FaCa: Fast Aware and Competition-Avoided Balancing for Data Center Network

Haiyang Jiang[1], Yuchao Zhang[1,*], Haoqiang Huang[1], Lei Wang[1], Xirong Que[1], Zhuo Jiang[2,*], and Wendong Wang[1]

[1] Beijing University of Posts and Telecommunications, Beijing, China
{jianghaiyang,yczhang,hhq_erii,lwang,rongqx,wdwang}@bupt.edu.cn
[2] ByteDance, Beijing, China
{zjiang}@bytedance.com

Abstract. Nowadays, the scale of business data is expanding at an unprecedented rate. To cater to the needs of large businesses, data center networks (DCNs) have been widely deployed and are continuing to expand. However, the influx of large-scale concurrent flows into DCNs often results in network congestion due to the concurrent competition for resources. While existing load balancing mechanisms can handle concurrent competition, they often do so at the cost of time. As a result, there is currently no ideal solution that effectively addresses both time consumption and concurrent competition issues. In this paper, we present a novel load balancing solution called FaCa, which runs on the host-end in a completely software-based manner. FaCa incorporates Inband Network Telemetry (INT), leveraging traffic transmission within the network to swiftly obtain a partial global view of network load. Additionally, we propose the Flowing&Jumping algorithm to mitigate concurrent path competition by introducing an element of randomness to load balancing process. FaCa is easy to deploy and has demonstrated superior performance compared to other mechanisms. Our evaluation on production DCN reveals that FaCa incurs minimal additional time overhead while achieving better load balancing results compared to existing approaches. Specifically, it resulted in a 14.28% reduction in congestion and a 22.5% increase in host throughput.

Keywords: Multi-Node HPC Cluster · DCN · Load Balancing

1 Introduction

Given the rapid expansion of the Internet, the volume of generated data has reached unprecedented levels, placing significant strain on data processing capa-

* Corresponding author

This work is supported in part by the National Natural Science Foundation of China (NSFC) under Grant 62172054 and 62072047, the National Key R&D Program of China under Grant 2019YFB1802603, and the BUPT-ByteDance Research Project.

bilities. According to a report by the International Data Corporation (IDC)[34], the total volume of data is projected to reach 163ZB until 2025. To effectively handle this data pressure, enterprises have been establishing high-performance computing (HPC) clusters within their data center networks (DCNs). To optimize the performance of HPC clusters, researchers have developed high-speed communication technologies, such as the All-Reduce[27] communication mode, to enhance the performance of HPC clusters. With the consideration of reliability, DCN also adopts multi-path design to connect hosts, along with interconnection architectures such as the Clos architecture[5] in networking. Based on existing resources, the requirement for an efficient load balancing method in DCNs is urgent to enhance overall performance. Nevertheless, the majority of existing load balancing methods are primarily designed for conventional cloud-scale networks and might not be well suited for DCNs with HPC clusters. This is due to the unique characteristics of DCNs with HPC clusters, which differ from traditional networks and necessitate specialized load balancing approaches[16]. One of the challenges in load balancing for DCNs is the presence of concurrent flows. In a DCN, when multiple hosts initiate communications simultaneously, a significant volume of data is concurrently transmitted within the network. These concurrent flows exhibit characteristics such as simultaneous generation, partial inclusion of large flows, and bursts. Existing load balancing algorithms often face challenges in effectively managing such flows.

The load balancing methods can be categorized into two types: **centralized** and **distributed**. The centralized mechanism[3, 10, 16, 30, 39, 38] collects network information periodically to calculate globally optimal load balancing strategies. However, it experiences delays due to the central controller waiting for and retrieving information from network devices. This delay significantly slows down the load balancing process, especially in DCNs with concurrent flows. The other approach is based on a distributed method. Some of them focus on hardware upgrades such as Ananta[28], which upgrades multiplexers (Mux), and SilkRoad[24], which utilizes load balancing ASICs. However, these designs require costly development of dedicated load balancing hardware, limiting their adoption in DCNs.

The most common distributed solutions[9, 11–14, 18, 19, 26, 32, 33, 36] focus on strategies for achieving load balancing by leveraging existing devices, such as programmable switches[12] and back-end servers[26]. This type of approach selects the path with the lowest load for distributing elephant flows, but it overlooks the issue of "concurrency competition". When two nodes simultaneously detect the same path with the lowest load, conflicts arise on shared links when both nodes transmit traffic concurrently. This leads to a significantly higher load on the common link compared to other links and creates the potential for network congestion.

Designing load balancing for DCN is a challenging task that involves balancing **fast reaction** and **competition avoidance** while preserving network performance. Collecting network load information typically requires communication between central controllers and remote network devices, which often incurs

$\geq$ 1 RTTs[3, 30, 38, 39]. Load balancing scheduling introduces a time delay that impacts the timeliness of the mechanism. In a distributed algorithm, nodes receive identical network load information within the same network environment. This can lead to nodes making identical scheduling decisions, resulting in concurrent flows competing for bandwidth on shared links and causing network congestion.

In this paper, we introduce *FaCa*, a load balancing solution that addresses the aforementioned problems. FaCa utilizes In-band Network Telemetry (INT) to quickly respond to network imbalance by leveraging network-transmitted packets to probe real-time network load without additional time consumption. In addition, we propose Flowing&Jumping, a distributed load balancing algorithm that mitigates concurrent competition. Flowing&Jumping schedules flows in a manner resembling water flow and introduces random "jump" to prevent path competition. This simple integration of randomness effectively avoids conflicts. Furthermore, FaCa is designed as software-based and built upon the foundation of Equal-Cost Multi-Path (ECMP)[14], which ensures easy deployment and will be explained in Section IV. The **main contributions** of this paper are as follows:

- We identify and highlight the existing problems in load balancing that can potentially diminish network performance.
- We propose FaCa, a load balancing method that utilizes INT to enable quick congestion reaction, and introduces Flowing&Jumping to mitigate concurrent competition.
- We develop a prototype of FaCa within a production DCN and carry out performance evaluations. The findings showcase that FaCa achieved a substantial 22.5% increase in throughput and a noteworthy 14.28% reduction in congestion.

The paper is structured as follows. In Section 2, we introduce prior research on network load balancing. In Section 3, we identifie existing issues that degrade network performance. In Section 4, we present insights on resilience and concurrent competition. In Section 5, we explain the load balancing system FaCa, including its modular introduction. In Section 6, we discuss the concrete challenges and implementation environment of FaCa. Finally, we prototype and evaluate FaCa's performance in Section 7 and conclude in Section 8.

2 Related Work

This section briefly introduces existing load balancing methods and their limitations. Load balancing approaches can be categorized into two groups: centralized and distributed methods.

