\phi)=N^{-1}\sum\nabla_a Q^{\pi_\phi}(s,a|\theta_1)|_{a=\pi_\phi(s)}\nabla_\phi\pi_\phi(s). \tag{4.3}$$

After training, the actor parameter will converge to a near-optimal policy with the help of critic function.

4.4 Training Procedure

We use curriculum learning to gradually help our policy be able to generalize to different traffic scenarios in offline training. In our data center network environment, we have static hardware conditions, the only changing schemes are the topology and the traffic patterns. However, as our algorithms take normalized observations which are also topology independent, we only train our algorithms on dumbbell topology and evaluate it on both dumbbell and two-level fat-tree topology.

During the offline training, we first train the agents in a 2-to-1 fan-in traffic scenario, where two flows are scheduled to send to the same destination within 1 ms. Afterward, we add 4-to-1 fan-in traffic into training and then 8-to-1 traffic; the fan-in number increases by 2 times every 10k episodes until 32-to-1. The action interval is a hyperparameter here as there is delay of both observations feedback and previous action effect illustration. We set this interval to be dynamic, based on 1/2 of the previous `RTT`.

Chapter 5

Common Experiment Settings

5.1 Simulator and RL Environment

in this chapter, we conduct extensive experiments using the most common traffic patterns in data centers on different scales.

5.1.1 Experiment Setup

We conducted experiments on our algorithm and other baselines using NS-3, a C++-based simulated network environment that is commonly used in previous papers[7, 8]. The infrastructure of network is inherited from previous work as well. To set up the reinforcement learning environment, we utilized ns3ai [54], a shared memory communication tool that enables the transmission of states and actions between the Python DRL algorithm and the simulator. The workflow of our proposed system is illustrated in Fig. 5.1. Each flow on a host is associated with a dedicated DRL agent. Notably, all these agents share a common parameterized policy network to determine the magnitude of the `WIN` adjustment.

5.2 Baselines

After a thorough examination of several previous works, we have selected three common baselines based on heuristics, namely HPCC, DCQCN, and TIMELY, as well as one learning-based baseline named RL-CC. We used the default parameters recom-

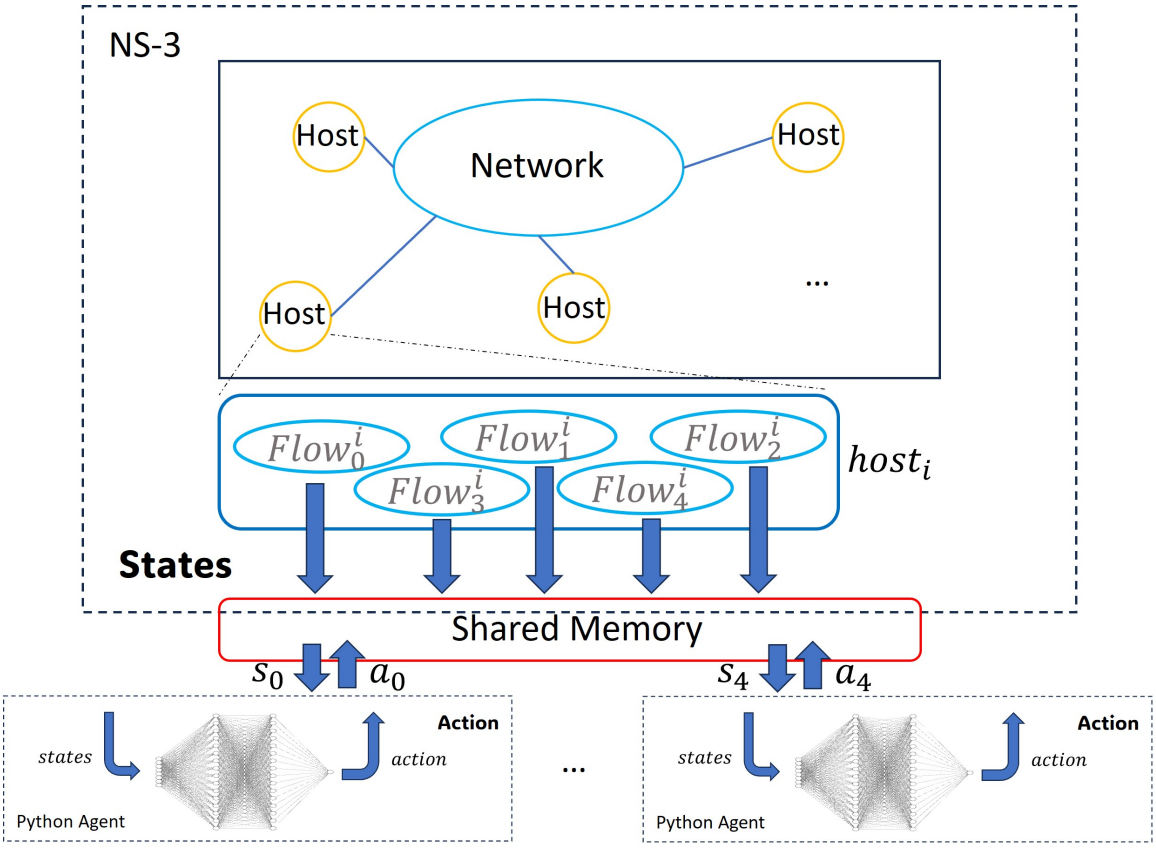


Figure 5.1: NS-3 and Python Algorithm Communication.

mended in their respective papers for all the baselines. In our approach, we rounded down the updated window to an integer to implement a more conservative control strategy, considering that each packet in our experiment has an identical MTU size of 1000 bytes.

