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

Ziteng Chen[1], Menghao Zhang[2,3], Guanyu Li[4], Mingwei Xu[2,4,5]
[1] School of Cyber Science and Engineering, Southeast University
[2] Department of Computer Science and Technology, Tsinghua University [3] Kuaishou Technology
[4] Institute for Network Sciences and Cyberspace, Tsinghua University [5] Quan Cheng Laboratory

CCS CONCEPTS

• **Networks** → **Transport protocols**;

KEYWORDS

Remote Direct Memory Access; Congestion Control

ACM Reference Format:
Ziteng Chen, Menghao Zhang, Guanyu Li, Mingwei Xu. 2023. Chameleon: Automatic and Adaptive Tuning for DCQCN Parameters in RDMA Networks. In *ACM SIGCOMM 2023 Conference (ACM SIGCOMM '23), September 10, 2023, New York, NY, USA.* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3603269.3610865

1 INTRODUCTION

Datacenter Quantized Congestion Notification (DCQCN) [12] is the default congestion control algorithm for Mellanox RDMA (Remote Direct Memory Access) NICs [2] in RoCEv2 (RDMA over Converged Ethernet v2) networks, one of the most widely used NICs in leading industry companies [4, 5, 7, 9]. In DCQCN, firstly switches mark packets with ECN (Explicit Congestion Notification) when the queue length exceeds ECN thresholds, then receivers respond to ECN-marked packets with CNPs (Congestion Notification Packets), and finally senders reduce transmission rate when receiving CNPs. DCQCN has 10+ parameters at both NICs and switches, including Alpha Update, Rate Increase & Decrease, Notification Point and ECN thresholds [3], and these parameters have a non-negligible impact on the network performance. Our experiments also verify the network performance of common AI (Artificial Intelligence) training workloads in RoCEv2 networks (e.g., all-to-all collective communication) is greatly influenced by different DCQCN parameter settings (§3). Therefore, when deploying applications in practice, the DCQCN parameters need to be carefully tested and tuned to improve the network performance.

However, current DCQCN parameter setting mainly relies on manual efforts of experts, which may take several weeks or even months [8]. First, different workloads have various SLAs (Service Level Agreement) and properties, including delay, throughput, FCT (Flow Completion Time), message patterns, RDMA transport types, etc. However, the default parameters cannot satisfy different requirements simultaneously. Consequently, operators have to manually tune a specific DCQCN parameter setting for each workload. Second, a static parameter setting always falls behind dynamic network environment. Especially, heterogeneous workloads can coexist and share the network, and be incrementally deployed at any time. Therefore, operators have to repeat the time-consuming tuning process in real RoCEv2 networks.

This work is supported by the National Natural Science Foundation of China (No. 62221003), China Postdoctoral Science Foundation (2022M720202), and Beijing Postdoctoral Research Foundation (2022-ZZ-078). Mingwei Xu is the corresponding author.

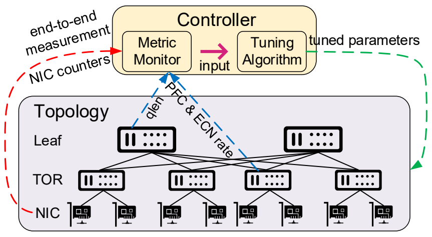

Figure 1: Chameleon overall architecture.

Researchers have proposed some automatic tuning tools for DCQCN parameters. ACC [11] automatically tunes the switch ECN thresholds based on runtime information collected from switches, but ignores DCQCN parameters at NICs. DCQCN+ [6] adjusts the additive increase step and recovery timer according to the incast scale, but ignores other parameters and needs significant protocol modification. Wang *et al.* [10] adaptively optimizes RDMA transport types based on message sizes and polling strategies, but overlooks the DCQCN parameters at both NICs and switches. To summarize, existing works fail to take DCQCN parameters at both NIC-side and switch-side into account simultaneously.

Nevertheless, designing an automatic and adaptive tuning tool for DCQCN parameters is non-trivial. The first challenge is to be aware of the network dynamics. In an RDMA cluster, heterogeneous and incremental workloads can occur temporally and spatially, making it difficult to get full knowledge of the dynamic traffic patterns and distributions. Therefore, we need a lightweight and effective way to monitor and collect necessary runtime metrics at both NICs and switches, to further adjust DCQCN parameters accordingly. The second challenge is to automatically tune parameters in an effective and efficient way. Since DCQCN has 10+ parameters, it is hard to search the optimal parameter setting directly, given the complicated relationship between parameters and network performance. Therefore, we need a tuning algorithm, which not only makes a proper tuning decision adaptive to dynamic network environment, but also determines the parameters within a short period.