2.1 Centralized Load Balancing

Centralized load balancing mechanisms typically employ logically centralized controllers to gather information on links (such as bandwidth, utilization, etc.)

and switches (including buffer size, queue length, etc.). Using this comprehensive network information, the centralized controllers distribute flows to less utilized paths.

Mahout [10] improves on Hedera by using SDN technology to reduce the overhead of identifying elephant flows. It also brings about time consumption of collecting network information. Freeway[39] divides links into low-latency oriented links (LOL) and high-throughput orient links (HOL) by estimating the link utilization. Different from the methods that distribute flows based on the amount of bytes, FDALB[38] divides flows into long-lived flows and short-lived flows. For long-lived flows, FDALB marks them on end-hosts and globally schedules such flows with a greedy and polling algorithm. For short-lived flows, it simply continues to use ECMP. MicroTE[3] utilizes OpenFlow DCN to forecast traffic patterns. It classifies flows into predictable and unpredictable categories. Predictable flows are assigned load balancing tasks by centralized controllers, while unpredictable flows make use of the default balancing mechanisms of the network, such as ECMP and WCMP [45]. Fastpass[30] achieves load balancing at the packet level by determining transmission time slots and paths for each packet using centralized controllers.

While centralized methods are theoretically capable of achieving optimal performance, the practical implementation is hindered by the time-consuming process of information collection and the overhead of redundant communication. Additionally, the fixed interval at which information is collected limits the ability to quickly respond to changes in network topology due to failures or hardware restarts. In real-world scenarios, centralized methods often struggle with coarse-grained scheduling, making it challenging to meet the desired level of accuracy in load balancing mechanisms.

2.2 Distributed Load Balancing

Distributed mechanisms are locally deployed on hosts and switches, which often make decisions based on local information, resulting in much faster reaction compared to centralized mechanisms. Nevertheless, they are not immune to competition issues.

As the first method to introduce Mux, Ananta[28] designs a multi-layer structure. It provides ECMP in the third layer and schedules connections in the forth layer. Duet[12] manages to develop load balancing with the API on programmable switches and combines it with Mux to provide a resilience to network failures. Silkroad[24] addresses the shortcomings of stateful load balancing by presenting a stateless solution based on switching ASIC development. This approach offloads load balancing from switches and leverages dedicated hardware for efficient and high-performance load balancing. However, the deployment overhead and scalability remain as challenges. FLARE[18] introduces the load balancing method called FLowlet and highlights a characteristic of TCP. It states that if the interval between sending two packets is greater than the delay of parallel paths, packet transportation on these paths will not cause out-of-order problems. Building upon the concept of Flowlet, numerous works have

been developed. CONGA[2] addresses the limitations of local congestion-aware load balancing in asymmetric network topologies and simplifies the complexity by introducing a leaf-to-leaf mechanism. LetFlow[36] explores the relationship between sub-flows and path load, and introduces a Markov model to demonstrate the high performance of Flowlet in multi-path load balancing. MLAB[11] proposes a modularized solution for multi-path DCN. In essence, MLAB practically implements Flowlet within a Fat-Tree topology. OLTEANU[26] makes use of back-end servers to offload connection states from switches, which has inspired our work to maintain the state on hosts instead of switches. However, in OLTEANU, severs receive a significant number of packets that belong to other connections, and this redundancy hinders performance. Google proposes PLB[32], which is built on the basis of ECMP/WCMP. It detects whether it is experiencing congestion through the TCP protocol with a threshold judgment and selects an available path to redirect the connection by assigning a new flow label for subsequent outbound packets. Although it can mitigate switch link load imbalance and reduce switch packet loss, it still lacks in concurrent competition.

Distributed load balancing mechanisms have the advantage of being able to react quickly to congestion in DCNs. However, one of their challenges is the potential for concurrent competition when blindly selecting the path with the lowest load for distribution. This issue arises because distributed mechanisms often have access to only partial information about the network, which may results in that multiple nodes simultaneously choose the same low-load path, leading to congestion and performance degradation. The above are the problems with existing load balancing work, some existing work[17, 6, 44] on network routing and link failure recovery in DCN, has provided us with assistance in solving the above problems.

3 Background

With substantial improvement of equipment performance, data center has the ability to complete complex tasks, such as data storage[8, 42], distributed machine learning (DML)[20–22, 37, 7] and content distribution[4, 1, 29]. Meanwhile, with the development of host performance, network performance increases as well. However, the development of load balancing, as one of the factors important to network performance, is still marking time. Despite its shortcomings, ECMP has been a convenient and widely used balancing mechanism for a long time. The state-of-the-art work comprehensively performs poorly.

Slow reaction. With the increasing scale of data and device access in DCNs, network congestion has become a significant challenge. Simply increasing the number of devices is not enough to alleviate congestion in large-scale networks. The key to effective load balancing lies in the ability to quickly detect congestion and react accordingly. However, centralized mechanisms used in traditional load balancing approaches rely on periodic information collection from the entire network. This results in delayed response times to congestion events and longer information collection times, especially in rapidly expanding network environ-

ments. Both factors contribute to wasted time and ultimately degrade network performance.

High concurrent network environment. DCN is inundated with concurrent flows. Numerous applications operate simultaneously on the network, resulting in a substantial volume of data transmission at all times. With the prevalence of massive businesses running on DCN, concurrency has become a common occurrence. Surveys about traffic patterns in data center[40, 35, 43] indicate that burst flows and concurrent competition often lead to congestion and serious packet loss. However, concurrent traffic has received relatively limited research attention. The focus on achieving optimal balancing often poses a challenge in effectively addressing concurrent competition. Assume that load balancing mechanism always chooses a minimum load path for each flow and what would happen? Most of the flows would be scheduled into a minority of paths! We call it *concurrent path competition*. The competition can cause even more serious unbalance and congestion. Introducing a centralized scheduler could potentially be helpful, but the slow reaction time of the scheduler can degrade the performance of DCN.

4 Motivation

In this section, we introduce the principle of ECMP and discuss its shortcomings. We also highlight a key insight using an example.

4.1 Why Not ECMP?

ECMP is considered the most adaptable load balancing solution in DCNs, which operates on a hop-by-hop basis and uses flow-based load balancing. When there are multiple links available to reach the same destination address, ECMP employs a specific strategy (such as Hash, Round robin, etc.) to distribute the flow across these paths[15]. It avoids the limitation of using a single link and improves available bandwidth by distributing traffic across multiple links. ECMP switches typically use the five-tuple of each data packet as a hash key and hash it to a random path. Additionally, ECMP does not take into account the size of flows, resulting in a rough flow scheduling based solely on hashing. This can lead to "hash collisions", where long-lived large flows are mapped to the same path, resulting in imbalances and congestion. Moreover, ECMP operates independently on each switch without coordination, meaning that forwarding decisions made in one switch are not influenced by those in other switches. Consequently, ECMP's lack of coordination between switches results in upstream switches not considering the capacity of downstream switches. This can lead to congestion when upstream switches send large flows that exceed the capacity of downstream switches.