5.3 Benchmarks

To evaluate the impact of our algorithms on performance metrics, we conducted tests on both dumbbell and fat-tree topologies, comparing them to baselines. For dumbbell topology, we compare the n-to-1 and long-short traffic pattern in a 64-to-64 topology described in §3.6. For the n-to-1 traffic pattern, the evaluation starts from 4-to-1 incast. Once the flow number exceeded the number of hosts of our topology, each host

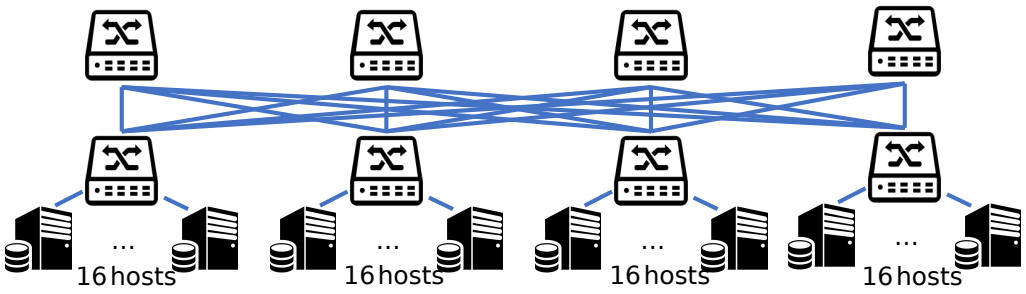


Figure 5.2: Two-level fat-tree topology.

sent multiple flows. All flows were initialized following a Gaussian distribution with an interval of 1 millisecond. In the case of the fat-tree topology, we utilized a two-level fat-tree network with 4 aggregation and 4 access switches. Each access switch was connected to 16 hosts, as depicted in Fig. 5.2. We compare n-to-1 and n-to-n traffic patterns. For n-to-1, we used the same traffic initialization interval as dumbbell. For n-to-n, we generated our flows using two publicly available distributions: *Web_Search*[5] and *FB_Hadoop*[15] to generate our flows. In all our experiments, the algorithms compared had the host inflight bytes window enabled. This means that an additional threshold was added to each flow to prevent any flow from having more than one bandwidth-delay product unacknowledged packets in flight. For all the links in our topologies, the bandwidth is set to 100Gbps which is compatible with real-world data center networks.

5.4 Test Metrics

In our performance evaluation, we utilize several commonly used metrics for comparison purposes.

- **FCT slow down**: Flow completion time slow down is calculated as the ratio between the practical absolute[1] flow completion time and the standalone flow completion time. A smaller value is preferable.

[1]The term 'absolute' refers to the time interval calculated as the difference between the end timestep and the beginning timestep, rather than a relative interval based on the global time simulation start and endpoints.

- **Qlen**: Qlen size is the buffer occupation of the switch. A lower value is preferable.

- **PFC pause time**: Priority flow control is the underlying flow control mechanism to facilitate the lossless communication of data center network. It is a dynamic threshold based on the shared buffer occupation and individual occupation on the switch ingress port. A smaller pause time percentage indicates more effective control.

- **Goodput**: Goodput refers to the effective throughput measured by the receiver. Higher values indicate better performance.

- **Unfair ratio**: The unfair ratio is calculated as the deviation of the normalized bandwidth of each flow during an incast event(i.e., $\frac{(BW_{max}-BW_{avg})}{BW_{avg}}$). A smaller value is preferred.

