CAVER: Enhancing RDMA Load Balancing by Hunting Less-Congested Paths

Haotian Deng[1], Yuan Yang[1], Menghao Zhang[2], Mingwei Xu[1]
[1]Tsinghua University [2]Beihang University

CCS CONCEPTS

• **Networks** → **Data path algorithms**; *Programmable networks.*

KEYWORDS

Remote Direct Memory Access; In-Network Load Balancing; Programmable Switches;

ACM Reference Format:
Haotian Deng, Yuan Yang, Menghao Zhang, Mingwei Xu. 2024. CAVER: Enhancing RDMA Load Balancing by Hunting Less-Congested Paths. In *ACM SIGCOMM 2024 Conference (ACM SIGCOMM Posters and Demos '24), August 4–8, 2024, Sydney, NSW, Australia.* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3672202.3673729

1 INTRODUCTION

Remote Direct Memory Access (RDMA) has become a prevailing technology for modern data centers (DCs) to achieve high throughput and low latency [8]. Many DCs have adopted RDMA over Converged Ethernet v2 (RoCEv2) [3] to provide superior performance for emerging application paradigms such as cloud storage [4] and distributed deep learning [11]. Network load balancing (LB) plays a critical role in optimizing the DC network performance. There is a large body of literature studying LB for traditional DCs [1, 2, 6, 7, 9], as it is well-known that the widely-used ECMP [12] has a limited LB performance. However, RDMA operates in a different manner compared to traditional TCP-based data transmission, and existing studies for traditional DCs do not well fit RDMA-enabled DCs. For example, RDMA is very sensitive to out-of-order packets which may lead to significant throughput degradation, and also, RDMA flow can hardly be partitioned into flowlets. Thus, existing packet-level LB approaches [6, 7] and flowlet-level LB approaches [2] perform poorly in RDMA-enabled DCs.

Researchers have realized that a key for good LB performance is to find out which path/link is congested, and LB decisions should be congestion aware. Thus, efforts are made on finding and using congestion signals (in both traditional DCs and RDMA-enabled DCs) [15]. For instance, CONGA [2] detects path congestion degrees leveraging flows being transmitted. The information is sent back to the source Top of Rack (ToR) switch, based on which routing decisions are made for subsequent flows. ConWeave [17] detects congestion by observing packet Round-Trip Times (RTTs), and a rerouting method is proposed to mitigate congestion. Proteus [10] combines multiple metrics to detect congestion more precisely. However, existing approaches still have a key limitation. That is, the path space is not fully explored by the flows traversing a ToR switch. In particular, the total number of available paths between a source-destination pair increases rapidly with the network scale and number of layers, while existing approaches can only detect the congestion degrees for only a small portion of these paths. Thus, a number of paths that are less congested will be overlooked when routing subsequent flows or rerouting existing flows. Furthermore, the widely-used bond scenario exacerbates the problem, where a host connects to two ToR switches and paths that may not be detected will increase.

To resolve these challenges, in this paper, we propose CAVER, a **C**ongestion-**A**ware LB approach for RDMA with **VE**cto**R** protocol. CAVER leverages acknowledgement (ACK) packets to realize a vector protocol to hunt less-congested paths for each source-destination pair in real time. In RoCEv2, each data packet will trigger an ACK packet to guarantee that no packet loss occurs. Therefore, it is reasonable to assume that plenty of ACK packets are traversing all links in the network at any time. A switch can thus announce local congestion condition info to neighboring switches by piggybacks using the ACK packets. More specifically, a switch tells a neighboring switch the least-congested path to a certain destination. Then, the neighboring switch computes the least-congested path for itself based on the received paths, and announces the results, similar to traditional vector routing protocols such as RIP [14] and BGP [16]. CAVER can guarantee loop-freeness and converge quickly in just one RTT. As a result, a source ToR switch is aware of the least-congested path to a certain destination, and all packets of the incoming flow will be forwarded along the least-congested path using source routing, so as to guarantee stability and avoid routing oscillation. Then, the least-congested path is updated for the next incoming flow. We implement CAVER with the NS3 simulator, and conduct experiments with real traffic traces [13]. The results show that CAVER can decrease the average flow completion time (FCT) slowdown by up to 15% and 10% compared with ECMP and ConWeave, respectively.

2 CAVER DESIGN

Figure 1 depicts the overview of CAVER. Most of the CAVER functionalities reside at the source ToR switch, including a Flow Table to conduct source routing and a Path Congestion Table to maintain path congestion metrics. Functionalities at other switches are relatively simple, mainly with a Path Congestion Table to record and deliver the path congestion metrics. Considering RDMA is sensitive to out-of-order packets and it is rare to see flowlets in RDMA

This work was supported by the National Key Research and Development Program of China under Grant 2022YFB2901300, the National Natural Science Foundation of China under Grant 62221003, and the Beijing Science and Technology Plan Project under Grant Z231100010323013.

