PC4: Precision Collective Communication Congestion Control for AI Cluster

Taoran Qi[1,2], Shuo Li[2], Xingqi Zou[2], Liangce Deng[2], Guodong Wei[1]
[1] School of Intergrated Circuit Science and Engineering, Beihang University, Beijing, China
[2] Department of Interconnect, Shanghai Enflame Technology Company Limited, China
iqitaoran@163.com, shuo.li@enflame-tech.com, xingqi.zou@enflame-tech.com, lance.deng@enflame-tech.com
Corresponding Author: Guodong Wei Email: jellwei@buaa.edu.cn

IEEE INFOCOM 2025 - IEEE Conference on Computer Communications | 979-8-3315-4305-1/25/$31.00 ©2025 IEEE | DOI: 10.1109/INFOCOM55648.2025.11044740

***Abstract*—The evolution of AI has driven the trend towards utilizing larger models with increasing parameter sizes, necessitating deployment in datacenters comprising tens of thousands of GPUs. However, as cluster sizes expand, the overall computational capacity of the system fails to scale linearly with the number of compute nodes. This non-linear scaling law is primarily due to decreased GPU utilization caused by network congestion, which significantly impacts model training speed. In this paper, we analyze the network topology and traffic characteristics of AI clusters and propose PC4, a precision collective communication congestion control algorithm tailored for AI clusters. PC4 optimizes congestion control based on collective communication operations (e.g., all-reduce, all-to-all) inherent in collective communication libraries (e.g., MPI, NCCL) used in distributed training. By leveraging information from these collective communication operations, we compute a base rate that enables rapid system convergence. Furthermore, PC4 employs a datacenter time synchronization mechanism, utilizing one-way delay as congestion signal. This approach allows for more precise control of link queue lengths and queuing time, thereby efficiently managing congestion. In our all-to-all evaluation, PC4 achieves a 72% reduction in tail FCT compared to DCQCN.**

***Index Terms*—Congestion control, Datacenter network, RDMA, Collective communication**

I. Introduction

Artificial intelligence (AI) has witnessed remarkable progress within the last years. The most renowned large-scale multimodal large language model, GPT-4, has demonstrated human-level performance on various professional and academic benchmarks [1]. The performance improvement of advanced large models (e.g., GPT-3, Codex, PaLM and GPT-4) is largely attributed to the continuous increase in their parameter scale. For instance, GPT-3, which was released in 2020, processed a parameter count of 175 billion. In 2023, the GPT-4 model had over one trillion parameters. The size and computational requirements of large models have expanded by more than 5000× in less than five years [2], [3], and the trend is still continuing. In order to keep up with the evolving trends in model scale development and maintain a competitive advantage, enterprises are continuously pursuing more complex model architectures and attempting to train faster on larger clusters. Many clusters for AI training now operate at the scale of more than 10000 GPUs [4]. However, as these models and clusters grow in size, they encounter significant challenges related to network congestion, which can severely impact the efficiency of AI training processes.

AI training workloads involve frequent computation and collective communication phases which results a significant increase in east-west traffic and congestion within the cluster. At production scale, advanced models utilize a mixture of model parallelism and data parallelism to scale-out to other machines [5]. This traffic consists of several types of collective communication operations like all-reduce, all-to-all, all-gather and collective-permute [3], [6]. These operations can be considered synchronous, bursty incast traffic, which exacerbates network congestion. As these operations rely on the successful delivery of all participating flows, even a single congested link can significantly impact the overall training process. Moreover, the increasing adoption of Remote Direct Memory Access (RDMA) and GPUDirect RDMA technologies further intensifies network congestion.

These congestions hinder the linear scaling of computational power and the number of GPU nodes in AI training clusters. According to a recent study, under the current widely adopted training architectures and system configurations, the communication during training takes more than 62% of total execution time [7]. We notice that currently the biggest bottleneck in AI clusters is large-scale-GPU interconnect capability rather than the computation power. The key to improving interconnect capability is to avoid congestion within the cluster. Therefore, it is crucial to have an effective congestion control mechanism to maintain high GPU utilization and reduce training time.

Numerous studies have addressed congestion in datacenters. For example, protocols such as DCQCN [8] leverages explicit congestion notification (ECN) from switches to mark congestion in the network, DX [9] utilizes one-way delay to achieve zero queuing delay, HPCC [10] employs in-network telemetry (INT) to perceive in-network congestion states, Swift [11] utilizes round-trip time (RTT) to detect network congestion status.

However, the majority of existing congestion control mechanisms are designed for general-purpose datacenter traffic patterns, rather than tailored for AI workloads. Besides, there exists a significant difference in traffic pattern and network architecture between traditional datacenters and AI clusters [12]. Therefore, current algorithms struggle to meet the comprehensive congestion control requirements in AI cluster environments.

In this paper, we propose PC4 (Precision Collective Com-

munication Congestion Control), a novel congestion control mechanism tailored for AI training clusters. PC4 is designed around two key ideas to minimize flow completion time (FCT) and latency while ensuring high system throughput.

Firstly, PC4 leverages the global traffic visibility within AI clusters. This enables PC4 to exploit the fixed set of collective communication patterns prevalent in AI training workloads and calculate a base rate for each flow, thereby facilitating fast convergence of the system. Unlike existing algorithms that require multiple iterations to determine optimal flow rates, our algorithm swiftly adapts to network changes, converging to a stable value within one RTT, thereby significantly enhancing overall system performance in AI training environments.

Secondly, PC4 employs one-way delay as the global congestion signal, which accurately quantifies the severity of congestion caused by network queuing. This enables precise control of one-way delay around a target queuing time, ensuring optimal network bandwidth utilization while preserving low latency. One-way delay measurement is achieved by the embedding timestamps in the data packets, eliminating the overhead of dedicated delay detection packets. Our approach does not require switch programming and achieve immediate and effective congestion control.

We evaluate PC4 through multiple simulation experiments. Our results demonstrate that, compared to DCQCN, PC4 provides significantly faster and more effective congestion control, maintaining stable queues and ensuring high system throughput. Under all-to-all traffic patterns, PC4 reduces tail FCT by 72%. In large-scale scenarios with 5,000 flows, PC4 achieves substantial reductions of 86% and 33% compared to Swift in the 99^{th}-p queuing delay and FCT slowdown, respectively.

The rest of our work is organized as follows. We discuss our motivation in §II, design details of PC4 in §III, performance of evaluation results is provided in §IV, discussion of the related work in §V. At last, we conclude the paper in §VI.

II. Motivation

The evolution of PC4 was driven by the characteristics of AI clusters and their workloads traffic patterns, which significantly differ from traditional datacenters. While multi-task clusters exist, this work focuses on single-task clusters due to its prevalence in large-scale model training. PC4 leverages these characteristics detailed below to implement efficient congestion control mechanisms. This tailored design enables PC4 to address the challenges posed by AI workloads, resulting in improved overall system performance and resource utilization in AI training environments. Besides, our approach does not rely on the complex programming of switches and achieves congestion control without the need for switch configurations. It is designed to be easily deployable in AI clusters.

