PARALEON: Automatic and Adaptive Tuning for DCQCN Parameters in RDMA Networks

Ziteng Chen[1], Menghao Zhang[2,6], Guanyu Li[3], Jiahao Cao[4], Yang Jing[5], Mingwei Xu[4,6], Renjie Xie[4], He Liu[5], Fangzheng Jiao[5], Xiaohe Hu[5]
[1]Southeast University [2]Beihang University [3]Unaffiliated [4]Tsinghua University [5]Infrawaves [6]Zhongguancun Laboratory

***Abstract*—RDMA is a kernel-bypass and transport-offload technology that provides high throughput and low delay for datacenter networks, and DCQCN is the default and most widely used congestion control algorithm in large-scale RDMA networks. DCQCN involves over 10 parameters at RNICs and switches, and their settings significantly affect network performance, currently relying heavily on exhaustive manual tuning. Although some automatic methods are proposed to tune a subset of DCQCN parameters, none of them comprehensively address all parameters at both RNICs and switches, resulting in compromised network performance. In this paper, we propose PARALEON, an automatic and adaptive system to tune DCQCN parameters comprehensively. We design a millisecond-level sketch-based monitoring mechanism for accurate network-wide measurement, which collects runtime metrics as feedback to guide the tuning process. We also analyze the complicated parameter impacts on network performance, and leverage an improved heuristic searching algorithm for timely performance optimization with better efficiency and convergence. We implement PARALEON and conduct extensive experiments in both NS3 simulations and a real-world testbed. The results show that PARALEON achieves $3.8\% \sim 61.4\%$ higher performance than existing tuning schemes.**

I. INTRODUCTION

Remote Direct Memory Access (RDMA) empowers a network client to directly access the memory of a remote server without involving its CPU. By offloading I/O responsibilities from CPU into RDMA-enabled network interface cards (RNICs), RDMA bypasses the network stack and enables zero-copy networking, achieving a significant enhancement in per-flow throughput and packet-level latency. With the advent of RDMA over Converged Ethernet Version 2 (RoCEv2) [1], the performance benefit of RDMA has been brought to many datacenter applications, such as large language model (LLM) training [2]–[5], cloud storage [6], [7], Remote Procedure Call (RPC) [8]–[10], and various distributed systems [11]–[15].

Generally, to maintain high transport performance, RDMA requires a lossless fabric, so Priority Flow Control (PFC) [16] is introduced to eliminate packet loss. Nonetheless, PFC can cause several practical issues, such as head-of-line blocking, unfairness, PFC storm [17], security vulnerability [18], therefore, congestion control (CC) is integrated to mitigate these issues. Datacenter Quantized Congestion Notification (DCQCN) [19], the default CC algorithm in NVIDIA RNICs, stands as the predominant RoCEv2 CC algorithm embraced by leading industry companies [2], [6], [7], [20]. DCQCN can alleviate the network queue length before PFC is triggered, thereby reducing PFC occurrences.

979-8-3503-5171-2/24/$31.00 ©2024 IEEE

DCQCN involves 10+ parameters at RNICs and switches [21], and different network environments require distinct DCQCN parameter settings to achieve superior network performance. Firstly, the underlying network devices and topology exhibit heterogeneity across different RDMA clusters, necessitating different DCQCN parameters to align with specific network capacity and topology. Secondly, and perhaps more crucially, in RDMA clouds [7], [22], [23], different RDMA clusters are characterized by heterogeneous workloads, demanding distinct DCQCN parameters to satisfy their Service Level Agreements (SLAs). Even within the same cluster, diverse traffic patterns coexist and undergo frequent shifts temporally, and they can be launched spatially at any nodes across network, requiring timely adjustment for DCQCN parameters to adapt to traffic dynamics.

Current DCQCN parameter setting highly relies on manual efforts of experts [24], [25], which not only imposes a significant burden on experts, but also can hardly keep pace with increasing traffic dynamics. To alleviate these problems, researchers have proposed several automatic methods to tune a subset of DCQCN parameters. ACC [25] uses multi-agent reinforcement learning techniques to tune ECN thresholds at switches. DCQCN+ [26] adjusts rate increase steps and timers at RNICs according to runtime incast scale. Nevertheless, these existing methods fail to consider all DCQCN parameters, leading to compromised network performance.

However, developing an automatic and adaptive system to tune DCQCN parameters comprehensively is non-trivial and encounters two key challenges. The first challenge is to maintain timely awareness of the entire dynamic RDMA network, including runtime network metrics and traffic patterns. We need a runtime monitor to perceive network-wide states and frequent changes, so as to guide *when* and *how* to tune these DCQCN parameters. The second challenge is to conduct timely yet effective searches for comprehensive DCQCN parameters. Given the large searching space and complex parameter influences, we need an efficient and effective searching algorithm to rapidly generate performant DCQCN settings to optimize the network performance.

In this paper, we design PARALEON (**PARA**meter Chame**LEON** for short), an automatic and adaptive tuning system for DCQCN parameters, which comprises a *Runtime Metric Monitor* module and a *Performance-oriented Tuning* module. PARALEON is an event-driven and closed-loop tuning system: Runtime Metric Monitor collects the network-wide

TABLE I: Key DCQCN parameters tuned by network experts (others are remained default).

ai_rate	hai_rate	rate_reduce_ monitor_period	min_time_ between_cnps	K_{min}	K_{max}	P_{max}
50Mbps	150Mbps	80μs	96μs	1600KB	6400KB	0.2

metrics periodically to determine whether the tuning process should be triggered; when triggered, Performance-oriented Tuning searches new DCQCN parameters for both RNICs and switches iteratively to adapt to the inputted metrics, until superior performance is achieved.

Runtime Metric Monitor is responsible for monitoring millisecond-level network metrics (e.g., throughput, RTT, PFC) and traffic pattern (e.g., flow size distribution (FSD)). However, measuring millisecond-level flow statistics is non-trivial, as commodity RNICs only provide per-port counters rather than in per-QP (queue pair) granularity, and mature monitoring tools such as NetFlow [27] and sFlow [28] only provide O(seconds) and coarse-grained flow statistics based on packet sampling. To capture accurate flow size distributions within O(milliseconds) monitor intervals, we modify Elastic Sketch [29] with a codesign of control plane and data plane: the control plane 1) periodically collects sketches from the data plane for every O(milliseconds) monitor interval; 2) maintains a sliding window to update ternary states of each flow by historical records. This mechanism alleviates the misidentification of elephant and mice flows, ensuring the accuracy of flow size distribution within O(milliseconds) monitor intervals.

In Performance-oriented Tuning, we design a utility function to express the RDMA network performance, with the objective of maximizing its value by searching for performant DCQCN parameters. We experimentally observe most single-parameter impacts can be categorized into throughput-friendly and delay-friendly directions, which can be used to guide parameter searching for higher throughput or lower delay. However, inter-parameter impacts are also non-negligible, since adjusting more than one parameter in the same direction may result in compromised performance. Therefore, PARALEON resorts to the simulated annealing (SA) algorithm that mutates DCQCN parameters in different randomness dimensions to compensate for potential inter-parameter impacts. For better searching efficiency and convergence, we further optimize SA by: 1) adopting guided randomness for reasonable and focused searches; 2) setting more relaxed temperature for timely parameter adjustment.

We implement a PARALEON prototype, and evaluate PARALEON's tuning performance in both NS3 simulation [30] and a hardware testbed. Simulation results show that, with different workloads, PARALEON achieves $3.8\% \sim 61.4\%$ lower flow completion time (FCT) than static settings (default [21] and expert in Table I) and automatic tuning schemes (ACC [25] and DCQCN+ [26]). PARALEON also acquires higher flow size distribution accuracy than other monitoring schemes (NetFlow [27] and naive Elastic Sketch [29]), and better tuning convergence than naive SA searching. Real-world testbed results show that PARALEON achieves up to 19.5% higher bandwidth than default and expert settings for distributed training workloads, and presents better adaptivity for runtime throughput and latency with coexistence of distributed training and RPC services. PARALEON has also been integrated into the toolboxes of Infrawaves currently.

TABLE II: `NCCL-Tests alltoall` performance with two DCQCN parameter settings.