Chapter 6

Evaluation and Analysis

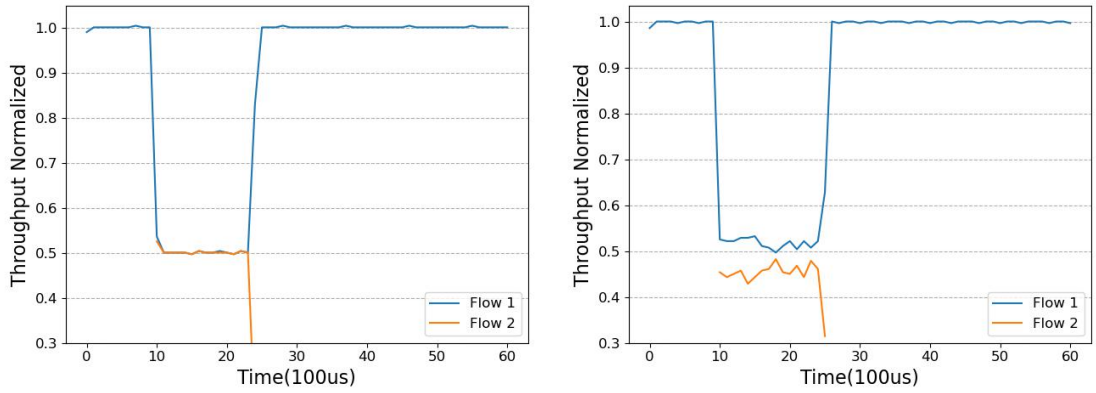
6.1 Long-short Traffic Experiments

Long-short setup. A continuous long flow is maintained in the background, while a short flow starts transmitting at 1 ms and completes within 1.5 ms.

Evaluation results. The results demonstrate that our algorithm utilizes the bottleneck link more efficiently than HPCC (as shown in Fig. 6.1). We also conducted a fair-share experiment where 4 flows join a link one by one every 35 ms, as shown in Fig. 6.2. Our algorithm exhibits reduced fluctuations upon the introduction of a new flow, leading to a more equitable distribution of bandwidth. This behavior is attributed to our algorithm's ability to anticipate future congestion changes. Although a constrained action space might render the algorithm less efficient in extensive incast scenarios, as detailed in Sec. 6.2, it potentially enhances stability in scenarios involving small, long-short traffic patterns.

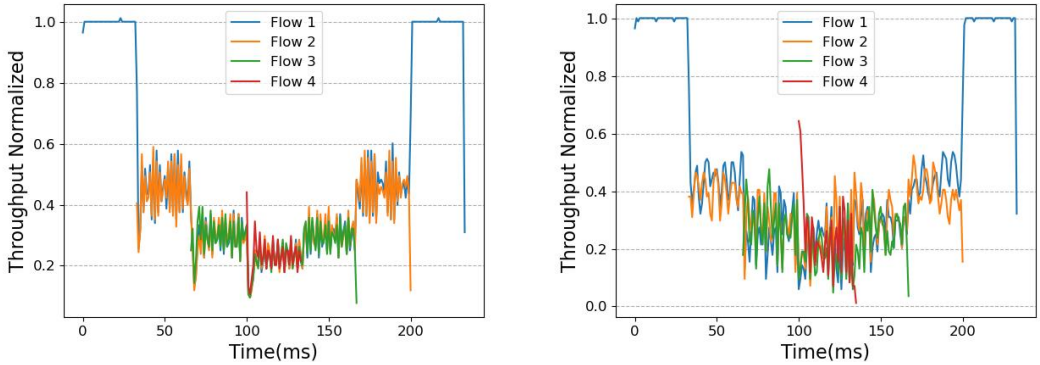
6.2 Incast Traffic Experiments

Incast setup. For incast simulation, we compared different baselines and our algorithm in both dumbbell and fat-tree topology. In dumbbell topology, all flows share a single bottleneck link. We conducted an n-to-1 fan-in experiment where each flow has a 20MB payload size in total and is initialized within 1 ms following Gaussian distribution. For fat-tree topology, the n-to-1 flows are generated from the first three



(a) OURS (b) HPCC

Figure 6.1: Long-short term traffic test.



(a) Fairness of OURS (b) Fairness of HPCC

Figure 6.2: Fair-share comparison in flow join traffic.

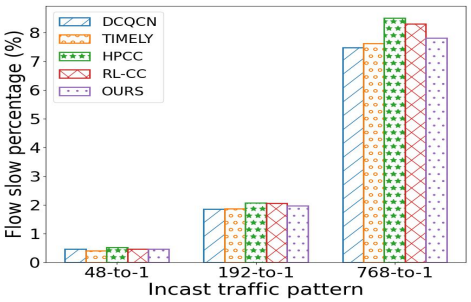
clusters and sent to the receiver in the final cluster.

Evaluation results. For dumbbell topology, the results are summarized in Table 6.2. Due to the additional header needed to store network states, it's expected that both our algorithm and HPCC demonstrate a larger slowdown in flow completion time as all flows here are long flows. However, our FDCC provides better overall fairness and comparable goodput. It's worth noting that our algorithm does not exhibit a better Qlen control compared with HPCC. This is reasonable since our action space is still much smaller than HPCC, even if we control the inflight window by our action.

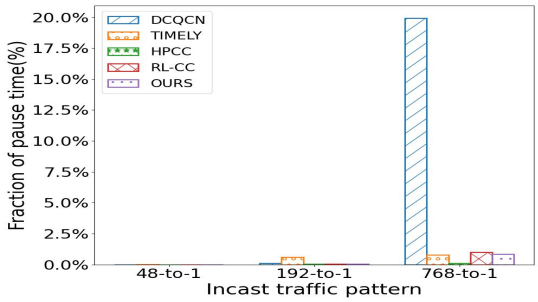
For fat-tree topology, the same as in dumbbell, each flow has 20MB payload in total and starts within a 1ms interval. Unlike the dumbbell topology, the fat-tree topology

Table 6.1: Normalized goodput comparison of n-to-1 traffic in fat-tree.