Indeed, ECMP has gained popularity in DCN due to its cost-effectiveness and robustness, but it falls short in terms of network performance. To illustrate this, we deploy a 2-layer Leaf-Spine[23] DCN architecture, where the hosts are

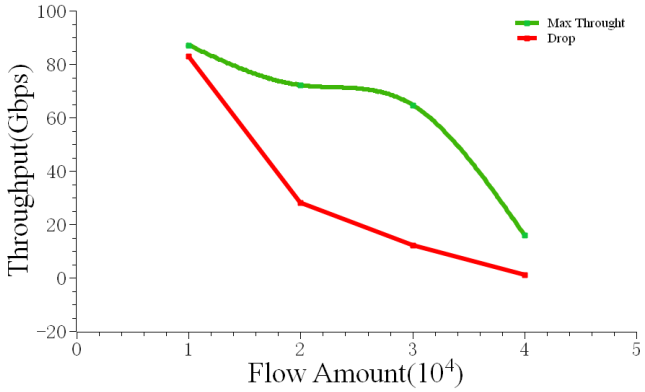
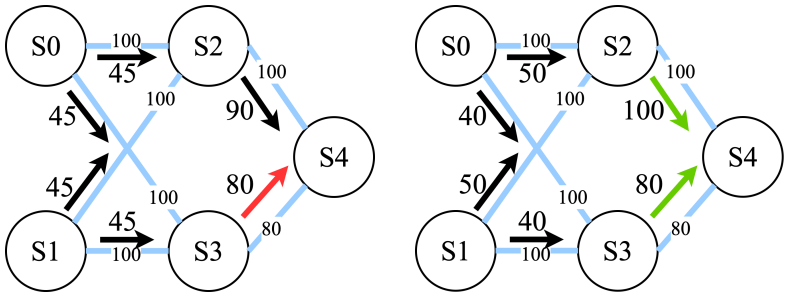


Fig. 1. Throughput "Drop" in DCN.

divided into two sides of the Clos network. Each side of the network sends flows to the other side, allowing us to observe the limitations of ECMP in practice. We evaluate the performance of ECMP in our DCN by measuring the max throughput and the metric named "Drop" of switch ports and is recorded on one of the layer-1 switches. The "Drop" represents the difference between the maximum and minimum throughput, it helps evaluate the distribution of throughput. As shown in Figure 1, as the number of flows increases, the maximum throughput of the network tends to decrease. When the number of flows increases to 40,000, the max throughput decreases down to almost a quarter. The drop in throughput also indicates that the overall network performance is at a low level. The fundamental factor is that ECMP is prone to collisions in a large concurrent environment.

4.2 A Motivation Example



(a) Concurrent Competition (b) Competition Avoidance

Fig. 2. The Path Competition in ECMP.

Figure 2 depicts a DCN example that highlights the issue with ECMP and the key insight of FaCa. Figure 2(a) and 2(b) show a topology with five switches. Each blue line is the link between two switches and the number shows the capacity of the link. Each arrow points to the traffic on the network and the numbers on arrows show the amount of traffic. Switches S0 and S3 simultaneously send traffic to switch S4 at a speed of 90Gbps. In Figure 2(a), switch S0 does not have information about the amount of traffic being sent by switch S1. However, S0 knows that both paths to S4 have sufficient bandwidth to handle its own traffic. With the simple ECMP, the traffic from switches S0 and S3 is evenly split and distributed across the two links adjacent to them.

The initial load distribution appears to be balanced in terms of available bandwidth on both paths. But when the network traffic from S0 and S1 arrive at link $S3 \rightarrow S4$ simultaneously, the traffic will exceed the capacity of the link, and a competition also occurs. To address the potential congestion issue, a solution is proposed where a portion of the traffic originating from S0 and S1 is redirected to the other available path, as depicted in Figure 2(b). Certainly, implementing such a solution requires access to network information in order to make informed decisions about traffic redirection. The details of obtaining and utilizing network information will be discussed in subsequent sections.

5 Design

The section presents the system design of FaCa, which is a modularized system and consists of three parts: network probing, load estimates and balancing decisions.

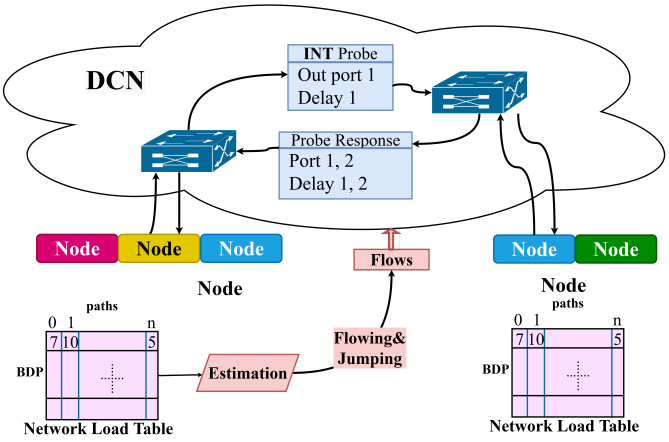


Fig. 3. System Design of FaCa.

5.1 Overview

FaCa implements all the parts on hosts and needs no changes on DCN. To acquire the load information and mapping relationships between paths and five-tuples, hosts send INT probe packets to each other. The information forms a network load table and on this basis, FaCa estimates short-term network load with Exponentially Weighted Moving-Average (EWMA). Derived by estimation, hosts make a decision on flow schedule with the help of Flowing&Jumping.

5.2 Network model

We consider the network topology as $G = (V, E)$, where V indicates the hosts and Layer3 switches, and E indicates edges between them. The link from host/switch i to host/switch j is simply denoted as ij. P_{ij} is the set of paths between host i and host j and P is the union of all different P_{ij}s. A path p_{ij}^k that connects host i and host j is an orderly permutation of links. For instance, host i and host j is connected by switch l, and then we can refer to it as $p_{ij}^k = (i, l, j)$. The bandwidth of the path is denoted as B_{ij}^k. Note that, B_{ij}^k is the minimum bandwidth of links that the path p_{ij}^k traverses. Similarly, we use D_{ij}^k to denote the delay of p_{ij}^k. F_{ij} is denoted as the set of flow demands from host i to host j and f_{ij}^c is the c-th flow demand in F_{ij}, F is the set of all different F_{ij}.