Out-of-place algbw (GB/s) Size (MB) / Settings	512	1024	2048	4096	8192
Default	6.37	4.73	4.73	6.95	11.56
Expert	25.69	12.59	12.97	39.32	40.93

The main contributions of this work include:

- We identify the necessity of DCQCN parameter tuning and the limitations of existing tuning schemes (§II).
- We design PARALEON to tune DCQCN parameters automatically and adaptively, which monitors O(milliseconds) runtime metrics and searches parameters to optimize network-wide performance (§III).
- We implement PARALEON and conduct experiments to show PARALEON's advantageous performance on the NS3 simulation and the real testbed (§IV).

II. Background & Motivation

PFC & DCQCN for RDMA. Generally, RDMA is very sensitive to packet loss [31], and PFC is deployed in RoCEv2 to prevent packet loss. When the queue length surpasses a predefined PFC threshold, the congested queue signals the upstream port to pause for a specified duration. In practical usage, PFCs can induce severe network-wide performance anomalies [17], [18], such as head-of-line blocking, unfairness, PFC storm [17], and even security vulnerability [18]. Therefore, CC is introduced to reduce PFC triggers, with DCQCN emerging as the *de facto* standard in widespread usage. DCQCN is based on Additive-Increase/Multiplicative-Decrease (AIMD) and involves three parties: 1) Congestion Point (CP): switches mark packets with Explicit Congestion Notification (ECN) when the local queue length exceeds ECN thresholds (lower than the PFC threshold); 2) Notification Point (NP): the receiver RNIC sends congestion notification packets (CNPs) to the sender RNIC when receiving ECN-marked packets; 3) Reaction Point (RP): the sender RNIC cuts sending rates in an MD manner upon receiving CNPs, otherwise keeps increasing rate via a roughly AI mechanism. DCQCN has more than 10 parameters, including RNIC-side parameters categorized into Rate Increase, Rate Decrease, Alpha Update and Notification Point to regulate the AIMD process of each QP, as well as switch-side ECN thresholds $K_{min}, K_{max}, P_{max}$ for ECN marking rate at CPs. More details can be found here [21].

Effects of DCQCN parameters on performance. As DCQCN parameters determine the AIMD process, different parameter settings can result in varying network performance. We conduct a real-world experiment where different DCQCN parameters are applied to a 16-node machine learning training cluster from our partner company, Infrawaves. Each node is equipped with 8 NVIDIA H100 GPUs and 8 NVIDIA ConnectX-7 RNICs. The network follows a 1:1 oversubscribed

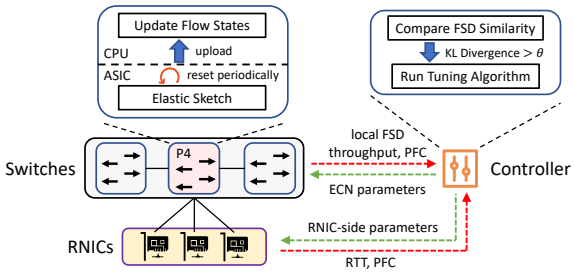


Fig. 1: PARALEON overview.

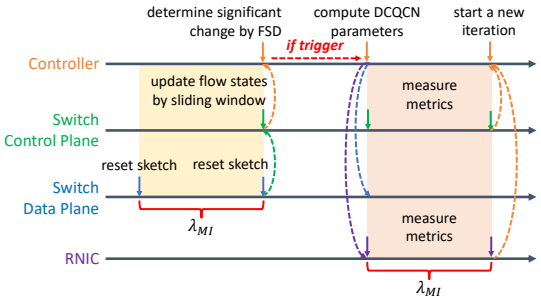


Fig. 2: PARALEON runtime metric monitor.

two-tier CLOS topology where all links are 400Gbps. We leverage `NCCL-Tests` [32], a test suite for the famous inter-GPU collective communication library NCCL, which is widely used by today's mainstream machine learning training frameworks. With a 128×128 `alltoall` collective communication primitive, we compare two DCQCN parameter settings and record their out-of-place algorithm bandwidth performances with increasing transmitted data sizes, as shown in Table II. The first one is default parameter setting provided by NVIDIA [21]. The second setting is manually tuned by network experts based on their experience knowledge and instructions [24] (Table I). Obviously, the expert setting achieves much higher bandwidth with increasing transmitted data size, indicating the expert setting can accelerate the LLM training. The results highlight the substantial effects of DCQCN parameter tuning, emphasizing the parameter tuning necessity for superior network performance.

DCQCN parameter tuning approaches. Nowadays, DCQCN parameter tuning highly relies on expert knowledge and manual experience. In industrial environments, a complete tuning usually consumes weekly manual efforts [24], [25], and operators have to adjust them frequently with constantly changing traffic patterns. To alleviate this issue, researchers have proposed several automatic tuning tools. ACC [25] deploys an agent at each switch control plane to monitor local port rate, ECN marking rate and queue length, which are taken as inputs for the Deep Double D-network algorithm to update local ECN thresholds. DCQCN+ [26] updates the CNP interval proportionally to congested flow amount, and notifies the RPs by appending the latest interval in CNPs, based on which the RPs can adjust rate increase steps and timers. Despite being automatic, these existing tuning tools only consider a subset of DCQCN parameters, resulting in compromised network performance (§IV-B). It is also difficult to combine ACC and DCQCN+, because ACC periodically collects runtime metrics from the switch data plane, while DCQCN+ relies on irregular generation of congestion events from NPs, leading to incompatible monitoring mechanism and tuning actions. Therefore, we need a new automatic and adaptive tuning system, which not only considers comprehensive DCQCN parameters at both RNICs and switches simultaneously, but also accommodates to the dynamic RDMA traffic patterns.

III. PARALEON DESIGN

A. Overview

PARALEON has two modules, including:

- *Runtime Metric Monitor* collects runtime metrics to: 1) determine whether network-wide traffic pattern has significant changes to trigger tuning, and 2) guide the tuning process.
- *Performance-oriented Tuning* tunes DCQCN parameters iteratively to optimize network-wide performance adaptive to runtime metrics.

PARALEON has three components depicted in Figure 1. ① **Programmable Top-of-Rack (ToR) switch**: the data plane runs Elastic Sketch [29] to record flow sizes; the control plane periodically updates flow states based on local sketch, and uploads throughput, PFC and local flow size distribution to the centralized controller. ② **RNIC**: uploads RTT and PFC metrics to the controller. ③ **Centralized controller**: collects metrics from switches and RNICs, triggers tuning by significant traffic pattern change and executes tuning algorithm.

PARALEON leverages Kullback-Leibler (KL) Divergence [33] to compute the similarity of two successive network-wide flow size distributions R_t and R_{t-1} at sub-second. If $KL(R_t, R_{t-1})$ exceeds a predefined threshold θ, it implies network-wide traffic has changed significantly within the cluster. Consequently, the tuning process is triggered, and new DCQCN parameters will be generated iteratively for the new traffic pattern.

B. Runtime Metric Monitor

The monitor includes two parts, as shown in Figure 2. The first part, *flow size distribution measurement* (highlighted in yellow), operates continuously in a layered style. Every monitor interval (i.e., λ_{MI}), each ToR switch control plane retrieves and resets the local sketch recorded in the data plane, and runs a sliding window to update flow states and local flow size distribution. The local flow size distributions from all ToR switches are transmitted to the centralized controller, where they undergo aggregation to update the network-wide flow size distribution. The layered flow size distribution measurement can reduce the data transfer and alleviate the resource burden of the controller. The second part, *runtime metric collection* (highlighted in pink), operates on an event-driven basis: it is initiated only if parameter tuning is triggered by a significant flow size distribution change. At the beginning of a monitor interval, the controller generates new DCQCN parameters, and assigns them to each switch and RNIC. During the monitor interval, the switches and RNICs monitor runtime throughput, RTT and PFC, then upload them to the controller at the end

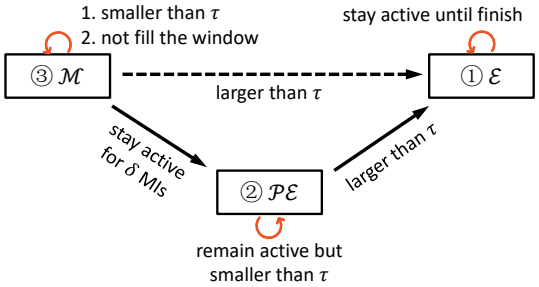


Fig. 3: Transition graph of flow ternary states.

of the monitor interval to start a new tuning iteration. *Runtime metric collection* stops until a complete tuning process finishes.