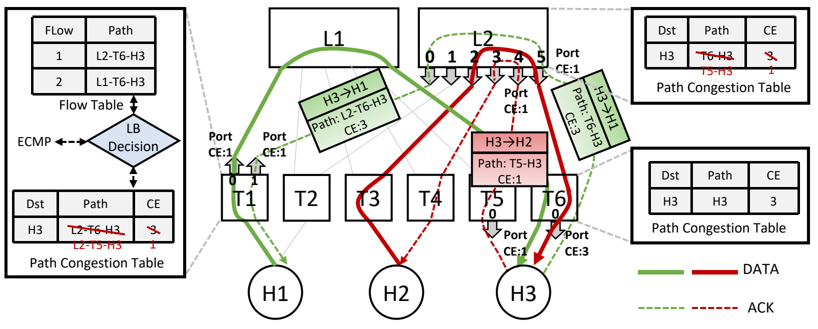

Figure 1: CAVER overview.

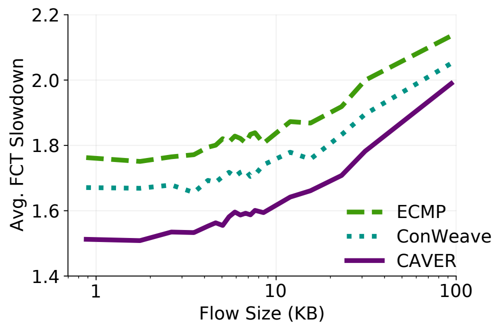

Figure 2: Evaluation Results.

flows [17], CAVER uses flow-level routing, where the packets of a flow are transported through the same path.

For the first packet of a flow, LB decision is made based on the Path Congestion Table at the source ToR switch, which maintains the least-congested path and corresponding congestion extent (CE) for each destination. The least-congested path is then added to the Flow Table, to instruct subsequent packets of the flow to traverse the same path. To save the switch memory, CAVER uses the hash value of the 5-tuple as the key for the Flow Table. Although hashing collisions are inevitable, this only makes a few flows lost some LB opportunities and does not incur any correctness issues. If a packet does not match the Flow Table and the Congestion Table, it implies that the CE has not been recorded for any path to the destination, and CAVER resorts to ECMP.

At the core of CAVER is updating the Path Congestion Table, where ACK packet of each data packet is leveraged to piggyback the congestion metrics from destination host to source ToR switch hop by hop, which is detailed below.

Update the Path Congestion Table. CAVER leverages the Discounting Rate Estimator (DRE) at each switch port to estimate CE in a way similar to CONGA [2], so as to update the Path Congestion Table. The DRE is increased for each data packet by the packet size in bytes, and is decreased periodically with a multiplicative factor α between 0 and 1: $DRE \leftarrow DRE \times (1-\alpha)$. Each data packet triggers an ACK packet in RoCEv2, which is forwarded back to the source ToR switch with ECMP. CAVER employs the ACK packet to piggyback the path congestion metrics (including a path field and a CE field) to the source ToR switch hop by hop. The point is that ACK packets of different flows terminated at a destination cooperate to compute the least-congested path to the destination following our vector protocol. We use the example in Figure 1 to illustrate the process.

We first consider the green flow in Figure 1. When an ACK packet arrives at destination ToR switch T6, the switch obtains the port CE value based on DRE at the ingress port, and updates the Path Congestion Table with source IP as the key. In particular, T6 records in the Path Congestion Table that the least-congested path to reach destination H3 is through port 0 with a CE value of 3. Then path T6-H3 and CE value 3 is carried by the ACK packet which is forwarded to the next hop, i.e., L2. L2 adds itself into the carried path and computes the maximum CE value along the path. If the resulting path is the least-congested path to the destination, L2 records the path and corresponding CE in the Path Congestion Table. Then, the ACK packet carries the current least-congested path (i.e., L2-T6-H3) and corresponding CE value (i.e., 3), and is forwarded to T1. Finally, with similar steps, T1 stores the least-congested path and corresponding CE value in the Path Congestion Table. Note that the CE value carried by the ACK packets of the same flow may change. In such a case, the switches along the path need to update the Path Congestion Tables accordingly.

Now let us consider the red flow in Figure 1 which arrives later than the green flow. Similarly, an ACK packet is sent from T5 to L2, carrying path T5-H3 and CE value 1. When this packet arrives at L2, L2 finds that the path for this in-port is less congested, so the Path Congestion Table needs to update accordingly. That is, path T5-H3 substitutes path T6-H3 as the least-congested path to reach H3, with the smaller CE value 1. It is worth noting that in CAVER, ACK packets can always piggyback the least-congested path from local switch to neighboring switch. Thus, the new path (i.e., L2-T5-H3) and the new CE value (i.e., 1) will be announced to both T1 and T4 by ACK packets of the green flow and the red flow respectively (which are not illustrated in Figure 1). Finally, the Path Congestion Table of T1 is updated to use path L2-T5-H3 with CE value 1. Newly incoming flows at T1 can query the Path Congestion Table to obtain the least-congested path. Note that after an incoming flow chooses the current least-congested path, the new least-congested path will be ready within one RTT, because the least-congested path is carried by every ACK packet.