5.3 Partly Global Vision with INT

Probe packets help hosts acquire a global vision of network status information and this probe technique is INT. In the beginning, INT is used only under an in-network way. We draw inspiration from the design concept of INT and successfully implement it at the host end. Firstly, hosts encapsulate the business packets into probe packets and send to other hosts which are the destinations of these packets. When probe packets go through a switch, the switch adds information at the end of the probe packets, including the network status information (such as delay, bandwidth, etc.) and the path that the packet traverses. When a host receives probe packets, it analyzes the information and updates it in the network load table. Meanwhile, the host answers a copy of the probe packets to the sender and parses them. Likewise, when the sender receives a response from a probe packet, it initiates an analysis of the information and updates the network status message table. Besides, INT only focuses on end-to-end path information, and every host builds up a "partly" global vision in this way. That is, each host only has the bandwidth and delay of the path to the destination host, rather than stores the information of the whole network, which reduces the time to wait for the global convergence of the network. As shown in Section 4.2, FaCa manages to be elastic to asymmetry with the global vision.

By obtaining the network status information in near real-time, as described above, we can achieve faster response when congestion occurs in the network. Compared to collecting information at regular intervals, this approach significantly reduces the time required and allows for more efficient decision-making processes.

5.4 Load Estimate

What FaCa gains is the delay and bandwidth of paths but it is far from path load. FaCa introduces bandwidth-delay product (BDP) to describe path load. BDP is formulated as follows:

$$\beta_{ij}^k = B_{ij}^k \cdot D_{ij}^k \tag{1}$$

where β_{ij}^k is the BDP of path p_{ij}^k. BDP measures the in-flight traffic and the remaining maximum bandwidth of a path. Nevertheless, there is a little time interval between updating a probe result and making a schedule decision. The interval makes it inaccurate to describe the realistic path load with BDP in load table. In the field of communication, EWMA is mainly used to estimate and smooth the state parameters of the network. Consequently, FaCa introduces EWMA to fill the gap and the estimation of load is formulated as follows:

$$E_{ij}^k(t) = \alpha E_{ij}^k(t - \Delta t) + (1 - \alpha)\beta_{ij}^k \tag{2}$$

where $E_{ij}^k(t)$ is the estimation of BDP β_{ij}^k at time t and Δt is the time interval between latest record update time and t. The $E(t)$ in the algorithm is the set of all $E_{ij}^k(t)$ where i and j are different. The α is a fitting coefficient to control the proportion of estimation and probe information, which is calculated by the attenuation function model in Newton's cooling law[25]. The specific formula is as follows:

$$\alpha = 1/e^{k*\Delta t} \tag{3}$$

The e and k in the above formula are constants. With BDP and EWMA, FaCa estimates load of paths makes a schedule decision.

5.5 Flowing&Jumping

Load balancing treats flows as running water and flows pour into the light load path until there is no imbalance. In this process, two details are worth thinking.
Path Selection. FaCa is totally software-based and it seems hard to select which path to transport the flow. In traditional network, routers decide routing and it is transparent to hosts. The turning point appears in the layer 3 switches that widely deployed in DCN. Switches schedule flows with a stable hashing algorithm ECMP, and actually, there is a hidden relationship between paths and five-tuple: *every five-tuple maps to a path.* The five-tuple of a probe packet maps to the path it traverses and it is the basis of our load balancing path selection. Just a modification to the source port in five-tuples is able to choose a selected path between sources and destinations, which accomplishes routing decision on host end.
Competition Avoidance. Numerous applications are running large and concurrent communications in DCN, such as database synchronization and DML. Numerous hosts start large flows in the same time and select paths with light load, and competition occurs. The main reason for concurrent competition is

Algorithm 1: Flowing&Jumping

Input: Paths P and Loads $E(t)$, Flow Demands F
Output: Flow Demand Schedule Result

foreach $F_{ij} \in F$ ***in parallel*** **do**
  while $\|F_{ij}\| > 0$ **do**
    randomly a select path p_{ij}^k;
    while $E_{ij}^{k-1}(t) > E_{ij}^k$ *and* $E_{ij}^{k+1}(t) > E_{ij}^k$ **do**
      select a flow from F_{ij}, schedule to p_{ij}^k;
      update E_{ij}^k and F_{ij}^k;
    while $E_{ij}^{k-1}(t) < E_{ij}^k(t)$ **do**
      select a flow from F_{ij}, schedule to p_{ij}^{k-1};
      update $E_{ij}^k(t)$ and F_{ij}^k;
    while $E_{ij}^{k+1}(t) > E_{ij}^k(t)$ **do**
      select flows from F_{ij}, schedule to p_{ij}^{k+1};
      update $E_{ij}^k(t)$ and F_{ij}^k;

that hosts are lack of the load balancing decisions from other hosts, and the decision would never be known before it is taken into practice. And that is the reason why concurrent competition is never considered in DCN load balancing. A centralized controller decides the load balancing schedule uniformly. While in DCN, it is not a perfect solution due to the hysteresis of centralization.

In this case, our solution is to sacrifice the "optimal" balancing in exchange for competition avoidance through adding a little randomness. Algorithm 1 explains the main process of Flowing&Jumping. For flow demands from host i to j, it randomly selects a path p_{ij}^k, and compares it with the two adjacent paths (the "adjacent" indicates continuous numbering). When p_{ij}^k is the lightest load path, flows are scheduled to it. While not, flows would be scheduled to the adjacent paths with lighter load. Either way, the load of paths is not allowed to exceed the adjacent ones. It seems like the water, which schedules flow demands to descend to the sunken place. We add randomness by randomly select the preferable path at every iteration, in case that flows are always scheduled to the lightest load path and the algorithm turns back to an optimal load balancing.