For sketch implementation, we use Elastic Sketch [29] in the data plane as measurement points to capture per-flow size statistics. Elastic Sketch comprises a Heavy Part to record top-k elephant flows and a Light Part for mice flows, along with "Ostracism" voting mechanism to maintain elephant flows in the Heavy Part. Each bucket stores a flowID f, a flag as well as $vote^+$ and $vote^-$ for positive and negative votes. Upon a packet arrival, Elastic Sketch hashes it to a designated bucket and checks for collision. If the original f residing in the bucket collides too many times (i.e., $\frac{vote^-}{vote^+}$ is larger than a threshold), it is kicked out to the Light Part, and the bucket is replaced with a new elephant flowID f'. Elastic Sketch maintains a provable tradeoff between SRAM utilization and sketch accuracy to capture per-QP size statistics in high-speed RDMA networks.

However, relying solely on Elastic Sketch presents two challenges for accurate flow size distributions. First, as a packet traverses through multiple measurement points along the forwarding paths, a flow may be recorded in multiple switches. This overlap of local flow size distributions, computed by each switch control plane, can result in inaccuracies when aggregating them into the network-wide flow size distribution. Second, to capture frequent O(milliseconds) workload changes and achieve timely tuning in highly dynamic RDMA environment, an millisecond-level monitor interval is necessary, but naive Elastic Sketch cannot identify elephant and mice flows correctly within such small monitor intervals. For example, during heavy congestion, an elephant flow may converge to a low throughput and transmit less than τ data size (say τ = 1MB [34]) within λ_{MI}, and naive Elastic Sketch will misidentify it as a mice flow. Moreover, if an elephant flow arrives very shortly before the sketch resets, its measured data size may not exceed τ, leading to incorrect identification as well. The root cause is, naive Elastic Sketch differentiates a flow based on only one single monitor interval, such that any state transition opportunities after several small monitor intervals are completely ignored. Therefore, supplements to Elastic Sketch are necessary for flow size distribution accuracy.

Keypoint 1: insert each packet to only one sketch. One possible solution is to provide a sketch manifest that assigns sketching actions for each measurement point [35]. This solution works with sophisticated sketch manifests, but requires frequent updates by centralized controller to catch up with dynamic traffics, which is neither flexible nor timely enough. Instead, PARALEON adopts a decentralized marking solution. When a packet arrives, each measurement point checks an unused Type of Service (TOS) bit before insertion. If marked, it implies the packet has been inserted into a previous sketch along the forwarding path, and the switch just forwards the packet. If not marked, it means the packet has not been inserted into any previous sketch, so the switch inserts this packet to the local sketch, marks the TOS, and then forwards it. This marking solution eliminates the sketch overlap, allowing the controller to simply aggregate local sketch results into an accurate network-wide flow size distribution. Note that RoCEv2 protocol does not use TOS to identify packets' Quality of Service (QoS) by default, therefore our marking mechanism will not influence the normal QoS functions.

Keypoint 2: update ternary flow states by sliding window. To capture flow state transition within O(milliseconds) monitor intervals, PARALEON 1) introduces a third state to describe flows that are "temporary" mice but likely to evolve into elephant in future; 2) uses a sliding window to update flow states with historical monitor intervals. Every λ_{MI} after reading sketches from the data plane, the switch control plane records the transmitted data size of each *flowID* in this monitor interval, and uses history data in previous monitor intervals to update each flow's state by the sliding window.

PARALEON introduces ternary states for each flow f: elephant ($\mathcal{E}$), potential elephant ($\mathcal{PE}$) and mice ($\mathcal{M}$), with the state transition graph shown in Figure 3. ①: f is $\mathcal{E}$ if its aggregated transmitted data $\Phi(f) \geq \tau$. ②: f is $\mathcal{PE}$ if $\Phi(f) < \tau$, but remains active within the window, i.e., f's transmitted data stays positive for not less than δ monitor intervals (δ is the window size). ③: f is $\mathcal{M}$ if $\Phi(f) < \tau$ and cannot fill the window, i.e., being active for less than δ monitor intervals. A $\mathcal{PE}$ flow can stay in $\mathcal{PE}$ if ② is satisfied, or transition to $\mathcal{E}$ if $\Phi(f)$ exceeds τ at a certain monitor interval.

Using the ternary state transition, PARALEON uses a sliding window to update f's ternary states with history monitor intervals. An illustrative example is given in Figure 4 with $\delta = 3$ and τ = 1MB. For f_1, its data size exceed τ, so f_1 is $\mathcal{E}$. For f_2, at the first two monitor intervals, its transmitted data does not exceed τ, so PARALEON regards f_2 as $\mathcal{M}$ at MI_1 and MI_2. At MI_3, f_2 fills the window and keeps active for δ monitor intervals, so f_2 transitions to $\mathcal{PE}$. From MI_3 to MI_6, f_2 stays in $\mathcal{PE}$ state with ② satisfied. Finally at MI_7, $\Phi(f_2)$ exceeds τ, so f_2 transitions to $\mathcal{E}$. For f_3, it keeps active and fills the window at MI_3, so f_3 transitions from $\mathcal{M}$ to $\mathcal{PE}$ at MI_3. However, no transmitted data is discovered at MI_8, which means f_3 is not active (maybe finish), so f_3 will not transition to $\mathcal{E}$. For each $\mathcal{PE}$ flow, PARALEON approximates its contribution to flow size distribution, which is intuitively proportional to its likelihood of transitioning to $\mathcal{E}$ in future, and the likelihood is refined as more monitor intervals elapse.

C. Performance-oriented Tuning

In this section, we first explain why we choose a performance-oriented approach, and define a utility function to characterize the network-wide performance. Then, we pro-

Fig. 4: Update flow states by sliding window.

vide several observations regarding the impact of DCQCN parameters on runtime metrics. Finally, based on the above observations, we design a heuristic tuning algorithm to search DCQCN parameters to maximize the utility function.

From reactive to performance-oriented. Conventional wisdom leverages a reactive approach to adjust DCQCN parameters based on observed network events and expert knowledge. For example, in DCQCN+ [26], the sender RNICs automatically tune the rate increase steps and timers reactive by observed incast events for better throughput. However, in complicated scenarios, a network event can be caused by multiple factors, including but not limited to CNP intervals, steps/timers at RNICs, and ECN thresholds at switches. It is difficult to summarize a comprehensive and accurate relationship from observed events to adjustment actions. Therefore, PARALEON resorts to a performance-oriented approach [36]–[38]: it takes runtime metrics as inputs, conducts multiple parameter attempts, and outputs the most performant parameter setting that maximizes the utility function. This approach offers a more direct and flexible path from observed metrics to tuning actions, avoiding predefined tuning principles and rigid actions from events to specific parameter values.

Utility function. The utility function is defined based on network-wide runtime metrics, as shown in Equation (1):

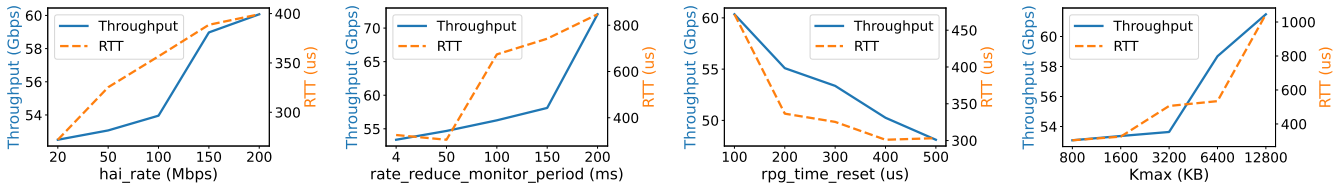
$$U = \omega_{TP} \cdot \mathcal{O}_{TP} + \omega_{RTT} \cdot \mathcal{O}_{RTT} + \omega_{PFC} \cdot \mathcal{O}_{PFC} \quad (1)$$

$\mathcal{O}_{TP} = \overline{\delta_l}$ is the average bandwidth utilization of each active uplink $l \in L$ connecting RNICs and ToR switches. $\mathcal{O}_{RTT} = \overline{\gamma(i,j)}$ is the average normalized RTT of each RNIC pair $i, j \in C$, and each $\gamma(i,j)$ is computed by $\frac{Base\ \ path\ \ delay}{Runtime\ \ RTT_{i,j}}$. Inspired by Swift [39], *Base path delay* equals $n_{i,j} \cdot d_{i,j}$, where $n_{i,j}$ and $d_{i,j}$ denote the hop counts (starting TTL minus received TTL) and minimum link propagation delay between RNIC i and j, respectively. $\mathcal{O}_{PFC} = 1 - \frac{\overline{\lambda_{xoff}}}{\lambda_{MI}}$ where $\overline{\lambda_{xoff}}$ is the average PFC pause duration of each device within a monitor interval λ_{MI}. ω_{TP}, ω_{RTT} and ω_{PFC} are performance weights assigned by operators and $\omega_{TP}+\omega_{RTT}+\omega_{PFC} = 1$. Operators can customize the weights according to specific scenarios and preferences. For example, for throughput-sensitive workload such as LLM training, we can empirically let $\omega_{TP} = 0.5$, $\omega_{RTT} = 0.2$ and $\omega_{PFC} = 0.3$.