A. Characteristics of AI Cluster

AI training workloads involves billions of parameters and large-scale sparse matrix computation, required optimized performance to minimize job completion time and maximize the utilization of expensive AI accelerators. Thus, AI cluster networks exhibit several distinguishing features compared to traditional datacenters.

Network topology: To achieve a scalable yet flat network offering high bandwidth and low end-to-end latency, AI clusters are built on a fully 1:1 non-oversubscribed environment by increasing more spine devices. This architecture is further optimized through the implementation of advanced traffic management techniques, such as Random Packet Spraying (RPS), multi-path adaptive routing and cell-based load balancing [13], [14]. These technologies effectively address imbalanced traffic loads and packet reordering issues, ensuring that the majority of traffic traverses only one-tier switches. These features result in congestion primarily occurring at host downlinks, rather than in the core of the network.

Traffic management: During collective communication, centralized controllers and schedulers are employed to manage and optimize traffic. They dynamically adjust configurations and flow paths, effectively allocating and accelerating training resources within the cluster. For example, general-purpose schedulers like Borg, YARN-like and Kubernetes (also known as K8s) [15], as well as dynamic resources-resizing scheduler like DL^2 [16] can be used. These centralized managements hold a global view of datacenter networks and enhance traffic predictability. Furthermore, each host is aware of all incoming and outgoing traffic, current collective communication operations, and baseline delays to destination address.

Time synchronization: Modern datacenters can leverage software clock synchronization systems without requiring specially designed hardware throughout the network [17]. This system can accurately measure one-way propagation time and achieve time synchronization to within 100 nanoseconds while running on standard hardware. This approach facilitates easy deployment in current AI clusters and provides the precision necessary for accurate quantification of network congestion levels.

B. AI Workloads Traffic Patterns

AI training workloads can be parallelized through various strategies, including data parallelism, pipeline parallelism, tensor parallelism, and hybrid parallelism [18]. These parallel processing approaches significantly influence the network traffic patterns in AI clusters:

Firstly, AI training processes involve intensive computations followed by data exchanges between processors. This cycle of computation and communication results in highly synchronized traffic patterns. The data exchange phase frequently results in bursty traffic characteristics. This synchronization and burstiness pose unique challenges for congestion control in AI clusters.

Secondly, unlike traditional datacenter traffic, where accurately estimating the size of every flow is challenging [19], [20], AI workloads exhibit a more predictable pattern. The total number of bytes involved in training iterations is generally known and remains constant throughout the lifetime of most training tasks [21]. Moreover, due to the large size

of model parameters, a substantial throughput is required. AI tasks typically involve a lower number of flows compared to general datacenter traffic, but each of these flows carries a significantly larger volume of data.

Finally, AI training workflows are characterized by their heavy reliance on collective communication operations, which are defined by standard communication libraries like MPI or NCCL. Different parallelism strategies result in different types of collective communication operations [5]. These operations include all-reduce, all-to-all, collective-permute, all-gather, reduce-scatter, and more. The deployment of these specific collective operations at different phases of the training process results in relatively fixed and recurrent traffic patterns throughout the entire training cycle, further enhancing the predictability of traffic patterns in AI clusters.

III. PC4 Algorithm Design

In this section, we provide a detailed description of the design principles and implementation for PC4. We propose PC4 by combining AI cluster characteristics and AI workloads traffic patterns. We avoid switch modifications to ensure compatibility with diverse cluster environments. PC4 should meet the following requirements:

- Provide low latency and high bandwidth: PC4 is engineered to support large-scale data transfers efficiently, ensuring high-performance data synchronization and throughput between training nodes. This design principle aims to minimize the FCT of network traffic, thereby reducing training time and significantly enhancing system efficiency.
- Provide good scalability: The congestion control mechanism should be as simple as possible, allowing for flexible expansion of network capacity and node count as the cluster scales. It should accommodate datacenter heterogeneity and be deployed solely on hosts, eliminating the need for network infrastructure modifications. Fairness should be ensured to avoid single-point overload and improve overall network utilization and performance.
- Provide high reliability: Ensure the stability and continuity of data transmission to minimize training interruptions and data loss that may arise from network-related issues, thereby guaranteeing robust execution of AI workloads.

Fig. 1 illustrates the transmission process and congestion control mechanism of PC4.

A. Base Rate Calculation for Fast Convergence

Traditional datacenters provide a diverse range of network services, encompassing web search, data storage and process, cloud compute, and numerous other applications [22]. However, AI training workloads, typically executed on distributed machine learning frameworks such as TensorFlow, PyTorch and PaddlePaddle, utilize fixed communication libraries with inter-accelerator communication predominantly involving collective communication operations, with all-reduce and all-to-all being the most prevalent.

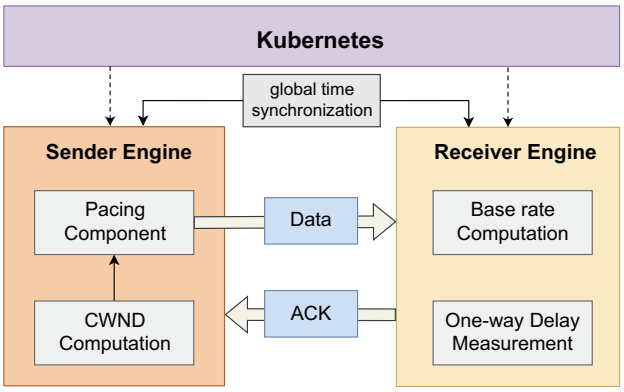


Fig. 1. Overview of PC4: Upon receiving a data packet, the receiver calculates the base rate and one-way delay and transmits this information back to the sender via an ACK packet. The sender adjusts the cwnd based on the information carried in the ACK, with the goal of maintaining the one-way delay close to a predefined target queuing time.

Algorithm 1 : RECEIVER ALGORITHM.

Input: n: the number of incoming flows. cc_mode: the collective communication mode. pkt: the header of data packet; each pkt has added informations:$tx\_time, baseline$.
Output: ACK with $base\_rate$ & $one\_way\ delay$

1: **procedure** RECEIVE DATA PACKET(pkt)
2: $base\_rate \leftarrow$ COMPUTE BASE RATE(pkt)
3: $one\_way\ delay \leftarrow now - pkt.tx\_time - pkt.baseline$
4: **function** COMPUTE BASE RATE(pkt)
5: **if** $cc\_mode = all\_reduce$ **then**
6: $base\_rate \leftarrow \frac{line\_rate}{n}$
7: **else if** $cc\_mode = alltoall\ or\ all\_gather$ **then**
8: $base\_rate \leftarrow \frac{line\_rate}{n-1}$
9: **else**
10: $base\_rate \leftarrow \frac{line\_rate}{n}$
11: **return** $base\_rate$

PC4 leverages the information provided by centralized controllers in AI clusters, enabling each worker to be aware of incoming traffic, including the number of flows and the specific collective communication method employed. Table I provides details of the base rate for common collective communication operations in AI clusters. For alternative implementations of these communication operations, such as the Ring Allreduce architecture and HD Allreduce for all-reduce operation, which differ from traditional operations, our algorithm can easily integrate new base rate calculations.