To show the process of Flowing&Jumping more clearly, we illustrate an example in Figure 4. It shows the balancing process from the perspective of a source host, which it has flow demands to send to a destination host. The source host links to eight paths and we number the paths from 0 to 7 with black number. For convenience, we assume that size of every flow demand is one unit and the size of bandwidth of each path is 10 units. At the beginning six flow demands need to be scheduled and we randomly select path 1. Compared to the adjacent paths, load of path 1 is less than path 0 but greater than path 2. Hence, we schedule flows to path 2 until load of path 2 reaches the level of path 1. Again, with remaining flow demands, we randomly select another path 6 and compare

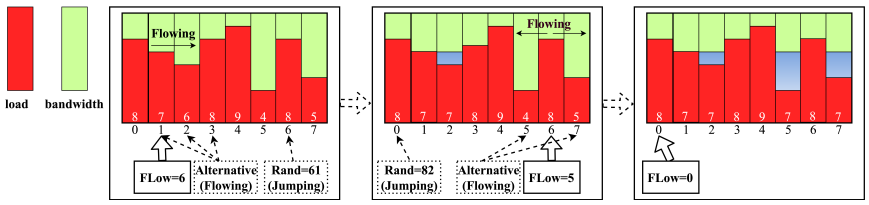


Fig. 4. An example of Flowing&Jumping. Each bar indicates a path and the black number below the bar denotes the rank of path. Above path numbers, red bars are the load of the paths and light yellow bars are the bandwidths. On the bottom of bars, the number with white color denotes the specific value of load. The solid box with flow demands points to the path randomly selected and the two dashed boxes separately points to flowing paths and jumping paths.

with the adjacent paths 5 and 7, while load of path 6 is higher than the that of other two paths. Thus, we schedule flows to path 5 and 7 until their loads reach the level of path 6. At this time flow demands are completely scheduled out, and the algorithm process stops.

Throughout this process, there is a little trick on random selection: *treat it like a ring*. In Figure 4, when selecting the next random path after path 1, there is a random number 61. While the total amount of paths are 8, we select path (61+1) mod 8 = 6 as the next one. The modular calculation makes the path tail to head. Similarly, the adjacent paths of path 7 is (7+1) mod 8 = 0 and (7-1) mod 8 = 6. It balances the situation on head paths and tail paths.

6 Implementation

This section describes the deployment environment of FaCa and the challenges encountered in implementing it.

6.1 Productive Experiment Setting

We utilize a production cluster in DCN operating under Clos topology, as shown in Figure 5. The Clos network comprises three layers, namely two access layers (S0, S3) and a core layer (S1). There are 16 switches in the core layer, and in each access layer, there are 8 switches, with one of them connecting five hosts, and the links between the hosts and S0 are unique. Specifically, each host is equipped with multiple RDMA[41] NICs(RNIC), and each port of the S0 switches is uniquely connected to one RNIC. The connection between the access layer and the core layer is fully connected. We develop a prototype based on perftest, which is a RDMA performance test tool and used to evaluate on this DCN.

Ideally, the Clos network can tolerate the whole traffic from hosts under the same S0 and forward it to the hosts under S3. However, once unbalance happens on the out ports of S0, it is possible to cause a congestion. We design a simple

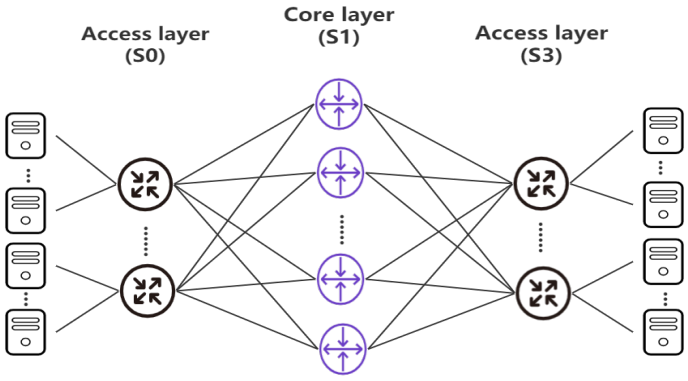


Fig. 5. Production Sub-Network(Clos) Topology.

perftest evaluation, which ten RNICs under one S0 switch send elephant flows to hosts under S3. And details are listed as follows:

Flow Setting. We choose RNICs under one S0 to send and RNICs under S3 to receive. Each RNIC sends two flows and initial sending rate of each flow is 100Gbps. For each flow we distribute enough buffer on both senders and receivers to ensure no congestion are caused by out of buffer.

IP Pair. We limit that one RNIC uniquely connects to another RNIC. For the reason that every RNIC has the unique IP, we call the connection "IP Pair". Such design is able to prevent incast[46] from disturbing experiment results.

Data Collection. Before flows run on the DCN, connection establishment consumes a little bandwidth. Flows last until the end of experiments and therefore, measurement indexes would be stable after establishment. Consequently, we collect experiment data when the load of paths is stable.

6.2 Implementation Challenges

The first challenge is exploration probe. INT only detects the load of switches and paths of probe packets, while information of five-tuples is still unknown. FaCa acquires it through hook function, which is able to probe flow information from system kernel. Note that, although RDMA is kernel-bypass, it still needs TCP or UDP to exchange connection information before establishing it. Therefore, it is easy to hook connection information in establishment with a daemon.

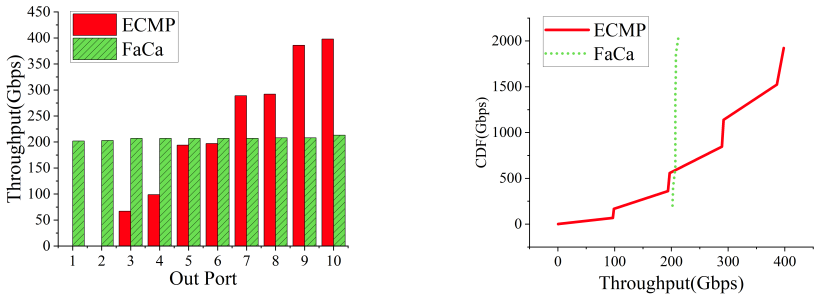
It requires extra work to provide a software-based load balancing that supports for lightweight implementation. Our solution is to develop a communication library that provides Flowing&Jumping on hosts. Based on InfiniBand(IB) Verbs[31], the establishment of RDMA connection covers several steps and the five-tuple information is hidden in Queue Pair(QP) modification. In establishment stage, QP status goes through the following process: RESET $\rightarrow$ INIT $\rightarrow$ Ready To Receive(RTR) $\rightarrow$ Ready to Send(RTS). Between INIT and RTR,

senders and receivers exchange the information of five-tuple. Consequently, FaCa updates the five-tuple between the two stages and manages to control routing on hosts. In this case, whenever business developers intend to implement FaCa, the only overhead is making communication library to establish connections. The final challenge is building Inter-Process Communication(IPC) between probe daemon and Flowing&Jumping. We achieve it by reader-writer model and support an asynchronous communication.

7 Evaluation

This section focuses on the performance of FaCa in productive DCN and micro-benchmark scenarios. First, we compared the performance of ECMP with FaCa in terms of throughput, mainly on S0 switch port and between IP pairs. Then, we compared their congestion in different experiments to verify FaCa's ability to handle concurrent competition. Finally, to better demonstrate our performance, we add comparison with PLB and WCMP. Due to the limitations of implementing PLB on a productive environment, we conducted a micro-benchmark simulation for experiments.