The reasons why we choose these three factors are given as follows. On one hand, throughput and RTT are the most direct metrics indicating the network performance. On the other hand, in RoCEv2 networks, PFC usually implies heavy congestion where queue length is growing very rapidly. Although RTT can reflect queue buildup, it falls short of identifying fine-grained congestion levels. For example, a high RTT may stem from two possible scenarios: 1) the probing packet passes multiple congestion points with lengthy but still tolerable queue length; 2) the probing packet is paused in the upstream device of the incast switch triggering PFCs. Obviously, the second scenario is severe and risky [17], [18], but RTT variance alone cannot differentiate them, necessitating $\mathcal{O}_{PFC}$ to identify the second scenario. Furthermore, $\mathcal{O}_{PFC}$ can capture more transient PFC occurrence, whereas throughput and RTT are average metrics failing to reflect transient anomaly.

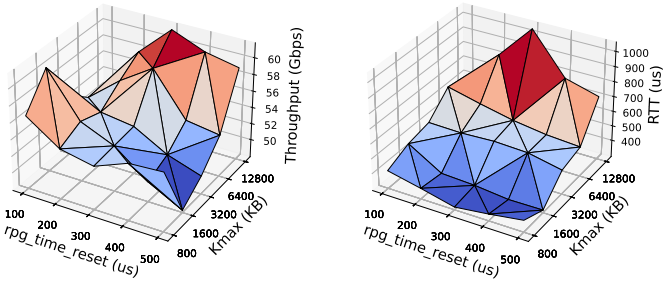
Observations on parameter impacts. Before delving into the tuning algorithm, we empirically investigate how an individual DCQCN parameter impacts the network performance, which is called *single-parameter impact*. In NS3, we launch a 20×20 `alltoall` collective communication in a two-tier clos topology with 12 switches, and monitor the average throughput and RTT for several monitor intervals. We change the values of 4 representative parameters, i.e., *hai_rate*, *rate_reduce_monitor_period*, *rpg_time_reset* and K_{max}, and the other parameter values remain default as referred to [21]. Figure 5 demonstrates the single-parameter impacts, revealing significant effect on network throughput and RTT. In general, the tuning directions of each individual parameter can be broadly divided into *throughput-friendly* and *delay-friendly* directions, with each being more beneficial to throughput and RTT, respectively. Taking *hai_rate* as an example, incrementing its value is a throughput-friendly direction: increasing *hai_rate* allows each QP to take a more aggressive increase step for a rate hyper-increase event, which is helpful to elevate the network throughput. Conversely, decrementing is a delay-friendly direction for *hai_rate*: as slower packets are injected into the network, packet queuing is mitigating at switches, resulting in decreasing RTT. Similar analysis can apply to *rpg_time_reset*, *rate_reduce_monitor_period* and K_{max}.

Building upon the above observations, a straightforward tuning scheme seems obvious: when network-wide traffic pattern is dominated by elephant/mice flows that require higher throughput/lower delay, we steer all DCQCN parameters in the throughput/delay-friendly direction accordingly. Nevertheless, this seemingly intuitive tuning approach overlooks potential inter-parameter impacts, i.e., driving more than two parameters in the same direction may lead to a compromised or even contradictory overall performance. To illustrate this, we investigate the inter-parameter throughput impacts of *rpg_time_reset* and K_{max}, as shown in Figure 6(a). Obviously, when we drive *rpg_time_reset* and K_{max} in the same throughput-friendly direction (i.e., decrement *rpg_time_reset* and increment K_{max}), network throughput does not exhibit a monotonic increasing trend, displaying several convex and concave points. The reason is, when these two parameters are too throughput-friendly simultaneously (i.e., the values of *rpg_time_reset* and K_{max} are too small and large, respectively), packet injection is too aggressive and exceeds the equilibrium point. This aggravates queuing buildup and triggers more CNPs and even PFCs to slow down the transmission rate instead, resulting in unexpected congestion and throughput decline. Similar inter-parameter impacts are observable for RTT in Figure 6(b). As a result, when adjusting DCQCN parameters, inter-parameter impacts should be carefully taken into consideration.



(a) *hai_rate* (b) *rate_reduce_monitor_period* (c) *rpg_time_reset* (d) K_{max}

Fig. 5: Single-parameter impacts on network performance.



(a) Throughput (b) RTT

Fig. 6: Inter-parameter impacts on network performance.

Improved SA algorithm. The inter-parameter impacts are non-negligible, but it is difficult to summarize them given 10+ DCQCN parameters and complicated network states. The optimal algorithm is to explore comprehensive inter-parameter impacts by traversing all possible DCQCN parameter combinations, but it fails to output timely results. Therefore, we resort to a simulated annealing (SA) searching algorithm that introduces randomness to the parameter searching to compensate for the inter-parameter impacts. The rationale is, with high temperature at the beginning, SA can explore and mutate new attempts in more random directions and steps, which is helpful to jump out of the local optima. As the temperature cools down, the overall system converges toward a lower energy state and ultimately approaches the optimal solution. Given the simple but very effective searching benefits, SA can explore more parameter combinations in various randomness dimensions, and output suboptimal results within polynomial time complexity, making it a viable method to compensate for potential inter-parameter impacts.

Nonetheless, naive SA still cannot satisfy PARALEON's timeliness requirement, which takes too many rounds to converge to a high utility function value. The fundamental issue lies in the indiscriminate nature of naive SA, which exhausts too many unnecessary attempts within the entire search space, without exploiting runtime traffic patterns and empirical parameter impact observations. As timely adjustment takes precedence over optimal tuning in a practical RDMA environment, we can "instruct" and prune the SA searching in more focused and sensible direction for fewer mutations and faster convergence.

PARALEON introduces two optimizations to improve PARALEON's SA searching efficiency. Optimization 1 is the guided randomness. On one hand, by exploiting the empirical parameter impacts and runtime traffic patterns, PARALEON drives each parameter to cater to the dominant flow type with higher probability, and maintain some level of random exploration in the anti-dominant direction. For example, if network-wide flow size distribution is composed of 80% elephant flows and 20% mice flows, for each parameter $p \in P$, PARALEON can drive p in throughput-friendly direction with 80% probability, or steer p in delay-friendly direction with 20% probability. On the other hand, we provide an empirical step s_p for each p based on impact observations, and multiply s_p with a random number in each iteration, making sure the tuning steps can be random but within a bounded range. This optimization strikes a balance between exploitation and exploration [40]: PARALEON exploits empirical observations & runtime traffic pattern for faster convergence, and maintains native random exploration of SA. Optimization 2 is to use more relaxed temperature settings for SA. Since timely adjustment precedes optimal tuning, we set a more relaxed temperature to jump out of loops more rapidly, prioritizing responsiveness to dynamic traffic changes.

Tuning process of SA. Algorithm 1 is the SA tuning process of PARALEON. For every λ_{MI}, the centralized controller collects throughput, RTT and PFC metrics from switches and RNICs, and updates the utility function value. Lines 14-22 mutate new DCQCN parameters by Optimization 1. According to collected flow size distribution from switches, PARALEON computes the network-wide dominant flow type and its proportion μ. For each DCQCN parameter $p \in P$, SA drives p in the dominant direction with a probability of $\min(\mu, \eta)$, or steers p in the anti-dominant direction with a $(1 - \min(\mu, \eta))$ probability, along with a random step $s'_p = s_p \times rand(0.5, 1)$. η is an upper bound of exploitation rate, making sure SA has at least $(1 - \eta)$ probability to explore the anti-dominant direction even with imbalanced flow size distributions. New comprehensive DCQCN parameter setting P_m are dispatched to RNICs and switches after a tuning round, and controller will wait for new metric feedback in the next monitor interval. The tuning process will finish until the temperature is lower than a predefined value.