3 EVALUATION AND FUTURE WORK

We implement a prototype of CAVER based on NS3, which is publicly available at Github [5]. We use a three-tier fat-tree (k = 4) topology and 8 servers, each server connect to two ToR switches (i.e., bond) in the same pod. Each link has a bandwidth of 100Gbps. We use the AliCloud storage workloads [13] in the simulation. We use FCT slowdown as metric, i.e., a flow's actual FCT normalized by the base FCT when the network has no other traffic. As shown in Figure 2, compared with ECMP and ConWeave, CAVER decreases the average FCT slowdown by up to 15% and 10% respectively.

In future, we plan to optimize the mechanisms and resource utilization of CAVER, implement all the details with programmable switches, and conduct extensive evaluations under different environment settings and workloads.

REFERENCES

[1] Mohammad Al-Fares, Sivasankar Radhakrishnan, Barath Raghavan, Nelson Huang, Amin Vahdat, et al. 2010. Hedera: dynamic flow scheduling for data center networks.. In *Nsdi*, Vol. 10. San Jose, USA, 89–92.

[2] Mohammad Alizadeh, Tom Edsall, Sarang Dharmapurikar, Ramanan Vaidyanathan, Kevin Chu, Andy Fingerhut, Vinh The Lam, Francis Matus, Rong Pan, Navindra Yadav, et al. 2014. CONGA: Distributed congestion-aware load balancing for datacenters. In *Proceedings of the 2014 ACM conference on SIGCOMM*. 503–514.

[3] InfiniBand Trade Association. 2020. InfiniBand Architecture Specification Release 1.4 Annex A17: RoCEv2.

[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 *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*. 49–67.

[5] CAVER. 2024. CAVER-NS3. https://github.com/denght23/CAVER.git.

[6] Advait Dixit, Pawan Prakash, Y Charlie Hu, and Ramana Rao Kompella. 2013. On the impact of packet spraying in data center networks. In *2013 proceedings ieee infocom*. IEEE, 2130–2138.

[7] Soudeh Ghorbani, Zibin Yang, P Brighten Godfrey, Yashar Ganjali, and Amin Firoozshahian. 2017. Drill: Micro load balancing for low-latency data center networks. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*. 225–238.

[8] Chuanxiong Guo, Haitao Wu, Zhong Deng, Gaurav Soni, Jianxi Ye, Jitu Padhye, and Marina Lipshteyn. 2016. RDMA over commodity ethernet at scale. In *Proceedings of the 2016 ACM SIGCOMM Conference*. 202–215.

[9] Keqiang He, Eric Rozner, Kanak Agarwal, Wes Felter, John Carter, and Aditya Akella. 2015. Presto: Edge-based load balancing for fast datacenter networks. *ACM SIGCOMM Computer Communication Review* 45, 4 (2015), 465–478.

[10] Jinbin Hu, Chaoliang Zeng, Zilong Wang, Junxue Zhang, Kun Guo, Hong Xu, Jiawei Huang, and Kai Chen. 2023. Enabling load balancing for lossless datacenters. In *2023 IEEE 31st International Conference on Network Protocols (ICNP)*. IEEE, 1–11.

[11] 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 *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*. 463–479.

[12] Petr Lapukhov, Ariff Premji, and Jon Mitchell. 2016. *Use of BGP for Routing in Large-Scale Data Centers.* RFC 7938. RFC Editor.

[13] 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 *Proceedings of the ACM Special Interest Group on Data Communication*. 44–58.

[14] Gary Malkin. 1998. *RIP version 2.* Technical Report.

[15] Mubashir Adnan Qureshi, Yuchung Cheng, Qianwen Yin, Qiaobin Fu, Gautam Kumar, Masoud Moshref, Junhua Yan, Van Jacobson, David Wetherall, and Abdul Kabbani. 2022. PLB: congestion signals are simple and effective for network load balancing. In *Proceedings of the ACM SIGCOMM 2022 Conference*. 207–218.

[16] Yakov Rekhter, Tony Li, and Susan Hares. 2006. *A border gateway protocol 4 (BGP-4).* Technical Report.

[17] Cha Hwan Song, Xin Zhe Khooi, Raj Joshi, Inho Choi, Jialin Li, and Mun Choon Chan. 2023. Network Load Balancing with In-network Reordering Support for RDMA. In *Proceedings of the ACM SIGCOMM 2023 Conference*. 816–831.