The pseudo-code for the receiver is specified in Algorithm 1. Upon receiving a data packet, the receiver computes the base rate and incorporates it into the ACK header (Lines 1-2 and 4-11 of Algorithm 1). For the sender, it maintains a record of the base rate for each flow in addition to the congestion window (cwnd). A difference between the base rate carried in the ACK and the rate stored in the sender register indicates a change in traffic topology, such as the addition of new flows or the completion of existing flows. In such cases, the sender

TABLE I
BASE RATE FOR COMMON COMMUNICATION OPERATIONS.

cc_mode	all-reduce	all-to-all	all-gather	other
base rate	line rate/n	line rate/(n-1)	line rate/(n-1)	line rate/n

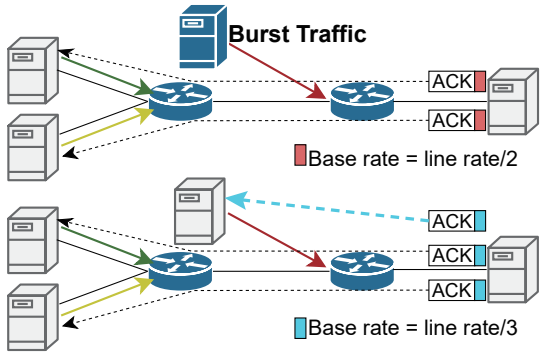


Fig. 2. When traffic fluctuations occur, such as burst traffic or the end of flows, the receiver immediately modifies the base rate in the ACK and sends it back to the sender.

promptly adjusts the cwnd and updates $base\_record$ with the new base rate. Given that the majority of traffic in AI clusters consists of long flows, we anticipate that base rate renewal will be an infrequent occurrence. However, the effectiveness of this approach becomes particularly evident during cluster reconfiguration or workload changes.

Fig. 2 shows that the base rate embedded in the ACK serves as the reference rate for the sender during network stability. When bursty traffic enters the network, the receiver detects these incoming traffic fluctuations and modifies the base rate in the ACK. The pseudo-code for the sender is specified in Algorithm 2. Upon detecting network changes, the sender swiftly adjusts the cwnd to $baserate \times RTT$ (Lines 4-7 and 12 of Algorithm 2). For the sender, subsequently fine-tuning occurs based on congestion signals to achieve precise control, which we will elaborate on in the next section.

PC4's adaptive base rate calculation mechanism effectively manages congestion by rapidly converging to a stable value when a new flow arrives or an existing flow completes. This rapid convergence, often achieved in a single iteration, significantly reduces both FCT and tail latency. Compared to other algorithms that require multiple iterations to stabilize after network changes, PC4 demonstrates superior responsiveness. By leveraging the unique characteristics of AI workloads and communication patterns, PC4 ultimately yields improved network performance through rapid convergence.

B. One-way Delay as an Accurate Congestion Signal

An accurate and simple congestion signal is a crucial component of congestion control algorithms. We have found that using one-way delay as a congestion signal is highly effective. It avoids potential errors caused by using RTT as congestion signal, such as return path delay and NIC processing time.

Algorithm 2 : SENDER ALGORITHM.

Input: ACK with $base\_rate$ & $one\_way\ delay$
Output: $cwnd$, $pacing\_delay$

1: **Parameter:** $base\_recorded$=0, hai: hyper additive increment, ai: additive increment, β: multiplicative decrease constant, max_mdf: maximum multiple decrease factor
2: **procedure** RECEIVE ACK(ack)
3: $target\_qtime \leftarrow$ TARGET DELAY() ▷ See §III-C
4: **if** $ack.base\_rate \neq base\_recorded$ **then**
5: $base\_recorded \leftarrow ack.base\_rate$
6: $tx\_rate \leftarrow ack.base\_rate$
7: $t\_last\_adjust \leftarrow now$
8: **else**
9: **if** $now - t\_last\_adjust \geq adjust\_interval$ **then**
10: $tx\_rate \leftarrow$ COMPUTE RATE(ack)
11: $t\_last\_adjust \leftarrow now$
12: $cwnd \leftarrow tx\_rate \cdot rtt$
13: **if** $cwnd < 1$ **then**
14: $pacing\_delay \leftarrow \frac{rtt}{cwnd}$
15: **else**
16: $pacing\_delay \leftarrow 0$
17: **procedure** ACK TIMEOUT( )
18: $cwnd \leftarrow cwnd \cdot (1 - max\_mdf)$
19: $base\_recorded \leftarrow 0$
20: **function** COMPUTE RATE(ack)
21: **if** $ack.one\_way\ delay = 0$ **then**
22: $tx\_rate \leftarrow tx\_rate + hai$
23: **else if** $0 < ack.one\_way\ delay < target\_qtime$ **then**
24: $tx\_rate \leftarrow tx\_rate + ai$
25: **else**
26: $tx\_rate \leftarrow tx\_rate \cdot max(1 - max\_mdf,$ $1\text{-}\beta \cdot (\frac{ack.one\_way\ delay - target\_qtime}{ack.one\_way\ delay + baseline}))$ ▷ Multiplicative Decrease(MD)
27: **return** tx_rate

Additionally, it does not require switch programming capabilities and provides a multi-bit signal that precisely reflects the congestion degree in the current packet path network. This means that the congestion control algorithm can operate without requiring any specific support or programming from the switches in the cluster, reducing the complexities of deployment and debugging.

Fig. 3 illustrates that RTT can be decomposed four components: *Forward Path Delay*, *Receiver Processing Delay*, *Reverse Path Delay*, and *Sender Processing Delay*. By measuring the Forward Path Delay, we can effectively ignore the potential impact of reverse path delay on RTT and increase the granularity of control algorithms.