7.1 Throughput Performance



(a) Path Throughput (b) Throughput CDF

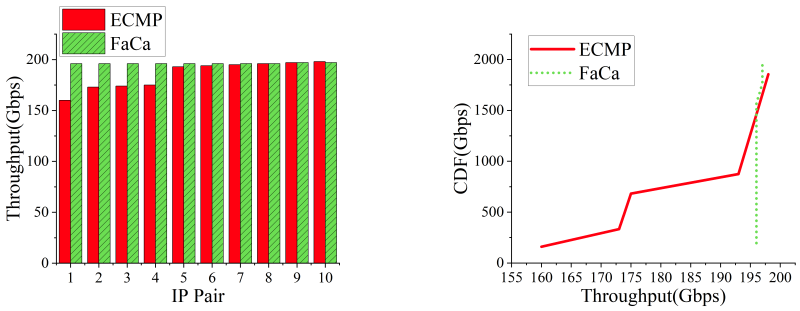
Fig. 6. Throughput of one S0 switch.

As shown in Figure 6, we measure the throughput of ten out ports in one of the S0 switches which receives totally 10×100Gbps flows and forwards to S1 switches. The red bars in Figure 6(a) show that ECMP forwards at least four 100Gbps flows to out port 10, the same to port 9. It leads to an unbalanced on port 0-4, while port 0 sends nearly no traffic. This asymmetric network resources causes unbalance and performance down. The situation in FaCa achieves a better balance. In the coarse-grained elephant flows situation, Flowing&Jumping

schedules two flows to each path. More precisely, Flowing&Jumping schedules the flows to go through different output ports of S0 on average. If we use the standard deviation of throughput to evaluate the balance performance, the value in ECMP is 148.06, while in FaCa, it is 2.96, which represents an improvement of almost 49 times.

As shown in Figure 6(b), to make flow distribution more distinct, we collected the throughput data and plotted it as a Cumulative Distribution Function(CDF). The red curve illustrates that the flow distribution in ECMP is quite dispersed, whereas the green curve depicts a denser flow distribution in FaCa, indicating better load balancing performance. However, due to flow collisions, the total throughput of ECMP is slightly lower than that of FaCa. Note that the experiment is based on elephant flows, and thus in business scenarios, the flow collision problem is likely to be less noticeable when the number of mouse flows is high. Additionally, because FaCa is based on ECMP, it may also achieve slightly higher performance.

During the experiment, we also measured the throughput of IP pairs. To the best of our knowledge, the imbalance problem in switches does not always result in a performance degradation on hosts. This is the reason why improvements in load balancing often do not significantly enhance network performance. As



(a) IP Pair Throughput (b) Throughput CDF

Fig. 7. Throughput of IP Pairs.

shown in Figure 7(a), performance difference on host RNIC is not as distinct as that on S0 out ports. The flow collision and unbalance on ECMP finally lead to a performance downgrade in host, as the red bars show. Compared to the throughput of ECMP, FaCa maintains a load-balanced performance and helps RNICs achieve full bandwidth. Precisely, the throughput "gap" of RNICs in FaCa load balancing is 6Gbps (191Gbps vs 197Gbps), while that in ECMP is 38Gbps (160Gbps vs 198Gbps), which is almost 6.3 times improvement. Meanwhile, the standard deviation of throughput in ECMP is 13.61, while that in FaCa is 1.78, which is a 7.08 times improvement. It is much less compared to the improvement in switches.

Similar to the performance in switches, FaCa achieves a denser CDF curve than that in ECMP, as shown in 7(b). And the total throughput in FaCa is also slightly higher than ECMP. Generally, the average of flow throughput in FaCa outperforms that in ECMP with 22.5%.

7.2 Concurrent Competition

Simultaneously, we test the resilience to concurrent competition and illustrate in Figure 8. Precisely, competition in ECMP is caused by hash collision rather than concurrency. Nevertheless, ECMP is still an ideal baseline because the comparison to ECMP directly reflects the improvement in production DCN for the wide deployment of ECMP.

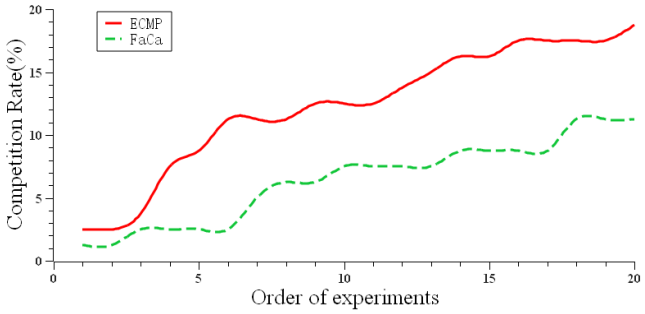


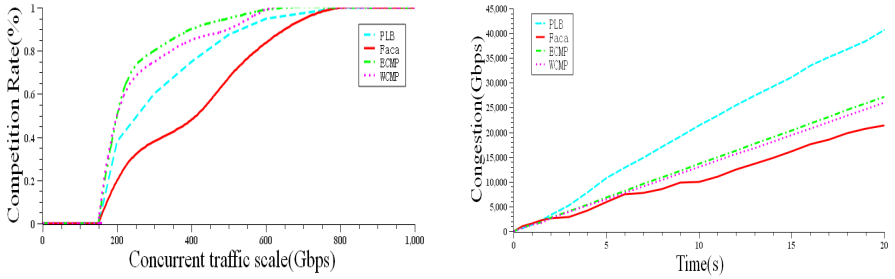
Fig. 8. Path competition in concurrent flows over twenty experiments.

In every experiment, we send persistent flows with random size in hosts and then concurrently send temporary flows. We evaluate path competition resilience by the proportion of the amount of competitive paths. In all congestion scenarios, the congestion rate of Faca is lower than that of ECMP. On average, FaCa outperforms ECMP with 5.88%. Associated with the improvement of throughput, slightly path competition brings serious unbalance.

7.3 Micro-Benchmark

In order to better evaluate the performance of FaCa, we conduct a micro-benchmark environment to compare it with some other methods. Our experiment is simulated using a two-layer Leaf-spine DCN topology. The leaf layer consisted of 16 switches with a capacity of 200Gbps, while the spine layer consisted of 4 switches with a capacity of 380Gbps and 4 switches with a capacity of 400Gbps. The switches in the access layer communicate with each other using a point-to-point communication mode. In each iteration of the experiment, the access switches simultaneously transmit fixed flows to their connected pairs.