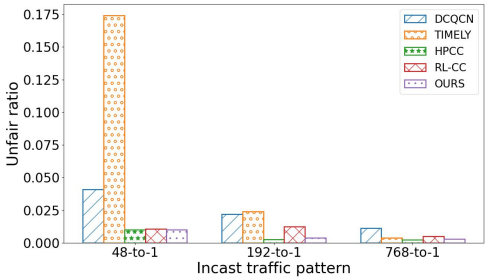
	DCQCN	TIMELY	HPCC	RL-CC	**OURS**
48-to-1	0.99	0.99	0.87	0.98	0.98
192-to-1	0.98	0.98	0.88	0.89	0.93
768-to-1	0.97	0.96	0.86	0.88	0.94



(a) FCT Slow Down



(b) PFC Pause Comparison



(c) Fairness Comparison

Figure 6.3: n-to-1 Two-Level fat-tree evaluation. Flow slow percentage *w.r.t.* percentile of flow slow down, Fraction of pause time *w.r.t.* percentile of PFC pause time over flow completion time, Unfair ratio *w.r.t.* the bandwidth allocation deviation.

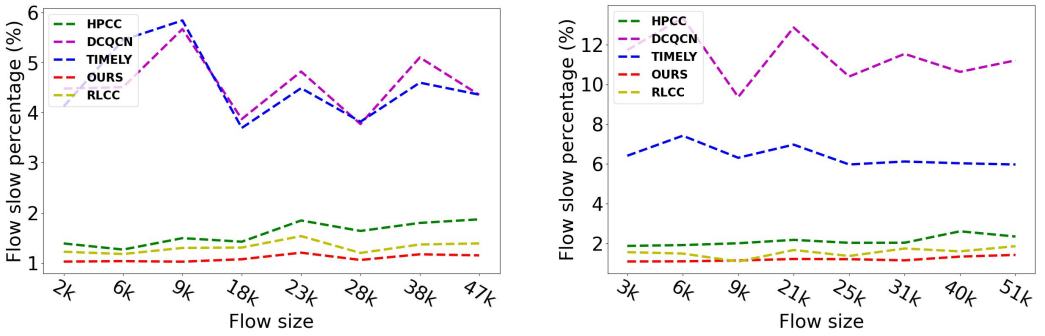
features multiple bottleneck links where the most congested link may transfer from hop to hop. Consequently, the feedback may experience further delays and all the algorithms might trigger the flow control mechanism. The results resemble dumbbell experiments where the inclusion of INT header causes a slowdown in both our algorithm and HPCC. Meanwhile, ours performs the best fairness and triggering less PFC and offers comparable goodput compared with DCQCN, TIMELY, and RL-CC, as shown in Table 6.1 and Fig. 6.3.

6.3 Mix Burst Traffic Experiments

Mix burst setup. We employ a public flow distribution to generate flows for our n-to-n mixed burst flow experiments, with a particular emphasis on short flow slowdowns. The flow intervals are determined using a Poisson distribution, where λ is equivalent to the average packet arrival interval, derived from the public flow size distribution. This is given by:

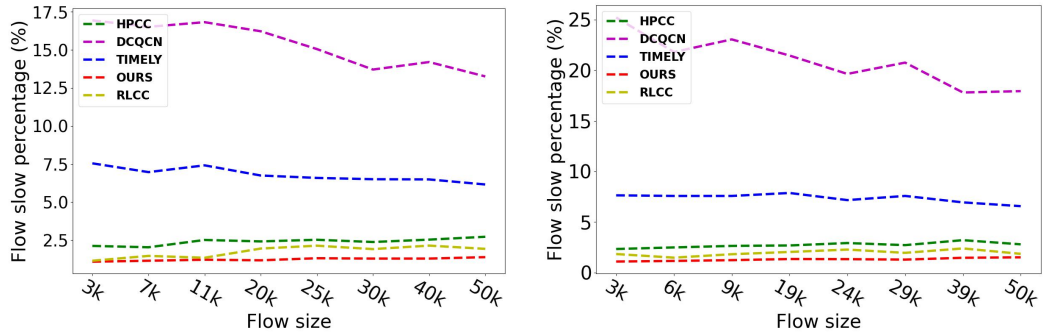
$$ArrivalInterval_{avg} = \frac{1}{BW \times \frac{workload_{\%}}{8 \times flow\_size_{avg}}} \times 10^9,$$

where BW denotes the link capacity connected with the hosts, The workload is incrementally increased from 20% to 80% to assess the resilience of the algorithms. **Evaluation results**. As illustrated in Fig. 6.4 and Fig. 6.5, our algorithm consistently outperforms the others, demonstrating better control over the slowdown of short burst flows.