The key challenge in using one-way delay as congestion signal is accurately measuring it in AI clusters, which can be difficult due to the much lower propagation delay compared to traditional datacenters. Fortunately, modern datacenter networks offer precise time synchronization mechanisms at a global ∼100 nanoseconds level without requiring specialized

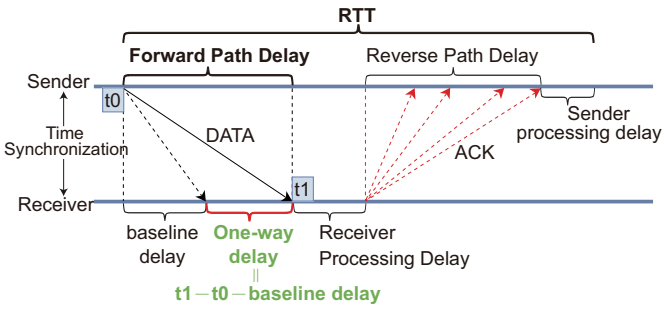


Fig. 3. The components of RTT and Forward Path Delay, as well as the method for measuring one-way delay.

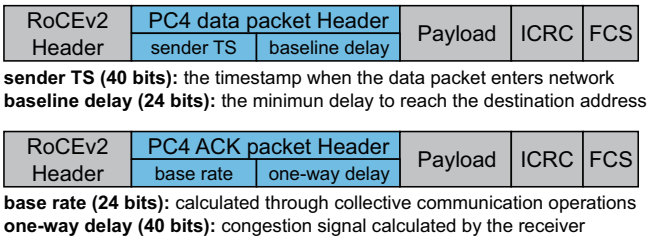


Fig. 4. The data packet and ACK packet format of PC4

hardware support [17], enabling us to measure congestion at the bottleneck link without relying on RTT. As a result, measuring the Forward Path Delay in the network becomes feasible.

Fig. 3 shows that the Forward Path Delay can be further divided into baseline delay and queuing delay (which we refer to as one-way delay in this context). We focus on the one-way delay, which directly reflects the state of congestion in the network and provides accurate information about the queue length of switches along the path. The baseline delay represents the theoretical time it takes for each data packet to reach its destination without queuing, including the sum of propagation delays and the serialization delay at the NIC and switch. In AI clusters, the baseline delay from end to end is known because it can be calculated based on the network topology provided by the schedulers and add to the header of the transmitted data packets.

In our implementation, the sender marks a timestamp $t0$ in the data packet as it leaves the NIC stack and enters the network. When the receiver receives the data packet, it records the timestamp $t1$. The Forward Path Delay is then calculated as $t1 - t0$. By subtracting the baseline delay from this value, we obtain the desired one-way delay of this data packet in the network, which is $(t1 - t0) - baseline\ delay$. Finally, the obtained one-way delay and base rate are added to the ACK and returned to the sender together. Fig. 4 illustrates the data packet and ACK packet formats utilized in PC4.

C. Target Queuing Time Control Link Utilization

In PC4, the cwnd adjustment module at the sender employs a modified Additive-Increase Multiplicative-Decrease (AIMD) algorithm. Research has shown that if there is no queuing in network scenarios, it can lead to lower network utilization [23]. To enhance resource utilization, we introduce a target queuing time, referred to as target qtime. By comparing the one-way delay with the target qtime, we evaluate and adjust the current cwnd. The goal is to adjust the one-way delay to be close to the target qtime.

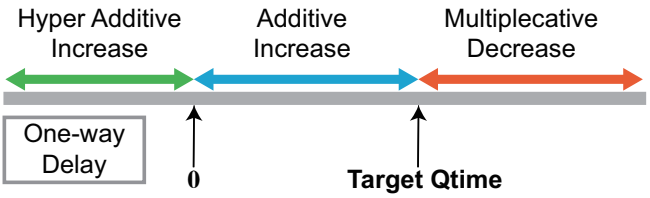


Fig. 5. AIMD with target qtime threshold

The rate adjustment process in PC4 is designed to balance responsiveness and stability. Our simulations revealed that adjusting cwnd for every ACK is suboptimal, as it takes at least one RTT for the data packets sent through the new cwnd to be received. Frequent adjustments can lead to overreaction and link instability, causing oscillations between congestion and low utilization. To mitigate this, we set an adjust interval, during which the cwnd remains constant unless the base rate changes (Lines 9-12 of Algorithm 2). This interval is set to exceed one RTT, ensuring sufficient time for the data packets with the updated cwnd to propagate.

When the conditions for adjustment are met, the sender compares one-way delay in the ACK with the target qtime (Lines 20-27 of Algorithm 2). Fig. 5 shows the three scenarios we considered:

a) Zero one-way delay: Indicates no queuing and congestion for the path of this packet. There is still capacity to accommodate more packets to achieve the target qtime. In this scenario, the sender initiates a hyper additive increase phase to improve network utilization.

b) One-way delay between zero and target qtime: It indicates that there is queuing in the network, but it is still relatively low. In this scenario, the sender enters an additive increase phase, probing for remaining bandwidth.

c) One-way delay exceeding target qtime: It suggests that there is significant queuing in the network. In this scenario, the cwnd is decreased multiplicatively, with the decrease depending on the ratio of the one-way delay minus the target qtime to the sum of the one-way delay and the baseline delay.

To ensure optimal performance and robustness in AI clusters, we scale the target qtime threshold according to network topology and competing flows. In large-scale datacenter topologies, diverse flow paths inherently result in varying baseline delays. PC4 utilizes the known baseline delay as a reference point for setting the target qtime threshold. Thus, PC4 ensures fairness for flows traversing both long and short paths, enables more precise control of link utilization, and enhances adaptability to diverse network environments.

In AI clusters, we model the traffic as synchronized flows.

For the convenience of description, we assume a link with N synchronized flows, where the average queue length grows with $O(N)$. This differs from the buffer space growth trend of unsynchronized flows discussed in reference [24], which follows an $O(\sqrt{N})$ growth trend. To maintain consistent buffer space utilization, we decrease the target qtime with an $O(\frac{1}{N})$ reduction. In practical deployment, lower target qtime can be set for latency-sensitive flows, while higher values can be set for throughput-sensitive flows. Such configurations should be tailored to the specific network conditions.

Given the large scale of AI clusters with tens of thousands of flows, PC4 allows the minimum cwnd to be less than one packet, with a lower bound of 0.0001 packets. When the cwnd falls below 1, the sender calculates the inter-packet delay as $\frac{RTT}{cwnd}$ and utilizes it to pace packet into the network (Lines 13–16 of Algorithm 2). This approach enables PC4 to handle congestion control at an extremely large scale.

D. ACK Timeout and Loss Recovery

Modern datacenters widely adopt RDMA technology to meet the low-latency and high-bandwidth requirements of AI training while balancing cost and performance. These environments typically employ the RoCEv2 protocol and enable the default priority flow control (PFC) to ensure lossless network operation. However, in scenarios where both congestion control and PFC mechanisms fail simultaneously, or in the event of physical link failures, packet loss can occur. Such loss triggers the "go-back-N" retransmission mechanism, potentially resulting in reduced goodput and poor efficiency in RDMA networks.