IV. IMPLEMENTATION AND EVALUATION

In this section, we evaluate PARALEON in both large-scale NS3 simulations [41] and a real-world testbed with respect to the following key questions: (1) How effective is PARALEON in various workload types (§IV-B1)? (2) How adaptive is PARALEON to adjust DCQCN parameters with workload dynamic change (§IV-B2)? (3) How necessary are PARALEON's designs for accurate monitoring and efficient tuning (§IV-B3, §IV-B4)? (4) How about PARALEON's performance in the real-world testbed (§IV-C)?

Algorithm 1: SA Tuning Algorithm

Initialization for SA parameters;
while *curr_temp* $>$ *final temp* **do**
  for $i = 0; i < total\_iter\_num; i++$ **do**
    Controller waits λ_{MI} to obtain $\overline{TP}$, $\overline{RTT}$, $\overline{PFC}$;
    Update new_util by Equation (1);
    if $new\_util > current\_util$ *or* $e^{\frac{new\_util - current\_util}{curr\_temp}} > rand(0,1)$ **then**
      $current\_util = new\_util$;
      $current\_solution = P_m$;
    end
    if $current\_util > best\_util$ **then**
      $P = current\_solution$;
      $best\_util = current\_util$;
    end
    Compute dominant flow proportion μ;
    for *each* p *in* P **do**
      Update step $s'_p = s_p \times rand(0.5, 1)$;
      if $rand(0,1) < \min(\mu, \eta)$ **then**
        Drive p in dominant direction with s'_p;
      else
        Drive p in anti-dominant direction with s'_p;
      end
    end
    Dispatch P_m to RNICs and switches;
  end
  curr_temp = cooling_rate $\times$ curr_temp;
end

TABLE III: PARALEON system settings in NS3.

Category	Parameter	Value
Ternary flow state update	elephant flow size threshold τ	1MB
	window size δ	3
Tuning trigger threshold & weights	KL divergence value θ	0.01
	$\omega_{TP}, \omega_{RTT}, \omega_{PFC}$	0.2, 0.5, 0.3
SA algorithm	total_iter_num	20
	cooling rate	0.85
	initial temperature	90
	final temperature	10
Miscellaneous	monitor interval λ_{MI}	1ms
	maximum SA exploitation rate η	0.8

A. Implementation

The NS3 simulation codes are publicly available in [30], and the PARALEON prototype on the real-world tested is implemented with 1500 incremental lines of C and Python codes based on the system of our partner company, Infrawaves. Each switch control plane and server deploy agents responsible for: 1) reading runtime counters & metrics by device interfaces; 2) uploading metrics to the controller; 3) receiving DCQCN parameters from the controller, and updating them to switch data plane and RNICs. Data transfer between the controller and switches & servers is via gRPC [42] with TCP protocol to guarantee reliable transport. To prevent interference with RDMA traffic, TCP and RDMA traffic are assigned to different queues. As for O(milliseconds) flow size distribution measurement, we deploy Elastic Sketch [43] in the data plane. Meanwhile, the agent in the switch control plane periodically reads and resets registers of Heavy Part in the data plane, then updates the ternary flow states by the sliding window.

B. NS3 Large-scale Simulation

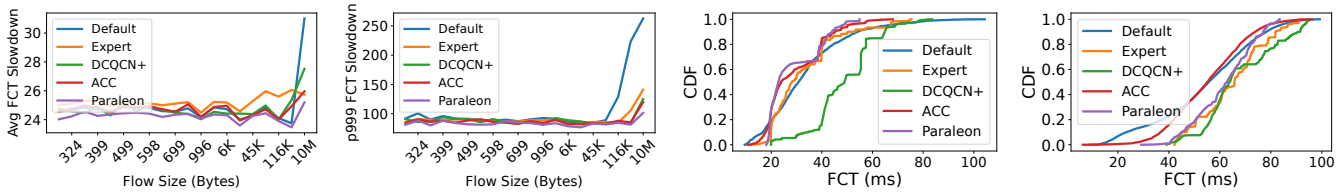
Network topologies. We use a two-tier 4:1 oversubscribed CLOS topology of 8 ToR switches, 4 leaf switches and 128 servers with all 100Gbps links and 5μs propagation delay. The switch buffer is 12MB.

Workloads. We evaluate two kinds of workloads. The first workload is a 128-node `FB_Hadoop` [44] where most flows are mice but most traffic is contributed by elephant flows, and we set the default load as 30%. The second workload is an ON-OFF LLM training workload [45] with 20 workers: during the ON period, the nodes send the same 12MB flow size to each other by `alltoall`; when finishing data transmission, each node spends 20ms for model update during the OFF period. We use `alltoall` due to its network-intensive property that induces severer incasts than ring- and tree-based traffic patterns generated by other collectives (e.g., `allreduce`).

Baselines. We first compare PARALEON with four different DCQCN parameter settings in terms of FCT and runtime throughput & RTT: two static parameter settings provided by NVIDIA [21] and by experts (Table I), as well as two automatic tuning schemes DCQCN+ [26] and ACC [25]. We also evaluate the necessity of dynamic tuning of PARALEON in terms of runtime throughput & RTT. We provide two static DCQCN parameter settings, which are offline pretrained by PARALEON, i.e., *Pretrained 1* for `alltoall` LLM training and *Pretrained 2* for `FB_Hadoop` workload.

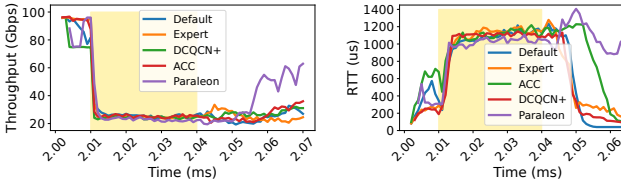
Design choices comparison. To prove the necessity and superiority of PARALEON's designs, we compare PARALEON with naive Elastic Sketch, and a mature monitoring tool, *NetFlow*, in commodity switches, in terms of flow size distribution accuracy and workload FCT. The packet sampling rate and monitor interval of NetFlow are 1:100 and 1s, respectively. We also conduct ablation studies on optimizations of guided randomness and relaxed temperature for SA tuning algorithm. The default settings are listed in Table III.

1) Overall performance on various workloads: For `FB_Hadoop` workload, Figure 7(a) and (b) show the average and 99.9-percentile FCT slowdown of different flow sizes in 30% loads. It is evident that PARALEON exhibits superior FCT performance compared to other tuning schemes with different flow sizes. For example, when flow size is smaller than 120KB, PARALEON's average FCT is at least 3.8% smaller than other tuning schemes. Notably, for elephant flows exceeding 1MB, PARALEON advantage becomes more obvious, surpassing others by a maximum of 61.4%. This is because, when mice flows dominate the traffic at the beginning, PARALEON can adjust DCQCN parameters to minimize the RTT and expedite the completion of mice flows. As most mice flows finish, remaining elephant flows take dominance, thus PARALEON tunes DCQCN parameters to maximize the throughput, facilitating faster completion of elephant flows. For LLM training workload, Figure 7(c)



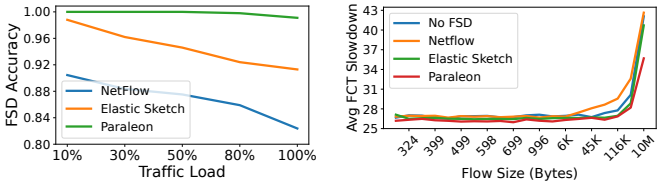
(a) Average FCT of FB_Hadoop (b) P999 FCT of FB_Hadoop (c) 20×20 alltoall (d) 40×40 alltoall

Fig. 7: FCT performance for FB_Hadoop and ON-OFF training.