(a) Work Load 20%

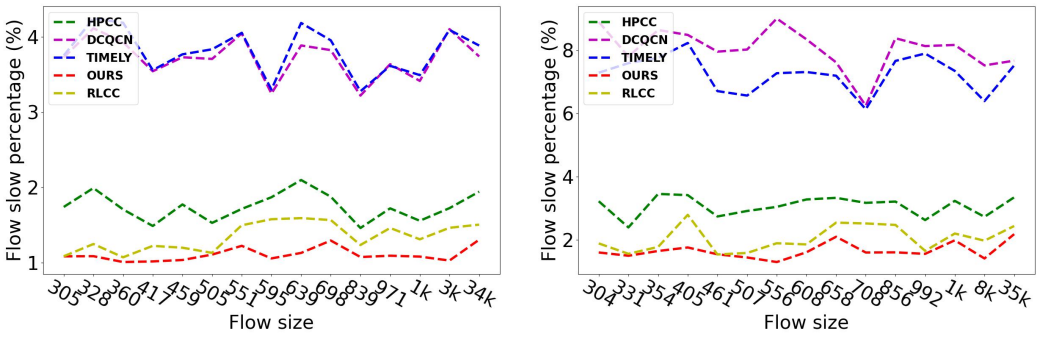
(b) Work Load 40%



(c) Work Load 60%

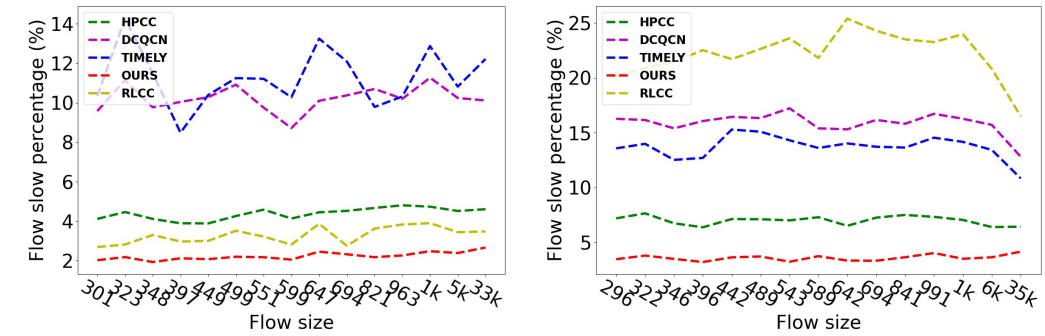
(d) Work Load 80%

Figure 6.4: *Web_Search* Two-Level fat-tree evaluation.



(a) Work Load 20%

(b) Work Load 40%



(c) Work load 60%

(d) Work Load 80%

Figure 6.5: *FB_Hadoop* Two-Level fat-tree evaluation.

Table 6.2: Comparison of different algorithms in dumbbell topology with n-to-1 fan-in traffic. Qlen denotes buffer occupation in bytes, FCTSD denotes FCT slow down, and Unfair Ratio denotes the bandwidth allocation deviation (i.e., $\frac{(BW_{max}-BW_{avg})}{BW_{avg}}$).

4-to-1	Qlen avg	Qlen 99^{th}	FCTSD avg	FCTSD 99^{th}	Unfair Ratio	Goodput
DCQCN	42993.3	48208	4.0	4.1	0.00002	0.96
TIMELY	43653.6	49632	4.0	4.0	0.00002	0.95
HPCC	1089.1	3354	3.3	4.3	0.29881	0.93
RL-CC	49564.8	52546	4.3	4.3	0.00001	0.90
OURS	49275.3	51128	4.1	4.1	0.00002	0.93

16-to-1	Qlen avg	Qlen 99^{th}	FCTSD avg	FCTSD 99^{th}	Unfair Ratio	Goodput
DCQCN	135259.8	151960	15.4	16.0	0.03805	0.99
TIMELY	130874.7	144672	15.8	16.2	0.01845	0.96
HPCC	2804.3	6708	17.4	17.6	0.01279	0.88
RL-CC	253288.8	253786	17.0	17.0	0.00003	0.89
OURS	30716	31525	16.9	16.9	0.00002	0.90

256-to-1	Qlen avg	Qlen 99^{th}	FCTSD avg	FCTSD 99^{th}	Unfair Ratio	Goodput
DCQCN	1098612.9	2882000	253.0	255.5	0.00997	0.96
TIMELY	1372074.3	4037088	293.8	296.1	0.00762	0.83
HPCC	131961.3	3274622	294.5	295.2	0.00243	0.83
RL-CC	4167130.2	4286572.2	255.1	255.4	0.00117	0.95
OURS	3578233.5	3688457.3	254.5	254.7	0.00091	0.96