PC4 is designed to minimize packet loss by rapidly converging each flow to a stable value within one RTT. Nevertheless, to enhance system robustness against physical-level failures, PC4 incorporates a timeout mechanism. This feature prevents widespread congestion that could arise from the absence of ACKs carrying crucial base rate and one-way delay information.

Although the reverse path delay of ACKs does not affect the accuracy of congestion control, if no ACK packet is received within a retransmission timeout (RTO) after sending a data packet, we consider it as either network congestion or packet loss. In either case, PC4 promptly reduces the cwnd by multiplying it by the maximum MD factor, and resets the corresponding flow's $base\_record$ to 0. This reduction continues until an ACK carrying base rate and adjust rate is received, and then the cwnd can be adjusted to quickly adapt to or predict widespread congestion (Lines 17-19 of Algorithm 2).

Due to the low packet loss rate in PC4, it does not become a bottleneck in the system during packet loss recovery and timeout situations. This characteristic facilitates maintaining low job completion time and good tail latency. PC4 does not explicitly delay ACKs to react more quickly to congestion. The mechanism of sending merged ACKs in the protocol does not affect congestion control.

IV. Experimental Results

We conducted a series of controlled experiments to evaluate the performance of PC4. Our experiments were conducted on a simulation testing platform developed by Enflame, which is used for accurately simulating real chip performance and AI cluster network behavior. We validated our platform by comparing its output with our laboratory results and the data from literature. After validation, we confirmed that our simulation environment closely resembles the realistic environment. We utilized this platform to simulate and analyze the performance of PC4 under various topologies.

All links were 100Gbps unless otherwise specified. We selected DCQCN for comparison due to its widespread adoption and proven efficacy in modern datacenter deployments. The parameters used in our tests were provided by NVIDIA/Mellanox [25] and the experiments were conducted on a testbed configured to run industry-standard benchmarks. The testbed topology was designed to mimic multiple RDMA blocks within a Pod. It consisted of two 12.8Tbps spine switches, 20 leaf switches, and 32 servers (each server had eight 100Gbps NICs). Each link had 1us propagation delay, resulting in a 4μs base RTT within a rack.

A. Significance of Base Rate

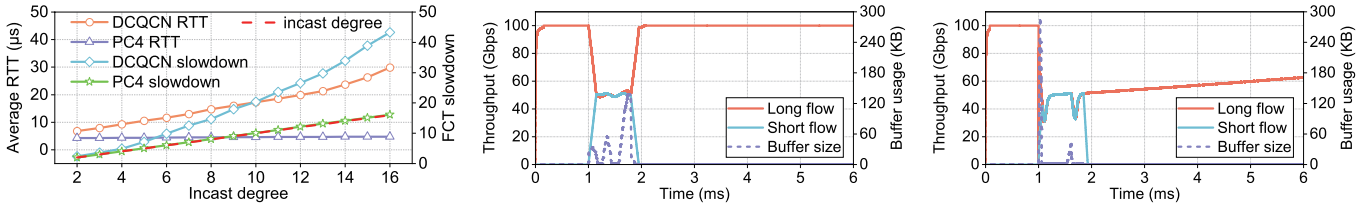
The concept of base rate plays a crucial role in congestion control for AI clusters. We conducted two experiments below to validate its effectiveness and impact on PC4's performance.

PC4 has lower RTT and FCT slowdown: The first experiment examined PC4's performance under the common parameter server (PS) framework, where all workers synchronize with the parameter server simultaneously [26], creating an n-to-1 incast traffic pattern—a major congestion scenario in deep learning. We varied the degree of incast from 2 to 16, with each sender initiating a long-running flow simultaneously. This experimental design allowed us to observe and analyze the behaviors as the degree of incast increased throughout the network.

Our metrics of interest were FCT slowdown[1] and average RTT, as lower FCT slowdown indicates more efficient network utilization, improved system efficiency, and increased GPU utilization in large-scale clusters, while reduced RTT suggests decreased queuing time and congestion in the system.

Fig. 6(a) showed the comparison between PC4 and DCQCN. We observed that PC4's RTT and system throughput does not vary with the degree of congestion. Due to the utilization of the base rate, PC4 rapidly adapted to the convergence cwnd within one RTT, maintaining a controlled queue length and average RTT close to the base RTT. Moreover, PC4 maintained high throughput, with FCT slowdown approximating the number of incast flows, indicating near-ideal network state. In contrast, DCQCN exhibited increased queuing, higher latency, and higher FCT slowdown due to its reliance on queue buildup

[1]"FCT slowdown" means a flow's actual FCT normalized by its ideal FCT when the network only has this flow.



(a) Incast traffic (b) Long-short PC4 (c) Long-short DCQCN

Fig. 6. Comparing PC4 and DCQCN with incast and long-short traffic

to the ECN threshold and inability to clearly indicate network congestion degree.

PC4 has better and faster convergence: In our second experiment, we simulated a scenario with a long-lived flow sent at line rate and a 5MB short flow joining the same path after 1ms. PC4 immediately shared bandwidth between long and short flows upon the short flow's arrival, and after the short flow's completion, the long flow quickly recovered its original line rate. DCQCN, however, failed to recover to line rate within 4ms after the short flow ended.

Fig. 6(b) and Fig. 6(c) illustrated the behaviors of PC4 and DCQCN with long-short traffic. The results demonstrated that PC4 achieved a 78% improvement in bandwidth utilization compared to DCQCN. This superior performance is attributed to PC4's feedback mechanism, which does not solely rely on the queue length but also takes into account the base rate, enabling faster convergence and more efficient bandwidth utilization. Furthermore, the reduced convergence time translates directly to improved overall system efficiency, minimizing time spent in suboptimal network states.

B. Significance of Adjust Rate

In complex traffic scenarios, multiple NICs at the sender side may experience varying degrees of burstiness. Consequently, relying solely on the base rate may not sufficiently address traffic fluctuations and bursts to achieve accurate convergence values. To mitigate this issue, we implemented a dynamic adjustment of the adjust rate.

To validate the role of the adjust rate, a crucial component of our congestion control mechanism, we conducted experiments using an all-to-all traffic pattern, which is commonly happened in the shuffle step of MapReduce [27]. Our simulation involved an 8×8 all-to-all workload, where each connection executed 8 tasks, with each task transmitting 50MB to all destination addresses. This configuration resulted in a total of 448 ($7 \times 8 \times 8$) flows within the cluster.

