ProactMP: A Proactive Multipath Transport Protocol for Low-Latency Datacenters

Rui Zhuang, *Graduate Student Member, IEEE,* Jiangping Han, *Member, IEEE,*
Kaiping Xue, *Senior Member, IEEE,* Jian Li, *Senior Member, IEEE,* Qibin Sun, *Fellow, IEEE,* Jun Lu

***Abstract*—With the development of datacenter networks (DCNs) towards high bandwidth and low latency, the demands of high-level datacenter applications are heading towards high performance and high reliability, which makes traffic congestion one of the most notable problems in DCNs and brings new challenges to transport protocols. Proactive transport protocols are gaining prevalence due to their ability to provide accurate feedback and precise end-to-end control, while multipath transmission is having a broader application space in the multi-path topology of large-scale DCNs. However, these advanced transport protocols aim to improve their performance by addressing some specific congestion problems, but fail to handle multiple congestion problems caused by incast, high workload and load imbalance. Their performance in terms of flow completion time (FCT), delay, robustness, and balance still has room for further improvement. In this paper, we propose ProactMP, a novel proactive multipath transport protocol for further improvement of datacenter communications. ProactMP utilizes the rich resources of parallel paths in modern DCN and spreads the load across available network paths to improve network efficiency. ProactMP deploys a credit-based bandwidth allocation strategy to achieve low delay and zero packet loss, and overcommits receiver downlinks to ensure high link utilization. We have implemented ProactMP in the Linux system. Our testbed experiments show that ProactMP outperforms the TCP variants, MPTCP variants and a leading proactive transport protocol in FCT, link utilization, fairness and latency.**

***Index Terms*—Datacenter network, proactive transport, multipath transmission, transport protocol, multipath TCP.**

I. Introduction

Datacenter has become an indispensable infrastructure in modern networks. Currently, the scale and link speed of datacenter networks (DCNs) are expanding continuously [1], with Internet traffic experiencing exponential growth. High-performance datacenters contain a large scale of nodes up to 10k∼100k. With the availability of next-generation 100Gbps spine blocks, advanced aggregation blocks can support 51.2Tbps of burst bandwidth while achieving ultra-low latency that reaches microsecond (μs) level [2]. These performance metrics of high bandwidth and low latency, along with the traffic characteristics in DCNs, create new operating conditions for network transport protocols, but also introduce new challenges and some unique problems.

Due to the deployment of high-speed, large-scale and low-latency DCNs, traffic congestion has become one of the most important problems in DCNs, which seriously affects the transmission performance and brings more challenges to congestion control [3]. There are three common congestion problems in DCNs, which are illustrated in Fig. 1.

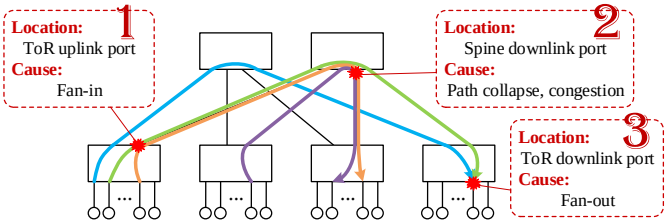


Fig. 1. Three common congestion problems in DCNs and their causes.

The first and the third congestion problems in Fig. 1 occur in Top of Rack (ToR) uplink port and ToR downlink port, respectively, and are mainly caused by traffic incast. Incast happens when multiple senders send "fan-out" requests to many workers, which respond simultaneously with "fan-in" responses, resulting in a drastic decrease in throughput and gross under-utilization of link capacity in many-to-one and many-to-many communication modes [4]. Meanwhile, the shallow switch buffer at the receiver end is more prone to congestion and overflow when handling incast traffic [5], which greatly increases the queuing delay and makes it more challenging to meet the requirement of ultra-low delay [6]. The second congestion problem exists in spine downlink port. Due to the growing need for network bandwidth to handle cloud applications, mixed media downloads and uploads, etc., spine blocks have become the dominant bottleneck that limits the compute power and server capacity of DCNs with Clos topologies [7], insufficient bandwidth or collapse of spine blocks will lead to waste of expensive server capacity and system imbalance [2]. However, imbalances still persist in datacenter fabrics. As observed in [8], some ToR switch ports and fabric switches are busy while their equivalents are relatively idle. Therefore, distributing traffic to different spine blocks to reduce imbalance and avoid congestion is of great significance for improving the effective capacity of network and ensuring user experience.