1024-to-1	Qlen avg	Qlen 99^{th}	FCTSD avg	FCTSD 99^{th}	Unfair Ratio	Goodput
DCQCN	6718384.9	6764840	975.4	982.1	0.00683	0.99
TIMELY	1914594.7	6554592	1026.1	1028.3	0.00218	0.95
HPCC	1347184.6	6502288	1089.2	1091.4	0.00195	0.89
RL-CC	3085705.3	6987246	984.35	986.5	0.00225	0.98
OURS	2495439.2	6572722	975.6	977.1	0.00153	0.99

Chapter 7

Conclusions & Future Work

7.1 Conclusions

In our study centered on the congestion control of data center networks, we meticulously examined the critical elements closely associated with the degree of traffic congestion. We identified `RTT`, `GOODPUT`, `QLEN`, and `SWITCHRATE` as the primary measurable variables along the flow path. Moreover, the `WIN`, computed using the bandwidth-delay product, emerges as the sole delay-free state and serves as our action within our DRL algorithm for host pacing rate adjustment.

We have proposed a learning-based, model-free congestion control algorithm called FDCC. It aims to provide improved fairness and compatibility for flow completion time (FCT) and goodput in various burst traffic scenarios. The results have demonstrated the effectiveness of a more stringent control by directly adjusting the host's inflight bytes window and utilizing all available information from the current time step and historical records with staged reward-guided training.

7.2 Future Work

7.2.1 Algorithm Optimization

Based on simulation results, it is evident that several trade-offs warrant further investigation and optimization in upcoming research. The primary challenges can be categorized as:

1) *Action Space Size*: Compared to heuristic rule-based approaches, learning-based algorithms, especially model-free methods, confront issues where the action space is significantly influenced by both the training traffic distributions and scaling functions. Given that all host-driven approaches utilize reactive control, the steps taken upon congestion are limited, making them a valuable resource. Thus, determining how to select actions both efficiently and judiciously remains a challenge.

2) *Computation Resources*: With data centers leveraging RDMA technology to diminish overhead from software-based data processing, it becomes imperative to contemplate how to implement learning-based algorithms in high-traffic environments like data center networks. Asynchronous handling for each flow and model compression should be prioritized in future research. An avenue worth exploring is model quantization, which has gained traction in large language models[55].

7.2.2 Next Generation of In-band Network Telemetry

From a macroscopic viewpoint, the primary challenge in monitoring traffic status within data center networks stems from the vast scale of clusters, compounded by the need to conserve resources and prevent additional congestion resulting from monitoring activities. With the advent of modern programmable switches offering advanced capabilities for packet manipulation and on-chip computation, there is potential for exploring enhanced telemetry strategies. Such approaches, encompassing flow path encoding and heightened microburst awareness, could pave the way for optimized congestion control.

Bibliography

[1] K. Hazelwood *et al.*, "Applied machine learning at facebook: A datacenter infrastructure perspective," in *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, IEEE, 2018, pp. 620–629.

[2] C. Guo *et al.*, "Rdma over commodity ethernet at scale," in *Proceedings of the 2016 ACM SIGCOMM Conference*, 2016, pp. 202–215.

[3] B. Stephens, A. L. Cox, A. Singla, J. Carter, C. Dixon, and W. Felter, "Practical dcb for improved data center networks," in *IEEE INFOCOM 2014-IEEE Conference on Computer Communications*, IEEE, 2014, pp. 1824–1832.

[4] R. Mittal *et al.*, "Revisiting network support for rdma," in *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, 2018, pp. 313–326.

[5] M. Alizadeh *et al.*, "Data center tcp (dctcp)," in *Proceedings of the ACM SIGCOMM 2010 Conference*, ser. SIGCOMM '10, New Delhi, India, 2010, 63–74, ISBN: 9781450302012.

[6] R. Mittal *et al.*, "Timely: Rtt-based congestion control for the datacenter," *SIGCOMM Comput. Commun. Rev.*, vol. 45, no. 4, 537–550, 2015, ISSN: 0146-4833.

[7] Y. Zhu, M. Ghobadi, V. Misra, and J. Padhye, "Ecn or delay: Lessons learnt from analysis of dcqcn and timely," in *Proceedings of the 12th International on Conference on Emerging Networking EXperiments and Technologies*, ser. CoNEXT '16, 2016, 313–327, ISBN: 9781450342926. DOI: 10.1145/2999572.2999593.

[8] Y. Li *et al.*, "Hpcc: High precision congestion control," in *Proceedings of the ACM Special Interest Group on Data Communication*, ser. SIGCOMM '19, Beijing, China, 2019, 44–58, ISBN: 9781450359566.

[9] C. Tessler *et al.*, "Reinforcement learning for datacenter congestion control," *SIGMETRICS Perform. Eval. Rev.*, vol. 49, no. 2, 43–46, 2022, ISSN: 0163-5999.

[10] B. He *et al.*, "Deepcc: Multi-agent deep reinforcement learning congestion control for multi-path tcp based on self-attention," *IEEE Transactions on Network and Service Management*, vol. 18, no. 4, pp. 4770–4788, 2021. DOI: 10.1109/TNSM.2021.3093302.