Fig. 7 illustrated the CDF of flow completion time for both DCQCN and PC4, with and without the adjust rate feature enabled. The result demonstrated that enabling the adjust rate feature in PC4 reduced the 99^{th}-p FCT and tail FCT by 31% and 34%, respectively, compared to when this feature was disabled. This improvement can be attributed to the fact that relying solely on the base rate for packet transmission may be susceptible to bursts or uneven polling, potentially leading to suboptimal bandwidth utilization. By fine-tuning the cwnd

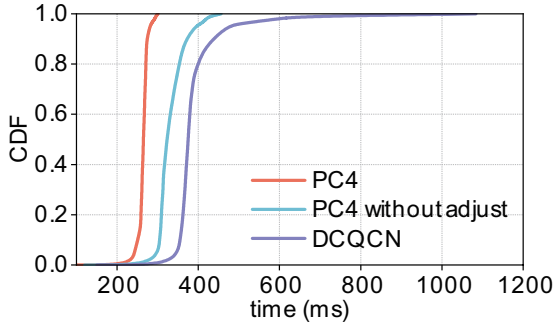


Fig. 7. CDF of all-to-all traffic pattern flow completion time

according to the network congestion degree, PC4 achieves lower job completion time.

Furthermore, when compared to DCQCN, PC4 exhibited performance improvements of 55% and 72% in the 99^{th}-p FCT and tail FCT, respectively. This significant enhancement can be attributed to PC4's ability to converge rapidly during traffic bursts without requiring continuous iterations like DCQCN, thereby reducing network fluctuations during the convergence process. PC4 demonstrates superior performance in handling complex traffic patterns, particularly in scenarios with multiple NICs and varying degrees of burstiness.

C. Function of Target Qtime

The target qtime parameter plays a crucial role in the performance of PC4, significantly impacting system stability and robustness. To investigate its effects, we designed an experiment using 8 senders, each with 8 identical flows, sending 100KB of data in an all-reduce collective communication operation to the same receiver.

By adjusting the target qtime parameter, we aimed to observe its influence on throughput, FCT slowdown, and latency, with the ultimate goal of determining an appropriate setting. To isolate the impact of target qtime, we intentionally excluded the computation of the base rate, relying solely on the AIMD adjust rate mechanism for this analysis.

Our simulation involved varying the target qtime from 0μs to 80μs. Notably, we observed that the actual one-way delay closely matched the target qtime. This correlation indicates that our algorithm effectively controls queue length and system utilization. Fig. 8 illustrates the performance metrics for different target qtime values.

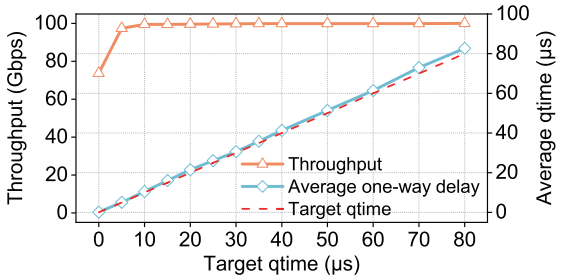


Fig. 8. Target qtime controls throughput and average one-way delay in 64 flows incast

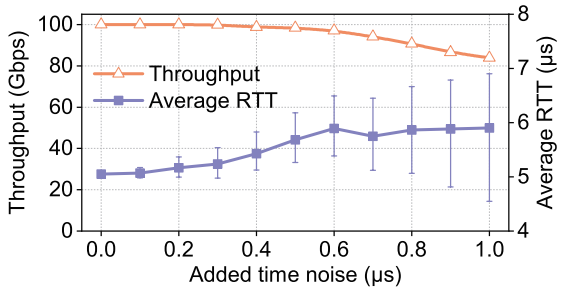


Fig. 9. The impact of time synchronization noise on PC4

TABLE II
PERFORMANCE OF PC4 UNDER 5000-TO-1 INCAST

Metric	Swift	HPCC	PC4
99^{th}-p Queuing(msec)	23.543	23.066	4.729
99^{th}-p FCT slowdown	7017	5037	5010

The analysis revealed interesting dynamics at different target qtime settings. Setting the target qtime to 0 effectively eliminated queuing in the system. However, this approach proved suboptimal, as even minimal queuing triggered a multiplicative decrease in cwnd, resulting in lower bandwidth utilization. As we increased the target qtime, we observed improved system utilization, albeit at the cost of increased queuing time.

The optimal setting depends on factors such as cluster size and the number of flows. We recommend an initial adjustment of the target qtime to achieve maximum throughput, followed by a gradual reduction to optimize latency while maintaining bandwidth utilization. This approach can assist in identifying appropriate target qtime configurations for various network environments, thus achieving optimal performance.

D. Large-scale Incast

In AI clusters, large-scale incast scenarios frequently occur due to flow synchronization effects. To address this challenge, PC4 sets the minimum cwnd to be less than one packet and controls the interval between pacing data packets. We evaluated PC4's performance in a cluster environment using a 5000-to-1 incast scenario, where 50 senders simultaneously initiated 100 identical flows to a single destination receiver. We measured the 99^{th}-p queuing and FCT slowdown.

Table II presented the results [23], demonstrating PC4's ability to achieve line-rate throughput with zero packet loss and low latency in this demanding scenario. PC4's superior performance can be attributed to its ability to accurately perceive network traffic and establish an appropriate base rate. This capability led to significant reductions in tail latency, with PC4 outperforming Swift and HPCC by 86% and 81%, respectively. Furthermore, the 99^{th}-p FCT slowdown for PC4 was measured at 5010, indicating near-ideal congestion control even under large-scale incast conditions.

The efficacy of PC4 extends beyond performance metrics to scalability considerations. The base rate calculation mechanism employed by PC4 maintains its efficiency regardless of the cluster size, ensuring that the approach scales well with increasing numbers of nodes involved in the communication. This scalability feature is particularly crucial in the context of ever-growing AI training clusters.

E. Impact of Time Synchronization Noise

Time synchronization noise on one-way delay measurements is a critical factor that can affect the performance of congestion control algorithms. This noise encompasses various elements, including the overhead of the protocol stack and kernel scheduling [28]. To evaluate the robustness of PC4 in the face of such challenges, we conducted an experiment to assess the impact of transient queues, timer inaccuracy, and datacenter time synchronization noise on its performance. We introduced controlled noise in a 10-v-1 traffic to the measured one-way delay and observed its effects on the congestion control mechanism. Specifically, we set the target qtime to be 1µs and added random noise uniformly distributed within the range of $[-\varepsilon, \varepsilon]$ µs to the queuing time, with ε values ranging from 0 to 1.

Fig. 9 illustrated the results of this experiment, revealed the average throughput and RTT of the system under various noise levels. Our findings indicated that an average noise of 1 microsecond significantly impaired system performance, leading to reduced link utilization and increased queue length. Conversely, noise levels below 400 nanoseconds were observed to have minimal impact on system performance. Therefore, an accurate time synchronization mechanism is crucial for the effectiveness of PC4.

F. Fairness Analysis

To evaluate the fairness characteristics of PC4, we conducted an experiment using a dumbbell topology. We added or removed a new flow every 10 milliseconds and measured the throughput of each flow. The results, as illustrated Fig. 10, demonstrate PC4 can rapidly adapt to traffic fluctuations and converge to a stable state.