Numerous proposals have been proposed in recent years to overcome the above problems and obtain better transmission performance in DCNs, including new architectural designs [2], improved TCP variants [6], [9], [10], and various new protocols [11]–[14], among which emerging proactive transport protocols driven by the receiver get increasing

R. Zhuang, J. Han, K. Xue, J. Li, Q. Sun and J. Lu are with the School of Cyber Science and Technology, University of Science and Technology of China, Hefei, Anhui 230027, China.

Corresponding Author: J. Han (e-mail: jphan@ustc.edu.cn), K. Xue (e-mail: kpxue@ustc.edu.cn).

attention from academia due to their unique designs and advantages. In proactive transport, link capacity is allocated proactively to each active sender as credits, either by receivers or a centralized controller, which ensures that senders can send packets at the optimal shared link rate to ensure high bandwidth utilization, low queuing delay, and zero packet loss. Therefore, existing proactive transport protocols are efficient in solving the first and the third congestion problems. However, as existing proactive transport protocols are limited to single-path transmission, their gains in efficiency and reliability are still limited, and cannot effectively solve the second congestion problem. When a certain spine block is congested or suffers path collapse, they cannot seamlessly utilize the bandwidth on the equivalent blocks with good link states, resulting in throughput loss. Therefore, a practical solution that simultaneously takes delay, incast, robustness and balance into account is still in urgent need.

Multipath transmission enables a connection to utilize multiple paths in parallel, which brings benefits in load balancing, resource utilization, and connection resilience. Therefore, multipath transmission has great potential in solving the second congestion problem. As the topology of DCNs heading towards multi-path topologies [15]–[18], multipath capabilities have a broader application space, creating new conditions for further improvement of the transmission performance in DCNs. However, the performance of existing multipath transport protocols, represented by MultiPath TCP (MPTCP) [14], is unsatisfactory in DCNs.

First, the congestion control of MPTCP mainly responds to feedback information, such as network states and variations. However, the delay of feedback can seriously reduce the effectiveness of control strategies in DCNs with low latency and shallow buffer, and easily lead to buffer overflow and increased Round-Trip Time (RTT). Especially when incast occurs, concurrent data from multiple servers can quickly fill up the switch buffer, resulting in heavy loss of packets and timeouts. Therefore, existing multipath transmission does not effectively address the first and third congestion problems. Second, although MPTCP can achieve higher aggregate bandwidth and provide stable transmission for large flows, its gains for small flows to occupy bandwidth and complete quickly are very limited, making it difficult to handle the complex traffic generated by modern datacenter applications.

In conclusion, practical transmission solutions that can effectively address the three common congestion problems remain absent in modern datacenters. From a multipath perspective, the overall performance of proactive transport protocols in DCNs still has room for further improvement, that is, utilizing multipath capabilities to improve bandwidth utilization and reliability. And the multipath transmission in DCNs can also achieve more accurate congestion control by referring to the control methods of proactive transports.

Therefore, in this paper, we combine multipath transmission with proactive transport to effectively solve the three congestion problems shown in Fig. 1. We design ProactMP, a proactive multipath transport protocol for datacenters. ProactMP provides connectionless transmission control to achieve flexible and low-cost multipath management. ProactMP seamlessly utilizes the rich bandwidth resources of the multi-path topology in datacenters, and performs load balancing and congestion migration between multiple paths to improve overall performance. It also employs a receiver-driven credit allocation method to control the amount of in-flight data within an appropriate range, thereby addressing the three common congestion problems in DCN transport. To achieve the above-mentioned benefits, we design the important operations and functions of ProactMP, including connection initialization, subflow establishment, packet scheduling and reassembling, coupled congestion control with overcommitment, and loss recovery, to ensure efficient and correct data transmission through multiple paths. We implement ProactMP in the Linux system, and demonstrate its effectiveness compared to both existing single-path and multipath protocols. The experiments show that ProactMP achieves higher network utilization to accelerate the completion time of different-sized flows. Moreover, it reduces queuing delay and tail RTT to diminish the impact of incast on transmission performance.