In this poster, we propose Chameleon to tune DCQCN parameters automatically and adaptively, as depicted in Figure 1. Chameleon consists of two modules, *Runtime Metric Monitor* and *Adaptive Tuning Algorithm*. The monitor aims at collecting and monitoring necessary runtime metrics from NICs and switches.

Based on collected metrics, the algorithm tunes the DCQCN parameters and dispatches them to NICs and switches. Experiment results show that Chameleon achieves lower tail FCT and higher throughput than the default parameter setting [3] and expert setting.

2 CHAMELEON DESIGN

2.1 Runtime Metric Monitor

The Runtime Metric Monitor module is designed to monitor runtime metrics from NICs and switches. It consists of a centralized controller, allowing NICs and switches to upload their metric data periodically. Chameleon establishes TCP connections between the controller and each host & switch's control plane to provide reliable transport. To avoid interference with RDMA traffic, RDMA and TCP are assigned to different queues by different scheduling policies. We also consider the controller placement according to network topology and realtime traffic. Currently, we place the controller to minimize the aggregated hops to NICs and switches.

We encounter two main challenges when designing the monitor. The first one is to select the useful metrics. Since there are numerous metrics we can acquire from NICs and switches, monitoring all of them will cause huge bandwidth and computing overhead. What's worse, some metrics are not helpful for reflecting network status, which may mislead the tuning algorithm to make poor decisions. We observe these numerous metrics can be categorized into informative and error types: the former represents realtime performance while the latter records error and exception handling. Therefore, we currently choose several informative metrics that indicate end-to-end performance and network congestion scale. First, NICs launch periodic end-to-end measurements and estimate runtime throughput and RTT. Second, NICs upload several counter metrics indicating the number of received PFCs and CNPs, while switches provide current queue length, generated & received PFCs and ECN marking rate.

The second challenge is to monitor metrics more efficiently. On one hand, due to inevitable network jitters, NICs and switches may obtain some incomplete or unstable raw metrics, which cannot indicate the real performance accurately. On the other hand, tuning parameters too frequently is unnecessary, which not only causes more jitters, but also wastes bandwidth and computing resources. We notice the local control planes of hosts and switches can provide computing power for simple data preprocessing and cache. Therefore, Chameleon employs a layered monitor mechanism: every small period (e.g., 10ms), hosts and switch control planes first filter the low-quality raw metrics, then compute some statistics (e.g., mean, sketches, gradient), and finally upload them to the controller for parameter tuning every long period (e.g., 100ms).

2.2 Adaptive Tuning Algorithm

We develop the Adaptive Tuning Algorithm module to tune DCQCN parameters dynamically, which takes collected metrics as input and outputs the parameters for NICs and switches. For NIC-side parameters, the algorithm tunes the steps and timers of rate Increase & Decrease, update period of α, CNP generation interval, etc. For switch-side parameters, the algorithm tunes the ECN thresholds at each switch, i.e., $K_{min}, K_{max}, P_{max}$. These tuned parameters are dispatched synchronously to guarantee global fairness.

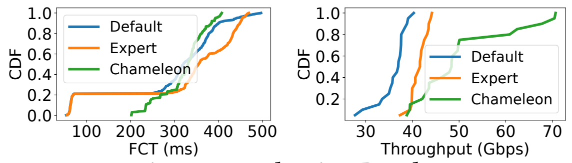

Figure 2: Evaluation Results.

Since DCQCN has 10+ parameters, it is difficult to obtain an optimal setting in polynomial time. Therefore, we resort to a greedy algorithm to select and adjust one parameter in each round. We define m_{i-1}^{p} and m_{i}^{p} as the metrics in the previous round r_{i-1} and current round r_i when adjusting parameter p. We aim at improving m_i^p after tuning p, for example, higher throughput, lower RTT and fewer generated PFCs/CNPs. We define v_i^p and s_i^p as the value and tuning step for parameter p at round r_i, and the algorithm adjusts p by $v_{i+1}^{p} = v_i^p \pm s_i^p$ from round r_i to r_{i+1}. The tuning step for each p is different, and s_i^p gets smaller with increasing rounds. In principle, the tuning step should get more fine-grained when p is approaching the ideal value. Therefore, we set initial s_1^p mainly based on our experience, and let $s_{i+1}^{p} = \lambda s_i^p$. We notice a tradeoff between tuning convergence and accuracy with different values of λ, and we empirically set $\lambda = 0.95$ in this poster.