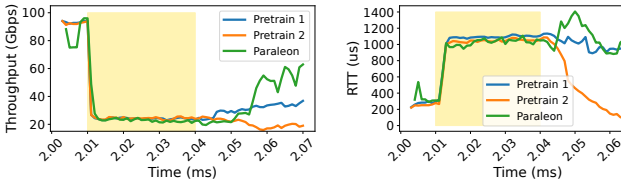
(a) Runtime throughput (b) Runtime RTT

Fig. 8: Different tuning schemes with FB_Hadoop "influx".



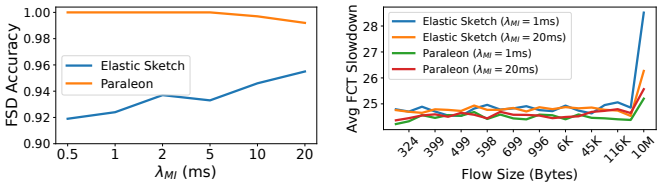
(a) FSD accuracy in traffic loads (b) 80% load of FB_Hadoop

Fig. 10: Monitoring design comparison.



(a) Runtime throughput (b) Runtime RTT

Fig. 9: Pretrained vs PARALEON with FB_Hadoop "influx".



(a) Value of λ_{MI} (b) FCT of FB_Hadoop

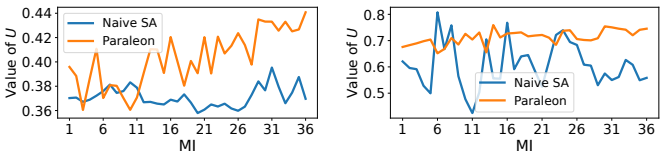
Fig. 11: Effect of monitor interval.

and (d) exhibit the CDF of FCT of different numbers of involved nodes. For LLM training workload, PARALEON can tailor parameters to optimize the throughput, empowering PARALEON to achieve up to a 54.5% reduction for tail FCTs with different alltoall scales. Since the completion of each training round depends on the straggler worker, the tail FCT improvement of PARALEON can greatly accelerate the training task. Meanwhile, we note that this ON-OFF pattern of the LLM training workload does not affect PARALEON's monitoring and tuning process, because it exhibits a similar traffic pattern over tens of milliseconds, preventing frequent fluctuation of network-wide flow size distribution.

2) Traffic dynamics with workload "influx": We investigate a workload "influx" scenario: an LLM training workload is at its ON period as background traffic, and a 30ms FB_Hadoop workload arrives and competes for network resource with the training workload. The runtime throughput and RTT are illustrated in Figure 8 highlighted in yellow. On one hand, when FB_Hadoop arrives at 2.01s, the dominant flow type transitions from elephant flows to mice flows. PARALEON is aware of the transition by flow size distribution variances, so it adjusts the DCQCN parameters to minimize RTT swiftly, thus exhibiting lower RTT during the influx phase than other tuning schemes. As most mice flows conclude, the dominant flow type transitions to elephant flows again, consisting of training workload and remaining FB_Hadoop elephant flows. Consequently, PARALEON tunes DCQCN parameters for higher throughput than other tuning schemes after 2.05s.

The network performance of pretrained schemes and PARALEON are demonstrated in Figure 9. Obviously, PARALEON outperforms two pretrained settings, which achieves lower RTT during influx period and higher throughput for remaining elephant flows. This is because, a pretrained parameter setting can only capture static characteristics and optimize performance for a known and trained workload. For the unknown and complicated traffic pattern, e.g., workload influx in this scenario, the two pretrained schemes fail to adapt to the dynamic workload changes. Conversely, PARALEON can capture traffic variance and mutate different parameter settings according to runtime metrics, therefore obtaining better runtime performance.

3) Monitoring design comparison: We compare PARALEON's monitoring design with other three schemes in terms of flow size distribution accuracy and FCT performance of FB_Hadoop. The first one is *No_FSD*, where flow size distribution is unavailable so SA algorithm are agnostic to runtime flow size distribution. The second one is *NetFlow*, which is available in common switches based on packet sampling and second-level monitor interval. The third scheme uses naive Elastic Sketch to measure runtime flow size distribution, without involvement of switch control plane. The accuracy of flow size distribution and FCT performance of four schemes are given in Figure 10. In Figure 10(a), with different traffic loads, PARALEON achieves more accurate flow size distribution than other monitoring schemes. For one thing, NetFlow is based on packet sampling, which neglects lots of flow statistics and information, and Elastic Sketch misses the intermediate flow status within each single O(milliseconds)



(a) FB_Hadoop (b) alltoall training

Fig. 12: Ablation study on SA optimizations.

monitor interval. For another, PARALEON can identify the $\mathcal{PE}$ flow state and use the sliding window to update the ternary states, thus achieving higher accuracy. As a result, PARALEON achieves the best FCT performance among these scheme, due to the most accurate flow size distribution that guides the SA mutation, as Figure 10(b) shows.

We also evaluate the effect of monitor interval on flow size distribution and FB_Hadoop FCT performance. We compare with Elastic Sketch and PARALEON in different millisecond-level monitor intervals. We ignore NetFlow in this comparison because it is based on O(seconds) monitor interval. In Figure 11(a), the accuracy of flow size distribution of PARALEON equals or is very close to 100% correctness in diverse λ_{MI}. Longer λ_{MI} is helpful for Elastic Sketch to identify an elephant, but its accuracy is still lower than PARALEON in O(milliseconds) λ_{MI}, therefore, PARALEON achieves better FCT performance in Figure 11(b). Meanwhile, we also notice that smaller monitor interval is beneficial for PARALEON, because smaller monitor interval can capture more timely traffic characteristics to guide the SA tuning. Especially when remaining elephant flows dominate the traffic pattern, a smaller monitor interval helps PARALEON drive DCQCN parameters in throughput-friendly direction more swiftly.

4) Ablation study on SA optimizations: We conduct ablation study on SA optimization, i.e., guided randomness and relaxed temperature. We compare PARALEON with *naive_SA*, which follows the original random mutation process for FB_Hadoop and training workloads. The utility function values are given in Figure 12, revealing that PARALEON achieves superior convergence compared to *naive_SA*. This is because naive mutation prevents *naive_SA* from finding high-quality DCQCN parameters within dozens of monitor intervals, which requires much more iterations to converge to a high utility function value. Instead, with guided randomness and relaxed temperature, PARALEON can mutate towards higher-quality DCQCN parameters more quickly and thus provide more responsive tuning outputs.

C. Real Testbed Evaluation

Network topology. We use a two-tier CLOS topology with 8 ToR switches, 4 leaf switches and 32 H100 servers. Each server is equipped with 8 NVIDIA H100 GPUs and 8 NVIDIA ConnectX-7 RNICs. The ToR switches are Intel Tofino2 switches. Each link is 400Gbps with 1:1 over-subscription ratio. We set λ_{MI} = 30ms in testbed experiments.

Workloads. For elephant flows, we use NCCL-Test [32] to generate an alltoall collective communication with

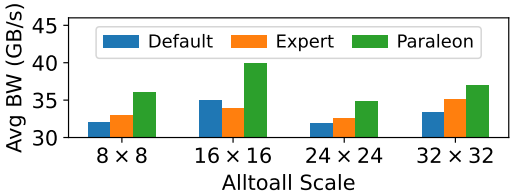
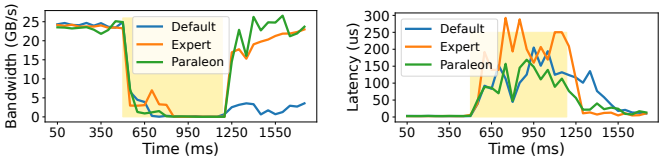


Fig. 13: alltoall bandwidth.



(a) Runtime bandwidth (b) Runtime latency

Fig. 14: Runtime metrics.

different numbers of workers. The default alltoall workload is 32 × 32 with 4GB message size and 1 QP. As for mice flows, we generate a SolarRPC workload [46], which is all comprised of mice flows smaller than 128KB. The controller informs each server's agent to launch RDMA *WRITE* operations with message size following SolarRPC's distribution. The *WRITE* arrival follows a Poisson distribution.

Baselines. We compare PARALEON with the default [21] and expert (Table I) DCQCN settings, and evaluate the bandwidth and latency performance, as well as the system overheads in terms of compute and communication.

1) Bandwidth and latency: The average bandwidth of alltoall workload with various nodes is presented in Figure 13. It is obvious PARALEON exhibits superiority and adaptability to different alltoall scales, surpassing default and expert settings by up to 19.5%. It also proves PARALEON can search for performant DCQCN parameters to accommodate to diverse training scales, which improves the training efficiency in real-world LLM training clusters.