The main contributions of this paper are as follows:

- We propose a proactive multipath transport protocol named ProactMP. ProactMP deploys packet scheduling, coupled congestion control and loss recovery to aggregate bandwidth from different paths and support data transmission using parallel subflows.
- We design a receiver-driven control loop for ProactMP to perform proactive congestion control. This loop enables cooperation between the sender and receiver to allocate network bandwidth to reduce queuing delay and buffer overflow, and uses an overcommitment-based credit allocation method to ensure high bandwidth utilization.
- We conduct a comprehensive evaluation of ProactMP in our datacenter testbed. Experimental results show that, compared with typical reactive transport protocols, ProactMP improves the efficiency in completing small flows and large flows by 17%∼175%. It reduces tail RTT by 71%∼77% during incast events and also exhibits better adaptability to complex and ever-changing network traffic in practical applications. Furthermore, ProactMP enhances the ability to utilize bandwidth and balance load by utilizing multipath capabilities.

The rest of this paper is organized as follows: We introduce the limitations of connection-oriented protocols in DCNs and some related works in Section II. The detailed design of ProactMP is proposed in Section III, including the basic architecture, operations and functions of ProactMP. To validate the effectiveness of ProactMP, we test ProactMP and existing protocols in our self-constructed datacenter testbed, and present the results and analysis in Section IV. At last, conclusions and plans for future work are drawn in Section V.

II. Background and Related Work

Transmission Control Protocol (TCP) is one of the most entrenched standards of the last forty years. Although TCP has a long and successful history in the Internet and is the protocol of choice for most datacenter applications, it faces

challenges in modern datacenter networks. TCP and MPTCP introduce overheads at many levels [19], thereby limiting the application-level performance. The datacenter environment contains millions of cores and thousands of machines that interact on microsecond timescales, which brings unprecedented challenges to reactive transport protocols. In this section, we analyze the limitations and problems of reactive transport protocols, represented by TCP, MPTCP, and their enhancements, and then introduce the emerging proactive transport protocols aiming to address these problems.

A. Limitations of TCP and MPTCP in Datacenter Networks

TCP is a connection-oriented protocol, which establishes and uses a connection to ensure proper data transmission. TCP provides reliable stream delivery services and is widely used in Internet communications. However, this reliability comes at the cost of maintaining a long-running state between the client-server pair during the communication, resulting in undesirable overhead for applications in datacenters that could have hundreds or thousands of connections. Similarly, as an extension of TCP, MPTCP faces the same problem as TCP, with even higher space and time overheads due to the necessity of maintaining multiple subflows. As a response to these challenges, researches on connectionless protocols are becoming more practical and valuable. Homa's Linux implementation [19] provides a message-based API and implements Remote Procedure Calls (RPCs) to eliminate the overheads associated with connections.

In addition, the performance of TCP and MPTCP in DCNs is compromised for several reasons. First, TCP and MPTCP's congestion controls (CCs) mainly react to feedback, such as network states and variations. However, in DCNs, the feedback delay can significantly reduce the effectiveness of control strategies, leading to queue overflow and consequent RTT increase. Some proposals adopt cautious adjustment methods to converge slowly and accurately to the appropriate sending rate, but at the cost of increasing the FCTs of large flows and reducing the ability of small flows to compete for buffers [20]. Second, there are a wide variety of flows in DCNs, with different sizes and different requirements. At present, applications that generate small flows predominate in real production datacenters [20]. Although Raiciu *et al.* [21] proved that using MPTCP in datacenters provides better performance and robustness than single-path TCP, such gains are relatively constrained and insufficient to support the complex types of traffic generated by modern datacenter applications. MPTCP ensures stable transmission for large flows by utilizing the aggregate bandwidth and alleviate the uneven distribution of network traffics. But unfortunately, it has limited effect on small flows to occupy bandwidth or complete quickly. This is because MPTCP's first subflow has to perform slow start, and its second subflow takes several RTTs from establishment to data transmission, while small flows get no gain in this process but a longer base delay caused by handshake and convergence.

B. Reactive and Proactive Transport Protocols

The common transport control schemes in DCNs can be divided into two types: sender-driven control schemes based on forward detection, and receiver-driven control schemes based on reverse detection.