[11] S. Emara, F. Wang, B. Li, and T. Zeyl, "Pareto: Fair congestion control with online reinforcement learning," *IEEE Transactions on Network Science and Engineering*, vol. 9, no. 5, pp. 3731–3748, 2022. DOI: 10.1109/TNSE.2022.3185253.

[12] X. Zhong, J. Zhang, Y. Zhang, Z. Guan, and Z. Wan, "Pacc: Proactive and accurate congestion feedback for rdma congestion control," in *IEEE INFOCOM 2022 - IEEE Conference on Computer Communications*, 2022, pp. 2228–2237. DOI: 10.1109/INFOCOM48880.2022.9796803.

[13] J. Zhang, X. Zhong, Z. Wan, Y. Tian, T. Pan, and T. Huang, "Rcc: Enabling receiver-driven rdma congestion control with congestion divide-and-conquer in datacenter networks," *IEEE/ACM Transactions on Networking*, vol. 31, no. 1, pp. 103–117, 2023. DOI: 10.1109/TNET.2022.3185105.

[14] *Network simulator 3*, 2023. [Online]. Available: https://www.nsnam.org/.

[15] A. Roy, H. Zeng, J. Bagga, G. Porter, and A. C. Snoeren, "Inside the social network's (datacenter) network," in *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, 2015, pp. 123–137.

[16] R. Mittal *et al.*, "Revisiting network support for rdma," in *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, 2018, pp. 313–326.

[17] Y. Lu *et al.*, "Memory efficient loss recovery for hardware-based transport in datacenter," in *Proceedings of the First Asia-Pacific Workshop on Networking*, 2017, pp. 22–28.

[18] P. Cheng, F. Ren, R. Shu, and C. Lin, "Catch the whole lot in an action: Rapid precise packet loss notification in data center," in *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)*, 2014, pp. 17–28.

[19] D. Zats, A. P. Iyer, R. H. Katz, I. Stoica, and A. Vahdat, "Fastlane: An agile congestion signaling mechanism for improving datacenter performance," *Proceedings of SoCC*, 2013.

[20] Q. Meng and F. Ren, "Lightning: A practical building block for rdma transport control," in *2021 IEEE/ACM 29th International Symposium on Quality of Service (IWQOS)*, IEEE, 2021, pp. 1–10.

[21] G. Kumar *et al.*, "Swift: Delay is simple and effective for congestion control in the datacenter," in *Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication*, ser. SIGCOMM '20, Virtual Event, USA, 2020, 514–528, ISBN: 9781450379557.

[22] D. Katabi, M. Handley, and C. Rohrs, "Congestion control for high bandwidth-delay product networks," in *Proceedings of the 2002 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, ser. SIGCOMM '02, Pittsburgh, Pennsylvania, USA, 2002, 89–102, ISBN: 158113570X.

[23] N. Dukkipati, N. McKeown, and A. G. Fraser, "Rcp-ac: Congestion control to make flows complete quickly in any environment," in *Proceedings IEEE INFOCOM 2006. 25TH IEEE International Conference on Computer Communications*, 2006, pp. 1–5. DOI: 10.1109/INFOCOM.2006.18.

[24] J. Zhang, F. Ren, R. Shu, and P. Cheng, "Tfc: Token flow control in data center networks," in *Proceedings of the Eleventh European Conference on Computer Systems*, ser. EuroSys '16, London, United Kingdom, 2016, ISBN: 9781450342407.

[25] P. Taheri, D. Menikkumbura, E. Vanini, S. Fahmy, P. Eugster, and T. Edsall, "Rocc: Robust congestion control for rdma," in *Proceedings of the 16th International Conference on Emerging Networking EXperiments and Technologies*, ser. CoNEXT '20, Barcelona, Spain, 2020, 17–30, ISBN: 9781450379489.

[26] A. Singh *et al.*, "Jupiter rising: A decade of clos topologies and centralized control in google's datacenter network," *SIGCOMM Comput. Commun. Rev.*, vol. 45, no. 4, 183–197, 2015, ISSN: 0146-4833.

[27] 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 *Proceedings of the 11th ACM Conference on Emerging Networking Experiments and Technologies*, ser. CoNEXT '15, Heidelberg, Germany, 2015, ISBN: 9781450334129.

[28] I. Cho, K. Jang, and D. Han, "Credit-scheduled delay-bounded congestion control for datacenters," in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, ser. SIGCOMM '17, Los Angeles, CA, USA, 2017, 239–252, ISBN: 9781450346535.

[29] B. Montazeri, Y. Li, M. Alizadeh, and J. Ousterhout, "Homa: A receiver-driven low-latency transport protocol using network priorities," in *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, ser. SIGCOMM '18, Budapest, Hungary, 2018, 221–235, ISBN: 9781450355674.