Let's take an example. Suppose the default tuning direction of p is incremental, and we are trying to maximize system performance according to collected throughput metrics. If throughput gets obvious improvement from r_{i-1} to r_i, say $\frac{m_i^p - m_{i-1}^p}{m_{i-1}^p} \geq \beta$, the algorithm will keep $v_{i+1}^{p} = v_i^p + s_i^p$ in round r_{i+1}. If throughput gets worse, i.e., $m_i^p < m_{i-1}^p$, it means the tuning direction for p is wrong, thus the algorithm will decrement p after round r_{i+1}, i.e., $v_{i+1}^{p} = v_i^p - s_i^p$. If throughput gets slight or little improvement for several rounds, i.e., $0 < \frac{m_i^p - m_{i-1}^p}{m_{i-1}^p} < \beta$, it implies further tuning for p is not helpful anymore, and the algorithm will stop tuning p and start with another parameter. The tuning process will stop until all DCQCN parameters are traversed or the network performance satisfies operators' demand.

3 EVALUATION & FUTURE WORK

We implement and evaluate Chameleon based on NS3 [1]. We use a two-tier clos topology (Figure 1) with 100Gbps link bandwidth and 5μs propagation delay. We launch a 20×20 all-to-all workload with 100MB flow size, which is common in RDMA-based AI training clusters. In this scenario, Chameleon completes the parameter decision within 200 rounds, thus the tuning process is finished within 10+ seconds. In terms of FCT and throughput, Chameleon is compared with 2 parameter settings provided by NVIDIA/Mellanox [3] and by an expert according to his experience. Figure 2 shows the CDFs of FCT and throughput. Chameleon achieves 19.7% and 13.2% lower tail FCT, and 40.5% and 25.1% larger average throughput than the default and expert settings, respectively. In future, we plan to 1) investigate more useful metrics and develop a more collaborative monitor mechanism; 2) explore the relationship between different parameters and use more intelligent algorithms for better tuning convergence and accuracy; 3) implement all details and conduct more evaluations under different environment settings and workloads.

REFERENCES

[1] 2020. High Precision Congestion Control. (2020). https://github.com/alibaba-edu/High-Precision-Congestion-Control

[2] 2022. DCQCN CC Algorithm. (2022). https://enterprise-support.nvidia.com/s/article/DCQCN-CC-algorithm

[3] 2023. DCQCN Parameters. (2023). https://enterprise-support.nvidia.com/s/article/dcqcn-parameters

[4] Wei Bai, Shanim Sainul Abdeen, Ankit Agrawal, Krishan Kumar Attre, Paramvir Bahl, Ameya Bhagat, Gowri Bhaskara, Tanya Brokhman, Lei Cao, Ahmad Cheema, et al. 2023. Empowering Azure Storage with RDMA. In *NSDI*.

[5] Yixiao Gao, Qiang Li, Lingbo Tang, Yongqing Xi, Pengcheng Zhang, Wenwen Peng, Bo Li, Yaohui Wu, Shaozong Liu, Lei Yan, et al. 2021. When Cloud Storage Meets RDMA. In *NSDI*.

[6] Yixiao Gao, Yuchen Yang, Tian Chen, Jiaqi Zheng, Bing Mao, and Guihai Chen. 2018. Dcqcn+: Taming large-scale incast congestion in rdma over ethernet networks. In *ICNP*.

[7] Yimin Jiang, Yibo Zhu, Chang Lan, Bairen Yi, Yong Cui, and Chuanxiong Guo. 2020. A unified architecture for accelerating distributed DNN training in heterogeneous GPU/CPU clusters. In *OSDI*.

[8] Yuliang Li, Rui Miao, Hongqiang Harry Liu, Yan Zhuang, Fei Feng, Lingbo Tang, Zheng Cao, Ming Zhang, Frank Kelly, Mohammad Alizadeh, et al. 2019. HPCC: High precision congestion control. In *SIGCOMM*.

[9] Dheevatsa Mudigere, Yuchen Hao, Jianyu Huang, Zhihao Jia, Andrew Tulloch, Srinivas Sridharan, Xing Liu, Mustafa Ozdal, Jade Nie, Jongsoo Park, et al. 2022. Software-hardware co-design for fast and scalable training of deep learning recommendation models. In *ISCA*.

[10] Kai Wang, Fang Dong, Dian Shen, Chengtian Zhang, Jinghui Zhang, and Junzhou Luo. 2021. Towards tunable RDMA parameter selection at runtime for datacenter applications. In *CSCWD*.

[11] Siyu Yan, Xiaoliang Wang, Xiaolong Zheng, Yinben Xia, Derui Liu, and Weishan Deng. 2021. ACC: Automatic ECN tuning for high-speed datacenter networks. In *SIGCOMM*.

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