Notably, the competing flows at the bottleneck exhibited equitable sharing of bandwidth resources, indicating a high degree of fairness in resource allocation. This fairness can be attributed to two key mechanisms within PC4: the combination

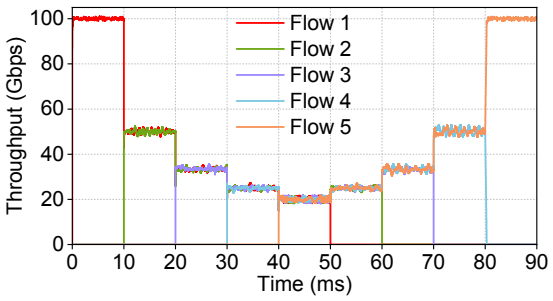


Fig. 10. PC4's fair allocation of throughput among five flows

of base rate computation and the well-established AIMD mechanism, which enables fair allocation of bandwidth within a short time scale, and the scaling of target qtime based on both baseline delay and the number of flows, ensuring fairness for flows with varying path lengths. PC4 not only ensures efficient utilization of available bandwidth but also promotes equitable resource distribution among competing flows. This capability is particularly crucial in AI research and development environments, where multiple processes may compete for network resources simultaneously.

V. Related Work

Our approach draws inspiration from a variety of existing research on datacenter congestion control algorithms and the characteristics of AI cluster infrastructure. Here we summarize several related works.

Congestion control for high-speed DCN: TIMELY uses hardware-measured RTT as a congestion signal and uses RTT gradients to adjust transmission rates may not converge stably [29]. DX is one of the early proposals to utilize one-way queuing delay to detect congestion for datacenters [9]. However, its measurement method for latency can only achieve an accuracy of about 1 microsecond, which does not meet the requirements of AI clusters. Swift compares the RTT with the target delay for cwnd adjustment, it remains susceptible to inaccurate congestion assessments due to ACK queuing on the reverse path [11]. PC4 addresses these limitations by providing accurate congestion detection based on one-way delay measurements.

DCQCN relies on ECN marked packets in switches that exceed the queuing threshold [8]. HPCC utilizes INT to obtain precise load information in the network, enabling fast convergence and low latency [10]. Bolt uses granular congestion signal provided by leveraging programmable switches and generating feedback directly from congested switches back to the sender [23]. Poseidon utilizes INT to implement congestion control specifically for the actual bottleneck hop [30]. In contrast, PC4 is a host-based congestion control approach that does not rely on switch configurations, making it easier to deploy.

While not directly addressing congestion control, scheduling-based approaches like Fastpass proposes that each sender should delegate control to a centralized arbiter to determine the transmission time and path for every packet, which is similar to AI clusters [31]. This scheduling-based congestion control approach has also inspired PC4 to utilize AI traffic characteristics for congestion control.

Datacenter time synchronization protocols: NTP utilizes a symmetric architecture and can maintain timing accuracy within a few milliseconds [32]. DTP leverages a specialized physical layer to achieve precision of achieving time precision at the level of millisecond, but is limited by the modified PHY [33]. Fault-tolerant clock synchronization systems such as Sundial achieves ∼100 nanoseconds time-uncertainty bound under various failure scenarios [34]. Sirius is an optically-switched network-based solution for datacenters, enabling precision at approximately ten nanoseconds [35]. Furthermore, PC4 achieves time synchronization to within 100 nanoseconds using software systems without the need for specially designed hardware [17]. In the past few years, the continuously improving time synchronization accuracy makes it more convenient to measure one-way delay for congestion control.

AI job schedulers: There has been many schedulers for AI training jobs and collective communication. TACCL proposes a framework that optimizes collective communication algorithms [36]. TicTac guarantees near-optimal overlap of computation and its communication [37]. Syndicate proposes minimizing communication bottlenecks and accelerating training for large-scale models and interconnection [5]. All of these approaches in this domain focus on optimizing collective communication transfers, while PC4 further enhances these strategies by incorporating congestion control through the scheduler's framework.

VI. Conclusion

This paper presented PC4, a precision congestion control algorithm for AI clusters. By calculating a base rate for each flow based on collective communication operations, PC4 achieves rapid convergence and significantly improves network utilization. Furthermore, PC4 leverages one-way delay as congestion signal, enabling accurate congestion detection while mitigating the impact of noisy reverse path measurements and reducing tail latency. PC4 achieves precise control over queuing time on network links by comparing one-way delay with a target qtime. Furthermore, by being implemented entirely in the host, PC4 facilitates straightforward deployment in AI clusters without incurring additional network overhead. Our evaluation results demonstrate that PC4 achieves high bandwidth utilization, low latency, and fairness among competing flows. PC4's rapid convergence significantly reduces flow completion time, demonstrating the effectiveness of integrating congestion control with AI cluster characteristics and AI workload traffic patterns.

Acknowledgment

This work is supported by the National Natural Science Foundation of China (Nos. 92164206, 52261145694, 62104230), the Beijing Municipal Natural Science Foundation (Z230004) and Beijing S&T Project (Z221100007722030).

REFERENCES

[1] J. Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya, F. L. Aleman, D. Almeida, J. Altenschmidt, S. Altman, S. Anadkat *et al.*, "Gpt-4 technical report," *arXiv preprint arXiv:2303.08774*, 2023.

[2] A. Chowdhery, S. Narang, J. Devlin, M. Bosma, G. Mishra, A. Roberts, P. Barham, H. W. Chung, C. Sutton, S. Gehrmann *et al.*, "Palm: Scaling language modeling with pathways," *Journal of Machine Learning Research*, vol. 24, no. 240, pp. 1–113, 2023.

[3] D. Mudigere, Y. Hao, J. Huang, Z. Jia, A. Tulloch, S. Sridharan, X. Liu, M. Ozdal, J. Nie, J. Park *et al.*, "Software-hardware co-design for fast and scalable training of deep learning recommendation models," in *Proceedings of the 49th Annual International Symposium on Computer Architecture*, 2022, pp. 993–1011.

[4] Z. Jiang, H. Lin, Y. Zhong, Q. Huang, Y. Chen, Z. Zhang, Y. Peng, X. Li, C. Xie, S. Nong *et al.*, "MegaScale: Scaling large language model training to more than 10,000 GPUs," in *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*, 2024, pp. 745–760.

[5] K. Mahajan, C.-H. Chu, S. Sridharan, and A. Akella, "Better together: Jointly optimizing ML collective scheduling and execution planning using SYNDICATE," in *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, 2023, pp. 809–824.

[6] D. Lepikhin, H. Lee, Y. Xu, D. Chen, O. Firat, Y. Huang, M. Krikun, N. Shazeer, and Z. Chen, "Gshard: Scaling giant models with conditional computation and automatic sharding," *arXiv preprint arXiv:2006.16668*, 2020.