Currently, sender-driven control schemes are the mainstream in DCNs. Transport protocols deployed with this kind of scheme are **reactive transport protocols**. TCP, MPTCP and their enhancements (e.g., DCTCP [6], TIMELY [22], HPCC [9], PowerTCP [10] and DCQCN [23]) are typical representatives. Their general method is to first build a model through theoretical analysis, then heuristically adjust the sending rate or congestion window ($cwnd$) at the sender based on the network feedback (e.g., delay, packet loss, queue length, Explicit Congestion Notification (ECN), and variations). For example, DCTCP uses ECN to mark packets after the buffer occupancy of switches exceeds a certain threshold, and adjusts $cwnd$ at the sender according to the proportion of marked packets returned. However, due to the fact that reactive transport protocols can only react after receiving feedback, and feedback typically requires one RTT to reach the sender, reactive transport protocols have limitations in DCNs. In DCNs characterized by high bandwidth and traffic bursts, the delay of one RTT can reduce the effectiveness of feedback, making it difficult for sender-driven control schemes to adjust $cwnd$ based on current network states, and inevitably lead to buffer backlog and congestion, as well as unstable queue length and throughput loss.

In order to achieve low latency, low buffer occupancy, and high network utilization to meet the increasing performance requirements, **proactive transport protocols** that adopt receiver-driven control schemes are booming in recent years. The basic control principle of proactive transport protocols is to explicitly allocate the bandwidth of bottleneck links between active flows to prevent congestion proactively. Typical proactive transport protocols adopt receiver-driven control schemes, which regard bandwidth as credits, allowing receivers to control the sending rate of credits according to the acceptance ability of bottleneck links. Meanwhile, senders can send a specified number of bytes only after receiving credits, thereby preventing packets from accumulating in the network and eliminating congestion proactively. For example, Fastpass [24] implements a packet scheduling strategy with global sense. ExpressPass [5] and pHost [11] explicitly schedule the arrival of data packets from different senders by controlling credits at the receiver. NDP [12] and Homa [13] further improve the above methods by trimming packets and using network priorities, respectively. Bolt [3] utilizes programmable data planes to provide precise congestion signals, and proactively generates granular feedback to reduce the control loop delay to sub-RTT levels. Hostping [25] automatically identifies the traffic source on overloaded links by monitoring and diagnosing intra-host bottlenecks, thereby improving network performance. Protego [26] implements a credit-based admission control strategy that regulates the rate of incoming requests to a server based on marginal improvements in throughput.

III. Design of Proactive Multipath Transport Protocol

As a proactive multipath transport protocol, ProactMP aims to provide higher completion efficiency, higher effective throughput, lower delay, and better robustness for transport protocols in datacenters by utilizing proactive capabilities and multipath capabilities. In this section, we introduce the architectural, operational, and functional designs of ProactMP.

A. ProactMP: An Overview

Operates at the transport layer, ProactMP consists of the proactive transport component and the multipath component, where the proactive transport component performs the functions of endpoint identification and message sending, and the multipath component consists of a set of additional functions over the proactive transport component to manage multiple subflows with proactive transport below it.

To realize connectionless, ProactMP implements the fundamental data transport through RPC, which uses a request-response message-passing mechanism to accomplish client-server interaction. Except for the Request message sent from a client to a server and the returned Response message that each RPC consists of, ProactMP mainly uses three packet types to facilitate the operations of the sender and the receiver. Table I explains the three packet types.

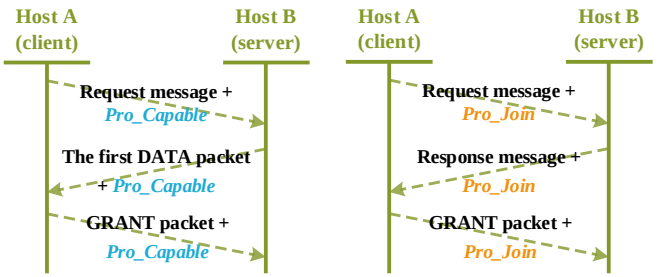
TABLE I
PACKET TYPES USED BY PROACTMP

Packet type	Explanation
GRANT	Sent from receiver (client) to sender (server). Contains the total available credits for the sender.
DATA	Sent from sender to receiver. Contains a certain number of bytes, as well as an offset and a length to indicate the range of bytes within a message.
RESEND	Sent from receiver to sender. Contains the first range of missing bytes within a message, and instructs the sender to retransmit the specified bytes.