[30] M. Handley *et al.*, "Re-architecting datacenter networks and stacks for low latency and high performance," in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, ser. SIGCOMM '17, Los Angeles, CA, USA, 2017, 29–42, ISBN: 9781450346535.

[31] M. Dong *et al.*, "PCC vivace: Online-Learning congestion control," in *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*, Renton, WA, 2018, pp. 343–356, ISBN: 978-1-939133-01-4.

[32] N. Jay, N. Rotman, B. Godfrey, M. Schapira, and A. Tamar, "A deep reinforcement learning perspective on internet congestion control," in *Proceedings of the 36th International Conference on Machine Learning*, K. Chaudhuri and R. Salakhutdinov, Eds., ser. Proceedings of Machine Learning Research, vol. 97, 2019, pp. 3050–3059.

[33] K. Winstein and H. Balakrishnan, "Tcp ex machina: Computer-generated congestion control," *SIGCOMM Comput. Commun. Rev.*, vol. 43, no. 4, 123–134, 2013, ISSN: 0146-4833.

[34] F. Y. Yan *et al.*, "Pantheon: The training ground for internet congestion-control research," in *2018 USENIX Annual Technical Conference (USENIX ATC 18)*, Boston, MA, 2018, pp. 731–743, ISBN: 978-1-939133-01-4.

[35] Y. Ma *et al.*, "Multi-objective congestion control," in *Proceedings of the Seventeenth European Conference on Computer Systems*, ser. EuroSys '22, Rennes, France, 2022, 218–235, ISBN: 9781450391627.

[36] N. Cardwell, Y. Cheng, C. S. Gunn, S. H. Yeganeh, and V. Jacobson, "Bbr: Congestion-based congestion control: Measuring bottleneck bandwidth and round-trip propagation time," *Queue*, vol. 14, no. 5, 20–53, 2016, ISSN: 1542-7730.

[37] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[38] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, "Deep reinforcement learning: A brief survey," *IEEE Signal Processing Magazine*, vol. 34, no. 6, pp. 26–38, 2017.

[39] Y. Bengio, A. Courville, and P. Vincent, "Representation learning: A review and new perspectives," *IEEE transactions on pattern analysis and machine intelligence*, vol. 35, no. 8, pp. 1798–1828, 2013.

[40] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, "Deterministic policy gradient algorithms," in *Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32*, ser. ICML'14, Beijing, China, 2014, I–387–I–395.

[41] S. Fujimoto, H. van Hoof, and D. Meger, *Addressing function approximation error in actor-critic methods*, 2018. arXiv: 1802.09477 [cs.AI].

[42] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," in *International conference on machine learning*, PMLR, 2015, pp. 1889–1897.

[43] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[44] L. Tan *et al.*, "In-band network telemetry: A survey," *Computer Networks*, vol. 186, p. 107 763, 2021.

[45] V. Mnih *et al.*, "Asynchronous methods for deep reinforcement learning," in *International conference on machine learning*, PMLR, 2016, pp. 1928–1937.

[46] J. Oh, V. Chockalingam, H. Lee, *et al.*, "Control of memory, active perception, and action in minecraft," in *International conference on machine learning*, PMLR, 2016, pp. 2790–2799.

[47] A. Khan, C. Zhang, N. Atanasov, K. Karydis, V. Kumar, and D. D. Lee, "Memory augmented control networks," *arXiv preprint arXiv:1709.05706*, 2017.

[48] M. Hausknecht and P. Stone, "Deep recurrent q-learning for partially observable mdps," in *2015 aaai fall symposium series*, 2015.

[49] F. Christianos, G. Papoudakis, M. A. Rahman, and S. V. Albrecht, "Scaling multi-agent reinforcement learning with selective parameter sharing," in *International Conference on Machine Learning*, PMLR, 2021, pp. 1989–1998.

[50] J. K. Terry, N. Grammel, A. Hari, L. Santos, and B. Black, "Revisiting parameter sharing in multi-agent deep reinforcement learning," 2020.

[51] D.-M. Chiu and R. Jain, "Analysis of the increase and decrease algorithms for congestion avoidance in computer networks," *Computer Networks and ISDN systems*, vol. 17, no. 1, pp. 1–14, 1989.

[52] G. F. Pfister, "An introduction to the infiniband architecture," *High performance mass storage and parallel I/O*, vol. 42, no. 617-632, p. 10, 2001.

[53] L. Meng, R. Gorbet, and D. Kulić, "Memory-based deep reinforcement learning for pomdps," in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Prague, Czech Republic, 2021, 5619–5626.

[54] H. Yin *et al.*, "Ns3-ai: Fostering artificial intelligence algorithms for networking research," in *Proceedings of the 2020 Workshop on Ns-3*, ser. WNS3 2020, Gaithersburg, MD, USA, 2020, 57–64. DOI: 10.1145/3389400.3389404. [Online]. Available: https://doi.org/10.1145/3389400.3389404.

[55] M. Cherti *et al.*, "Reproducible scaling laws for contrastive language-image learning," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023, pp. 2818–2829.