In the scenario of continuously sending concurrent streams, we record all the congested traffic on the core switch over a duration of 20 seconds. Figure



(a) Under different flow scale (b) Time

Fig. 9. Congestion on core switches.

9(b) shows the corresponding change trend. FaCa demonstrates significant improvements compared to other methods. Due to the emphasis on balance in ECMP/WCMP, while we focus on randomness, the ECMP-based strategy can exhibit slightly better performance than the latter method under the traffic scale that the link can carry. However, this phenomenon gradually disappears as the traffic increases, and FaCa become more evident, congestion in the network under the FaCa increases at a slower rate compared to other methods. In the scenario where all nodes send traffic evenly, it is possible for ECMP to appear more efficient than PLB. However, although this scenario is not typically encountered in practical network environments, it still proves the superiority of FaCa.

Additionally, we also compare the congestion rates of various methods under different levels of sending traffic. Figure 9(a) displays the congestion rates of each algorithm by increasing the size of concurrent streams. As the scale of concurrent streams continued to increase, ECMP and WCMP quickly experience congestion, followed by PLB. FaCa can fully utilize lower-load links while avoiding conflicts, thereby reducing the occurrence of congestion. Compared to evenly balancing the utilization of all paths, the probability of congestion occurring is much lower. Specifically, FaCa achieves a nearly 50% improvement in congestion rate compared to ECMP, and a 31.47% improvement compared to PLB. These results highlight the effectiveness of FaCa in mitigating network congestion.

8 Conclusion

In this paper, we address the primary challenges associated with implementing load balancing methods on multi-node HPC cluster networks and introduce FaCa as a solution. FaCa is designed for ease of implementation and is built upon a software design approach, leveraging the widely deployed ECMP mechanism as its foundation. By introducing INT probe technology, FaCa establishes a partially global view of the network, reducing the time required for information collection and enabling fast response to network congestion. To address concurrent path competition, FaCa introduces a load balancing algorithm called Flowing&Jumping, which incorporates a small amount of randomness into the flow

scheduling process. Our evaluation demonstrates that FaCa achieves 14.28% reduction in network congestion and outperforms native ECMP by 22.5% in terms of network throughput.

References

1. Abudaqa, A.A., Mahmoud, A., Abu-Amara, M., Sheltami, T.R.: Super generation network coding for peer-to-peer content distribution networks. IEEE Access **8**, 195240–195252 (2020)
2. Alizadeh, M., Edsall, T., Dharmapurikar, S., Vaidyanathan, R., Chu, K., Fingerhut, A., Lam, V.T., Matus, F., Pan, R., Yadav, N., et al.: Conga: Distributed congestion-aware load balancing for datacenters. In: Proceedings of the 2014 ACM conference on SIGCOMM. pp. 503–514 (2014)
3. Benson, T., Anand, A., Akella, A., Zhang, M.: Microte: Fine grained traffic engineering for data centers. In: Proceedings of the seventh conference on emerging networking experiments and technologies. pp. 1–12 (2011)
4. Bilal, K., Khalid, O., Erbad, A., Khan, S.U.: Potentials, trends, and prospects in edge technologies: Fog, cloudlet, mobile edge, and micro data centers. Computer Networks **130**, 94–120 (2018)
5. Clos, C.: A study of non-blocking switching networks. Bell System Technical Journal **32**(2), 406–424 (1953)
6. Cong, P., Zhang, Y., Wang, L., Wang, W., Gong, X., Yang, T., Li, D., Xu, K.: Dit and beyond: Inter-domain routing with intra-domain awareness for iiot. IEEE Internet of Things Journal pp. 1–1 (2023). https://doi.org/10.1109/JIOT.2023.3293500
7. Cong, P., Zhang, Y., Wang, W., Xu, K.: Soho-fl: A fast reconvergent intra-domain routing scheme using federated learning. IEEE Network pp. 1–8 (2023). https://doi.org/10.1109/MNET.132.2200505
8. Corbett, J.C., Dean, J., Epstein, M., Fikes, A., Frost, C., Furman, J.J., Ghemawat, S., Gubarev, A., Heiser, C., Hochschild, P., et al.: Spanner: Google's globally distributed database. ACM Transactions on Computer Systems (TOCS) **31**(3), 1–22 (2013)
9. Cui, W., Qian, C.: Difs: Distributed flow scheduling for adaptive routing in hierarchical data center networks. In: 2014 ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS). pp. 53–64. IEEE (2014)
10. Curtis, A.R., Kim, W., Yalagandula, P.: Mahout: Low-overhead datacenter traffic management using end-host-based elephant detection. In: 2011 Proceedings IEEE INFOCOM. pp. 1629–1637. IEEE (2011)
11. Fan, F., Hu, B., Yeung, K.L.: Routing in black box: Modularized load balancing for multipath data center networks. In: IEEE INFOCOM 2019-IEEE Conference on Computer Communications. pp. 1639–1647. IEEE (2019)
12. Gandhi, R., Liu, H.H., Hu, Y.C., Lu, G., Padhye, J., Yuan, L., Zhang, M.: Duet: Cloud scale load balancing with hardware and software. ACM SIGCOMM Computer Communication Review **44**(4), 27–38 (2014)
13. Ghorbani, S., Godfrey, B., Ganjali, Y., Firoozshahian, A.: Micro load balancing in data centers with drill. In: Proceedings of the 14th ACM Workshop on Hot Topics in Networks. pp. 1–7 (2015)
14. Hopps, C.: Analysis of an equal-cost multi-path algorithm. RFC (2000)