B. Important Operations in ProactMP

Connection initialization and subflow establishment are the two important operations in ProactMP. Connection initialization refers to establishing a connection between host pairs to support subsequent data transmission along with multiple paths. Subflow establishment refers to selecting appropriate paths among multiple paths to improve the efficiency and reliability of data transmission.

1) Initiating a ProactMP Connection: Each ProactMP connection initially starts with the first subflow and then adds additional subflows. The *Pro_Capable* option will be carried by the Request message, first DATA packet and returned GRANT packet, so as to verify whether the remote host supports ProactMP and wants to select multipath working mode. The detailed process of initiating a ProactMP connection is illustrated in Fig. 2(a). It should be noted that the meaning of "connection" mentioned in this paper differs from that of TCP or MPTCP. ProactMP's connection is essentially a cluster of identifiers (IDs), which is used to indicate which ProactMP subflows are included in a ProactMP connection. Each subflow has a unique ID and is associated with the message ID to differentiate between different requests. ProactMP only maintains the states of active subflows, and subflows belonging to the same connection are mutually independent except for some coupled controls.



(a) Initiating a ProactMP connection (b) Starting a new subflow

Fig. 2. Detailed process and information of initiating a ProactMP connection and starting a new subflow.

After both of the communicating parties successfully confirm *Pro_Capable*, ProactMP establishes the first subflow and enters ESTABLISHED state. Otherwise, if the remote host does not support multipath transmission or does not want to select multipath working mode, ProactMP returns to single-path proactive transport. It is worth mentioning that the control functions of ProactMP are the same for connections with only one subflow and connections with multiple subflows.

2) Starting a New Subflow: Subsequent subflows can be started and associated with a ProactMP connection after its initial subflow is established. A path manager provides functionalities for establishing, adding, and removing subflows. To manage paths, each ProactMP endpoint maintains a list of IP addresses, consisting of the IP addresses of each interface, and establishes new subflows between different source and destination IP address pairs based on the lists of source and destination hosts. As shown in Fig. 2(b), during the establishment of new subflows, the *Pro_Join* option will be carried by the Request message, Response message, and GRANT packet, which allows the communicating parties to confirm the proper establishment of a new subflow.

When s host recognizes that a new subflow is established, it confirms and stores the ID of this subflow and the ID of the connection that advertises this subflow. ProactMP stores the IDs of subflows, and associates them with connection IDs to ensure that all advertised subflows are knowable and available to the respective connection.

C. Functions Implemented in ProactMP

In order to manage multiple subflows and ensure successful data transmission, ProactMP possesses the functions of packet scheduling, data transmission and reassembling, congestion control, and loss recovery. These functions work synergistically to achieve optimal transmission performance. Packet scheduling and data reassembling are basic to multipath transmission, they ensure that packets can be balanced across multiple paths and reassembled into complete messages in

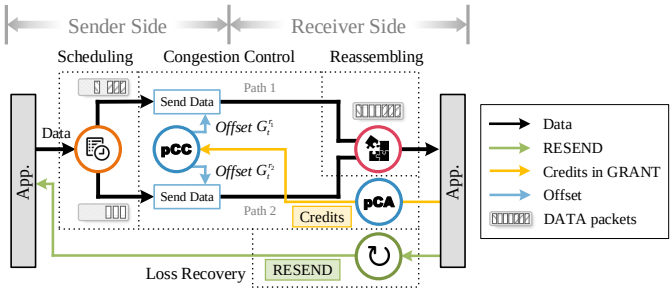


Fig. 3. Transmission process of data packets and the role played by the main functions of ProactMP during this process (take a ProactMP connection with two subflows as an example).

sequence. Congestion control plays a crucial role in avoiding packet loss and network congestion, thereby enhancing both fairness and efficiency. Loss recovery serves as the last remedy to ensure data correctness and integrity. The functional decomposition of ProactMP as well as the transmission process of data packets are illustrated in Fig. 3.