[7] M. Wang, C. Meng, G. Long, C. Wu, J. Yang, W. Lin, and Y. Jia, "Characterizing deep learning training workloads on alibaba-pai," in *2019 IEEE international symposium on workload characterization (IISWC)*. IEEE, 2019, pp. 189–202.

[8] Y. Zhu, H. Eran, D. Firestone, C. Guo, M. Lipshteyn, Y. Liron, J. Padhye, S. Raindel, M. H. Yahia, and M. Zhang, "Congestion control for large-scale rdma deployments," *ACM SIGCOMM Computer Communication Review*, vol. 45, no. 4, pp. 523–536, 2015.

[9] C. Lee, C. Park, K. Jang, S. Moon, and D. Han, "Dx: Latency-based congestion control for datacenters," *IEEE/ACM Transactions on Networking*, vol. 25, no. 1, pp. 335–348, 2016.

[10] Y. Li, R. Miao, H. H. Liu, Y. Zhuang, F. Feng, L. Tang, Z. Cao, M. Zhang, F. Kelly, M. Alizadeh *et al.*, "Hpcc: High precision congestion control," in *Proceedings of the ACM special interest group on data communication*, 2019, pp. 44–58.

[11] G. Kumar, N. Dukkipati, K. Jang, H. M. Wassel, X. Wu, B. Montazeri, Y. Wang, K. Springborn, C. Alfeld, M. Ryan *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*, 2020, pp. 514–528.

[12] Q. Hu, P. Sun, S. Yan, Y. Wen, and T. Zhang, "Characterization and prediction of deep learning workloads in large-scale gpu datacenters," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2021, pp. 1–15.

[13] J. Hu, J. Huang, W. Lv, Y. Zhou, J. Wang, and T. He, "Caps: Coding-based adaptive packet spraying to reduce flow completion time in data center," *IEEE/ACM Transactions on Networking*, vol. 27, no. 6, pp. 2338–2353, 2019.

[14] A. Dixit, P. Prakash, Y. C. Hu, and R. R. Kompella, "On the impact of packet spraying in data center networks," in *2013 proceedings ieee infocom*. IEEE, 2013, pp. 2130–2138.

[15] "Kubernetes," https://kubernetes.io/.

[16] Y. Peng, Y. Bao, Y. Chen, C. Wu, C. Meng, and W. Lin, "Dl2: A deep learning-driven scheduler for deep learning clusters," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 8, pp. 1947–1960, 2021.

[17] Y. Geng, S. Liu, Z. Yin, A. Naik, B. Prabhakar, M. Rosenblum, and A. Vahdat, "Exploiting a natural network effect for scalable, fine-grained clock synchronization," in *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*, 2018, pp. 81–94.

[18] R. Mayer and H.-A. Jacobsen, "Scalable deep learning on distributed infrastructures: Challenges, techniques, and tools," *ACM Computing Surveys (CSUR)*, vol. 53, no. 1, pp. 1–37, 2020.

[19] M. Alizadeh, S. Yang, M. Sharif, S. Katti, N. McKeown, B. Prabhakar, and S. Shenker, "pfabric: Minimal near-optimal datacenter transport," *ACM SIGCOMM Computer Communication Review*, vol. 43, no. 4, pp. 435–446, 2013.

[20] 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*, 2015, pp. 1–12.

[21] S. Rajasekaran, S. Narang, A. A. Zabreyko, and M. Ghobadi, "Mltcp: Congestion control for dnn training," *arXiv preprint arXiv:2402.09589*, 2024.

[22] A. Greenberg, J. Hamilton, D. A. Maltz, and P. Patel, "The cost of a cloud: research problems in data center networks," pp. 68–73, 2008.

[23] S. Arslan, Y. Li, G. Kumar, and N. Dukkipati, "Bolt: Sub-RTT congestion control for Ultra-Low latency," in *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, 2023, pp. 219–236.

[24] G. Appenzeller, I. Keslassy, and N. McKeown, "Sizing router buffers," *ACM SIGCOMM Computer Communication Review*, vol. 34, no. 4, pp. 281–292, 2004.

[25] "Dcqcn parameters." 2023, https://enterprise-support.nvidia.com/s/article/dcqcn-parameters.

[26] C. Chen, W. Wang, and B. Li, "Round-robin synchronization: Mitigating communication bottlenecks in parameter servers," in *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*. IEEE, 2019, pp. 532–540.

[27] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[28] C. Luo, H. Gu, X. Yu, L. Zhu, Z. Zhou, H. Zhang, and W. Hou, "Alarm: An adaptive routing algorithm based on one-way delay for infiniband," *IEEE Transactions on Network Science and Engineering*, 2024.

[29] 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*, 2016, pp. 313–327.

[30] W. Wang, M. Moshref, Y. Li, G. Kumar, T. E. Ng, N. Cardwell, and N. Dukkipati, "Poseidon: Efficient, robust, and practical datacenter CC via deployable INT," in *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, 2023, pp. 255–274.

[31] J. Perry, A. Ousterhout, H. Balakrishnan, D. Shah, and H. Fugal, "Fastpass: A centralized" zero-queue" datacenter network," in *Proceedings of the 2014 ACM conference on SIGCOMM*, 2014, pp. 307–318.

[32] D. L. Mills, "Internet time synchronization: the network time protocol," *IEEE Transactions on communications*, vol. 39, no. 10, pp. 1482–1493, 1991.

[33] K. S. Lee, H. Wang, V. Shrivastav, and H. Weatherspoon, "Globally synchronized time via datacenter networks," in *Proceedings of the 2016 ACM SIGCOMM Conference*, 2016, pp. 454–467.

[34] Y. Li, G. Kumar, H. Hariharan, H. Wassel, P. Hochschild, D. Platt, S. Sabato, M. Yu, N. Dukkipati, P. Chandra *et al.*, "Sundial: Fault-tolerant clock synchronization for datacenters," in *14th USENIX symposium on operating systems design and implementation (OSDI 20)*, 2020, pp. 1171–1186.

[35] H. Ballani, P. Costa, R. Behrendt, D. Cletheroe, I. Haller, K. Jozwik, F. Karinou, S. Lange, K. Shi, B. Thomsen *et al.*, "Sirius: A flat datacenter network with nanosecond optical switching," 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*, 2020, pp. 782–797.

[36] A. Shah, V. Chidambaram, M. Cowan, S. Maleki, M. Musuvathi, T. Mytkowicz, J. Nelson, O. Saarikivi, and R. Singh, "TACCL: Guiding collective algorithm synthesis using communication sketches," in *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, 2023, pp. 593–612.

[37] S. H. Hashemi, S. Abdu Jyothi, and R. Campbell, "Tictac: Accelerating distributed deep learning with communication scheduling," *Proceedings of Machine Learning and Systems*, vol. 1, pp. 418–430, 2019.