Figure 14 shows the runtime bandwidth and latency. An alltoall workload is running as background traffic, and a SolarRPC workload arrives at a specific time and lasts for a duration highlighted in yellow. The results validate PARALEON's adaptivity to dynamic traffic changes. When SolarRPC arrives, flow size distribution undergoes significant changes due to the overwhelming mice flows. As a result, PARALEON quickly drives DCQCN parameters to minimize the network latency to serve the dominant mice flows. When SolarRPC completes, flow size distribution changes again where elephant flows re-dominate the traffic. In response, PARALEON rapidly recovers the network throughput for the remaining alltoall elephant flows. Compared with default and expert settings, PARALEON can automatically tune parameters adaptive to runtime flow size distribution.

2) System overheads: We evaluate the system overheads of PARALEON as listed in Table IV. On one hand, compute overhead is represented by the CPU utilization of centralized controller for KL divergence computing and SA tuning, and switch control plane in relation to ternary flow state updating. We can see both components have low CPU utilization, imposing a low level of compute burden for controller and

TABLE IV: PARALEON system overheads.

Category	Overhead	Value
CPU utilization	Switch control plane	20.3%
	Centralized controller	3.2%
Memory consumption	Switch control plane	9.5MB
Data transfer	Switches to controller	520B
	RNICs to controller	12B
	Controller to switches & RNICs	76B

switch control plane. For memory consumption, the flow state updating has limited memory costs in the switch control plane, while the sketch occupation in the switch data plane can refer to Elastic Sketch [29]. PARALEON introduces data transfer including runtime metrics and DCQCN parameters, but the data size is quite small, which only takes negligible communication overhead within each monitor interval.

V. DISCUSSIONS

Relaxation of programmable switches. To obtain accurate O(milliseconds) flow size distribution, PARALEON employs sketches in programmable switches to monitor precise per-flow metrics for DCQCN parameter tuning. It is worth noting that PARALEON's "monitoring-tuning" philosophy can still work even when programmable switches are unavailable. For example, for more predictable workloads such as LLM training [47], operators can resort to coarser-grained flow size monitoring and measuring mechanisms by commodity switches, e.g., NetFlow or sFlow in O(seconds)-level granularity. Besides, if future RNICs can provide per-QP counters and statistics, the monitoring and measurement tasks can be conducted completely in RNICs, making programmable switches unnecessary as well. The relaxation of programmable switches can be beneficial to better deployability in industrial environments.

PFC parameters. Although RoCEv2 lossless transport is a coupled traffic control system of PFC and DCQCN, PFC parameters are relatively stable in a given network environment. Generally, operators need to tune only one PFC parameter, i.e., α to limit maximum occupation of shared buffer by each queue, to achieve appropriate PFC triggers. Consequently, when network topology and device&link capacity are determined, the value of α can be computed in advance to ensure lossless transmission, which leaves α a stable setting irrelevant to DCQCN parameters. Usually, α is set as 1/8 to achieve appropriate PFC triggers based on empirical studies [7], [19].

Traffic stability. Despite being adaptive to runtime metrics, PARALEON's tuning action is based on past network conditions. This necessitates an assumption of a stable network-wide traffic pattern during SA tuning, e.g., typically tens or hundreds of milliseconds. In production environment, this assumption makes sense, as most RDMA workloads usually last for a long time, and exhibit consistent traffic characteristics [25]. For example, for LLM training workloads, although periodic, each of them presents a similar traffic pattern over hundreds of milliseconds. This ensures relative traffic stability during the tuning process. Nonetheless, there may exist a few extreme situations where O(microsecond) changes of traffic pattern (e.g., microbusts) happen [48], resulting in inevitable tuning leg. We leave the exploration of more timely and intelligent tuning algorithms to such situations as future work.

PARALEON for large-scale environment. For simplicity, PARALEON adopts a centralized tuning approach, where the controller outputs homogeneous DCQCN parameter settings for all distributed devices. We note that in an extreme large-scale RDMA cloud, due to the heterogeneous traffic pattern and network devices, it is difficult to maintain the homogeneous DCQCN setting for all devices; meanwhile, using only one centralized controller can pose great maintenance challenges as well. As a solution, the operators can divide the cloud into several clusters, each of which is managed by an individual controller and applied heterogeneous DCQCN parameters tailored to its performance preferences.

VI. RELATED WORKS

RDMA CC. Besides DCQCN, there are several CC variants designed for RDMA. TIMELY [49] and Swift [39] argue the ECN signal of DCQCN is coarse-grained, and propose RTT as the congestion signal to adjust AIMD by RTT values or gradients. HPCC [24] introduces finer-grained congestion signals through switch in-band telemetry (INT), allowing RNICs to access more detailed in-network metrics for CC actions. On one hand, in practical usage, many of them encounter similar parameter tuning challenges, and PARALEON's philosophy can apply to them as well. On the other hand, in RDMA networks, DCQCN is still the predominant CC algorithm with wide deployment, attributed to its hardware-friendly implementation and less dependence on advanced network features like INT. This is one of the reasons we target DCQCN in this paper.

Sketch for network measurement. Sketch is an efficient data structure for a variety of network measurement tasks, such as heavy hitter, flow size distribution, cardinality, etc. The emergence of programmable switches enables sketches to be deployed in the data plane, facilitating packet recording at line rate without missing packet information. Numerous efforts have been made to optimize sketches across different dimensions, focusing on accuracy & resource utilization [29], [29], [50], [51] and generality [35], [52]–[54]. We believe sketch-based measuring solutions can play a more crucial role to improve the visibility of high-speed RDMA networks.

VII. CONCLUSION

This paper shows the necessity of DCQCN parameter tuning, and proposes PARALEON to tune DCQCN parameters automatically and adaptively. PARALEON uses a sketch-based design to monitor runtime metrics and flow size distribution in millisecond-level monitor intervals, and introduces ternary flow states and the sliding window to ensure flow size distribution accuracy. With runtime feedback, PARALEON employs an optimized SA searching algorithm with guided randomness and relaxed temperature, which considers the complicated parameter impacts and maximizes the utility function with faster convergence. We implement PARALEON and conduct experiments to show the advantageous performance compared with other DCQCN tuning schemes. We hope PARALEON can serve as the foundation for next-generation "zero-configuration" RDMA networks.

ACKNOWLEDGMENT

We thank our shepherd, Soudeh Ghorbani, and the anonymous ICNP reviewers for their valuable comments. This work is supported in part by the National Natural Science Foundation of China (No. 62221003, 62402025, 62202260, 623B2062), and the Fundamental Research Funds for the Central Universities. Menghao Zhang, Mingwei Xu and Ziteng Chen are the corresponding authors.

REFERENCES

[1] InfiniBand Trade Association, "InfiniBand Architecture Specification Release 1.2.1 Annex A17: RoCEv2," https://cw.infinibandta.org/document/dl/7781, 2014.

[2] 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 *ISCA*, 2022.

[3] K. Qian, Y. Xi, J. Cao, J. Gao, Y. Xu, Y. Guan, B. Fu, X. Shi, F. Zhu, R. Miao, C. Wang, P. Wang, P. Zhang, X. Zeng, Z. Yao, E. Zhai, and D. Cai, "Alibaba hpn: A data center network for large language model training," in *SIGCOMM*, 2024.

[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 *NSDI*, 2024.

[5] A. Gangidi, R. Miao, S. Zheng, S. J. Bondu, G. Goes, H. Morsy, R. Puri, M. Riftadi, A. J. Shetty, J. Yang *et al.*, "Rdma over ethernet for distributed ai training at meta scale," in *SIGCOMM*, 2024.

[6] Y. Gao, Q. Li, L. Tang, Y. Xi, P. Zhang, W. Peng, B. Li, Y. Wu, S. Liu, L. Yan *et al.*, "When cloud storage meets rdma," in *NSDI*, 2021.

[7] W. Bai, S. S. Abdeen, A. Agrawal, K. K. Attre, P. Bahl, A. Bhagat, G. Bhaskara, T. Brokhman, L. Cao, A. Cheema *et al.*, "Empowering azure storage with rdma," in *NSDI*, 2023.

[8] A. Kalia, M. Kaminsky, and D. Andersen, "Datacenter rpcs can be general and fast," in *NSDI*, 2019.