15. Hsu, K.F., Tammana, P., Beckett, R., Chen, A., Rexford, J., Walker, D.: Adaptive weighted traffic splitting in programmable data planes. In: Proceedings of the Symposium on SDN Research. p. 103–109. SOSR '20, Association for Computing Machinery (2020)
16. Hu, J., Huang, J., Lyu, W., Li, W., Li, Z., Jiang, W., Wang, J., He, T.: Adjusting switching granularity of load balancing for heterogeneous datacenter traffic. IEEE/ACM Trans. Netw. p. 2367–2384 (2021)
17. Huang, H., ZhangB, Y., Wang, R., Xiang, Q., Wang, W., Que, X., Xu, K.: Fravar: A fast failure recovery framework for inter-dc network. In: 2023 IEEE Wireless Communications and Networking Conference (WCNC). pp. 1–6 (2023). https://doi.org/10.1109/WCNC55385.2023.10119088
18. Kandula, S., Katabi, D., Sinha, S., Berger, A.: Dynamic load balancing without packet reordering. ACM SIGCOMM Computer Communication Review **37**(2), 51–62 (2007)
19. Katta, N., Hira, M., Kim, C., Sivaraman, A., Rexford, J.: Hula: Scalable load balancing using programmable data planes. In: Proceedings of the Symposium on SDN Research. pp. 1–12 (2016)
20. Kraska, T., Talwalkar, A., Duchi, J.C., Griffith, R., Franklin, M.J., Jordan, M.I.: Mlbase: A distributed machine-learning system. In: Cidr. vol. 1, pp. 2–1 (2013)
21. Li, M., Andersen, D.G., Smola, A.J., Yu, K.: Communication efficient distributed machine learning with the parameter server. Advances in Neural Information Processing Systems **27** (2014)
22. Li, X., Gomena, S., Ballard, L., Li, J., Aryafar, E., Joe-Wong, C.: A community platform for research on pricing and distributed machine learning. In: 2020 IEEE 40th International Conference on Distributed Computing Systems (ICDCS). pp. 1223–1226 (2020)
23. Luo, S., Xing, H., Li, K.: Near-optimal multicast tree construction in leaf-spine data center networks. IEEE Systems Journal pp. 2581–2584 (2020)
24. Miao, R., Zeng, H., Kim, C., Lee, J., Yu, M.: Silkroad: Making stateful layer-4 load balancing fast and cheap using switching asics. In: Proceedings of the Conference of the ACM Special Interest Group on Data Communication. pp. 15–28 (2017)
25. Newton, I.: Mathematical principles of natural philosophy. In: London: Printed for Benjamin Motte (1729)
26. Olteanu, V., Agache, A., Voinescu, A., Raiciu, C.: Stateless datacenter loadbalancing with beamer. In: 15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18). pp. 125–139 (2018)
27. Patarasuk, P., Yuan, X.: Bandwidth optimal all-reduce algorithms for clusters of workstations. Journal of Parallel and Distributed Computing **69**(2), 117–124 (2009)
28. Patel, P., Bansal, D., Yuan, L., Murthy, A., Greenberg, A., Maltz, D.A., Kern, R., Kumar, H., Zikos, M., Wu, H., et al.: Ananta: Cloud scale load balancing. ACM SIGCOMM Computer Communication Review **43**(4), 207–218 (2013)
29. Peng, G.: Cdn: Content distribution network. arXiv preprint cs/0411069 (2004)
30. Perry, J., Ousterhout, A., Balakrishnan, H., Shah, D., Fugal, H.: Fastpass: A centralized" zero-queue" datacenter network. In: Proceedings of the 2014 ACM conference on SIGCOMM. pp. 307–318 (2014)
31. Pfister, G.F.: An introduction to the infiniband architecture. High performance mass storage and parallel I/O **42**(617-632), 102 (2001)
32. Qureshi, M.A., Cheng, Y., Yin, Q., Fu, Q., Kumar, G., Moshref, M., Yan, J., Jacobson, V., Wetherall, D., Kabbani, A.: Plb: congestion signals are simple and effective for network load balancing. In: Proceedings of the ACM SIGCOMM 2022 Conference. pp. 207–218 (2022)

33. Raiciu, C., Barre, S., Pluntke, C., Greenhalgh, A., Wischik, D., Handley, M.: Improving datacenter performance and robustness with multipath tcp. ACM SIGCOMM Computer Communication Review **41**(4), 266–277 (2011)
34. Reinsel, D., Gantz, J., Rydning, J.: Idc white paper: Data age 2025:"the evolution of data to life-critical" (2017)
35. Suryavanshi, M., Yadav, J.: Mitigating tcp incast in data center networks using enhanced application layer technique. International Journal of Information Technology pp. 1–9 (2022)
36. Vanini, E., Pan, R., Alizadeh, M., Taheri, P., Edsall, T.: Let it flow: Resilient asymmetric load balancing with flowlet switching. In: 14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17). pp. 407–420 (2017)
37. Verbraeken, J., Wolting, M., Katzy, J., Kloppenburg, J., Verbelen, T., Rellermeyer, J.S.: A survey on distributed machine learning. Acm computing surveys (csur) **53**(2), 1–33 (2020)
38. Wang, S., Zhang, J., Huang, T., Pan, T., Liu, J., Liu, Y.: Fdalb: Flow distribution aware load balancing for datacenter networks. In: 2016 IEEE/ACM 24th International Symposium on Quality of Service (IWQoS). pp. 1–2. IEEE (2016)
39. Wang, W., Sun, Y., Zheng, K., Kaafar, M.A., Li, D., Li, Z.: Freeway: Adaptively isolating the elephant and mice flows on different transmission paths. In: 2014 IEEE 22nd international conference on network protocols. pp. 362–367. IEEE (2014)
40. Wang, Z., Li, Z., Liu, G., Chen, Y., Wu, Q., Cheng, G.: Examination of wan traffic characteristics in a large-scale data center network. In: Proceedings of the 21st ACM Internet Measurement Conference. p. 1–14. IMC '21, Association for Computing Machinery (2021)
41. Yan, S., Wang, X., Zheng, X., Xia, Y., Liu, D., Deng, W.: Acc: Automatic ecn tuning for high-speed datacenter networks. In: Proceedings of the 2021 ACM SIGCOMM 2021 Conference. p. 384–397. SIGCOMM '21, Association for Computing Machinery (2021)
42. Yang, C.T., Shih, W.C., Huang, C.L., Jiang, F.C., Chu, W.C.C.: On construction of a distributed data storage system in cloud. Computing **98**(1), 93–118 (2016)
43. Zhang, J., Ren, F., Lin, C.: Modeling and understanding tcp incast in data center networks. In: 2011 Proceedings IEEE INFOCOM. pp. 1377–1385. IEEE (2011)
44. Zhang, Y., Zhang, H., Cong, P., Wang, W., Xu, K.: Grandet: Cost-aware traffic scheduling without prior knowledge in sd-wan. In: 2023 IEEE/ACM 31st International Symposium on Quality of Service (IWQoS). pp. 1–10 (2023). https://doi.org/10.1109/IWQoS57198.2023.10188706
45. Zhou, J., Tewari, M., Zhu, M., Kabbani, A., Poutievski, L., Singh, A., Vahdat, A.: Wcmp: Weighted cost multipathing for improved fairness in data centers. In: Proceedings of the Ninth European Conference on Computer Systems. pp. 1–14 (2014)
46. Zhu, Y., Eran, H., Firestone, D., Guo, C., Lipshteyn, M., Liron, Y., Padhye, J., Raindel, S., Yahia, M.H., Zhang, M.: Congestion control for large-scale rdma deployments. ACM SIGCOMM Computer Communication Review **45**(4), 523–536 (2015)