[9] S. K. Monga, S. Kashyap, and C. Min, "Birds of a feather flock together: Scaling rdma rpcs with flock," in *SOSP*, 2021.

[10] A. Kalia, M. Kaminsky, and D. G. Andersen, "Fasst: Fast, scalable and simple distributed transactions with two-sided rdma datagram rpcs," in *OSDI*, 2016.

[11] A. Dragojević, D. Narayanan, E. B. Nightingale, M. Renzelmann, A. Shamis, A. Badam, and M. Castro, "No compromises: distributed transactions with consistency, availability, and performance," in *SOSP*, 2015.

[12] X. Wei, J. Shi, Y. Chen, R. Chen, and H. Chen, "Fast in-memory transaction processing using rdma and htm," in *SOSP*, 2015.

[13] J. Shi, Y. Yao, R. Chen, H. Chen, and F. Li, "Fast and concurrent rdf queries with rdma-based distributed graph exploration," in *OSDI*, 2016.

[14] Y. Chen, X. Wei, J. Shi, R. Chen, and H. Chen, "Fast and general distributed transactions using rdma and htm," in *EuroSys*, 2016.

[15] X. Wei, Z. Dong, R. Chen, and H. Chen, "Deconstructing rdma-enabled distributed transactions: Hybrid is better!" in *OSDI*, 2018.

[16] IEEE, "IEEE 802.1 Qbb - Priority-based Flow Control," https://1.ieee802.org/dcb/802-1qbb/, 2011.

[17] C. Guo, H. Wu, Z. Deng, G. Soni, J. Ye, J. Padhye, and M. Lipshteyn, "Rdma over commodity ethernet at scale," in *SIGCOMM*, 2016.

[18] S. Wang, M. Zhang, Y. Du, Z. Chen, Z. Wang, M. Xu, R. Xie, and J. Yang, "Lordma: A new low-rate dos attack in rdma networks," in *NDSS*, 2024.

[19] 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," in *SIGCOMM*, 2015.

[20] Y. Jiang, Y. Zhu, C. Lan, B. Yi, Y. Cui, and C. Guo, "A unified architecture for accelerating distributed dnn training in heterogeneous gpu/cpu clusters," in *OSDI*, 2020.

[21] "Dcqcn parameters," 2024. [Online]. Available: https://enterprise-support.nvidia.com/s/article/dcqcn-parameters

[22] X. Kong, J. Chen, W. Bai, Y. Xu, M. Elhaddad, S. Raindel, J. Padhye, A. R. Lebeck, and D. Zhuo, "Understanding rdma microarchitecture resources for performance isolation," in *NSDI*, 2023.

[23] J. Lou, X. Kong, J. Huang, W. Bai, N. S. Kim, and D. Zhuo, "Harmonic: Hardware-assisted rdma performance isolation for public clouds," in *NSDI*, 2024.

[24] 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 *SIGCOMM*, 2019.

[25] S. Yan, X. Wang, X. Zheng, Y. Xia, D. Liu, and W. Deng, "Acc: Automatic ecn tuning for high-speed datacenter networks," in *SIGCOMM*, 2021.

[26] Y. Gao, Y. Yang, T. Chen, J. Zheng, B. Mao, and G. Chen, "Dcqcn+: Taming large-scale incast congestion in rdma over ethernet networks," in *ICNP*, 2018.

[27] B. Claise, "Cisco systems netflow services export version 9," Tech. Rep., 2004.

[28] M. Wang, B. Li, and Z. Li, "sflow: Towards resource-efficient and agile service federation in service overlay networks," in *ICDCS*, 2004.

[29] T. Yang, J. Jiang, P. Liu, Q. Huang, J. Gong, Y. Zhou, R. Miao, X. Li, and S. Uhlig, "Elastic sketch: Adaptive and fast network-wide measurements," in *SIGCOMM*, 2018.

[30] "Paraleon," 2024. [Online]. Available: https://github.com/czt8888/Paraleon-ns3

[31] C. H. Song, X. Z. Khooi, R. Joshi, I. Choi, J. Li, and M. C. Chan, "Network load balancing with in-network reordering support for rdma," in *SIGCOMM*, 2023.

[32] "Nccl-test," 2024. [Online]. Available: https://github.com/NVIDIA/nccl-tests

[33] S. Kullback and R. A. Leibler, "On information and sufficiency," *The annals of mathematical statistics*, vol. 22, no. 1, pp. 79–86, 1951.

[34] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan, "Data center tcp (dctcp)," in *SIGCOMM*, 2010.

[35] Z. Liu, A. Manousis, G. Vorsanger, V. Sekar, and V. Braverman, "One sketch to rule them all: Rethinking network flow monitoring with univmon," in *SIGCOMM*, 2016.

[36] M. Dong, Q. Li, D. Zarchy, P. B. Godfrey, and M. Schapira, "Pcc: Re-architecting congestion control for consistent high performance," in *NSDI*, 2015.

[37] M. Dong, T. Meng, D. Zarchy, E. Arslan, Y. Gilad, B. Godfrey, and M. Schapira, "Pcc vivace: Online-learning congestion control," in *NSDI*, 2018.

[38] T. Meng, N. R. Schiff, P. B. Godfrey, and M. Schapira, "Pcc proteus: Scavenger transport and beyond," in *SIGCOMM*, 2020.

[39] 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 *SIGCOMM*, 2020.

[40] A. K. Gupta, K. G. Smith, and C. E. Shalley, "The interplay between exploration and exploitation," *Academy of management journal*, vol. 49, no. 4, pp. 693–706, 2006.

[41] "Network simulator 3," 2024. [Online]. Available: https://www.nsnam.org

[42] "grpc," 2024. [Online]. Available: https://grpc.io

[43] "Elasticsketch," 2024. [Online]. Available: https://github.com/BlockLiu/ElasticSketchCode

[44] A. Roy, H. Zeng, J. Bagga, G. Porter, and A. C. Snoeren, "Inside the social network's datacenter network," in *SIGCOMM*, 2015.

[45] J. Liu, J. H. Wang, and Y. Jiang, "Janus: A unified distributed training framework for sparse mixture-of-experts models," in *SIGCOMM*, 2023.

[46] R. Miao, L. Zhu, S. Ma, K. Qian, S. Zhuang, B. Li, S. Cheng, J. Gao, Y. Zhuang, P. Zhang *et al.*, "From luna to solar: the evolutions of the compute-to-storage networks in alibaba cloud," in *SIGCOMM*, 2022.

[47] W. Li, X. Liu, Y. Li, Y. Jin, H. Tian, Z. Zhong, G. Liu, Y. Zhang, and K. Chen, "Understanding communication characteristics of distributed training," in *APNet*, 2024.

[48] H. Zheng, C. Huang, X. Han, J. Zheng, X. Wang, C. Tian, W. Dou, and G. Chen, "μmon: Empowering microsecond-level network monitoring with wavelets," in *SIGCOMM*, 2024.

[49] R. Mittal, V. T. Lam, N. Dukkipati, E. Blem, H. Wassel, M. Ghobadi, A. Vahdat, Y. Wang, D. Wetherall, and D. Zats, "Timely: Rtt-based congestion control for the datacenter," in *SIGCOMM*, 2015.

[50] Q. Huang, P. P. Lee, and Y. Bao, "Sketchlearn: Relieving user burdens in approximate measurement with automated statistical inference," in *SIGCOMM*, 2018.

[51] Y. Zhang, Z. Liu, R. Wang, T. Yang, J. Li, R. Miao, P. Liu, R. Zhang, and J. Jiang, "Cocosketch: High-performance sketch-based measurement over arbitrary partial key query," in *SIGCOMM*, 2021.
[52] H. Namkung, Z. Liu, D. Kim, V. Sekar, and P. Steenkiste, "Sketchlib: Enabling efficient sketch-based monitoring on programmable switches," in *NSDI*, 2022.
[53] Z. Liu, R. Ben-Basat, G. Einziger, Y. Kassner, V. Braverman, R. Friedman, and V. Sekar, "Nitrosketch: Robust and general sketch-based monitoring in software switches," in *SIGCOMM*, 2019.
[54] H. Namkung, Z. Liu, D. Kim, V. Sekar, and P. Steenkiste, "Sketchovsky: Enabling ensembles of sketches on programmable switches," in *NSDI*, 2023.