1) Packet Scheduling of ProactMP: Packet scheduling is one of the key methods to improve the throughput of parallel multipath transmission, especially in heterogeneous network environments. The packet scheduler receives data from applications and sends them to each subflow after segmenting and processing. Considering that ProactMP is mainly deployed in datacenter networks, and adopts a credit-based control strategy to effectively balance load and avoid buffer overflow, we expect the quality difference between multiple paths to be very small in ProactMP. Moreover, given that ProactMP is connectionless and does not require ordered delivery of the data, the major problem affecting the performance of parallel multipath transmission, i.e., Head-of-Line (HoL) blocking, can be negligible in ProactMP.

Therefore, we adopt a simplified design in ProactMP's scheduler for high bandwidth utilization and efficiency in the common case. The scheduler of ProactMP first sends data on subflows with the lowest RTT until the window with the size of $OWDbytes$ is filled, and then starts sending data on subflows with the second lowest RTT, and so on. Packet scheduling takes effect only for established subflows. If a ProactMP connection has only one subflow, the scheduler will keep sending data to this subflow until additional subflows are established. The data amount of $OWDbytes$ is calculated as:

$$OWDbytes = OWD_{base} \cdot B_{dl}, \tag{1}$$

where OWD_{base} is the base one-way delay (OWD) of the client-server pair and B_{dl} is the bandwidth of the ToR downlink. $OWDbytes$ is used to ensure that senders transmit enough bytes to cover the OWD between sender and receiver without introducing additional delays.

2) Data Transmission and Reassembling of ProactMP: A sender of ProactMP can transmit all its data through different subflows in parallel, and the data will be sorted and reordered at the receiving end. Each DATA packet contains the information of its offset and length, indicating the range of bytes within this message. Therefore, ProactMP does not require ordered delivery of the data, DATA packets can arrive in any sequence, which effectively avoids the negative impact of disorder caused by multipath transmission. ProactMP performs protocol processing and message reassembly at the receiver. The message data received from each subflow will be uniformly stored in a list and sorted according to the offsets carried in the packets. The receiver's buffer is limited, if the pool of unprocessed messages grows too large, ProactMP will stop receiving data.

3) Coupled Congestion Control of ProactMP: Congestion control in ProactMP is implemented on both the receiver and sender sides. A receiver invites senders to transmit all bytes in the message up to a given offset, and controls the offset according to its acceptance ability. A sender allocates the offset to each subflow based on the feedback from all subflows, and instructs them to send an appropriate number of bytes. Therefore, as shown in Fig. 3, ProactMP congestion control consists of two components: the credit allocation (pCA) component at the receiver, which allocates bandwidth as credits to incoming flows, and the credit control (pCC) component at the sender, which controls the data amount sent by each subflow.

Allocating bandwidth as credits to senders ensures low queuing delay and high link utilization, but accordingly also brings the delay of at least one RTT for calculating and delivering credits. With the rapid growth of DCN link rate, the bandwidth wasted in the first RTT due to waiting is having a greater impact on the completion efficiency of DCN flows. Therefore, data should be sent immediately after the first arrival rather than waiting for credits or grants [5], [24]. So far, some existing proactive transport designs (e.g., Homa [13] and Aeolus [27]) have shown that most flows in DCNs can benefit from sending data immediately after arrive. They propose to accelerate small flows by fully utilizing the spare bandwidth in the first RTT. We adopt this design in ProactMP and divide ProactMP congestion control into two phases: pre-grant phase and granted phase.

① *Pre-grant phase*: For the initially established subflow of each ProactMP connection, the sender skips slow start and enters pre-grant phase directly. It blindly sends $OWDbytes$ of data (unscheduled bytes) right after receiving a Request message in the first RTT. Subsequently, the remaining bytes are defined as scheduled bytes, and are transmitted only in response to explicit GRANTs from the receiver. Such bursts in the first RTT guarantee that bandwidth is not wasted due to waiting for credits, thereby achieving shorter FCTs for small flows. In most cases, the shared pools of buffers provided by modern switches in datacenters are sufficient to hold the burst bytes in the first RTT [19], [28]. And ProactMP limits the amount of data blindly sent in pre-grant phase to a small range, which prevents unscheduled bytes from excessive usage of bandwidth and buffers [19].

② *Granted phase*: In the subsequent RTTs, the sender stays in granted phase and performs coupled congestion control. Except for the first subflow used to initiate a ProactMP connection, other subflows do not burst in the first RTT after startup but wait for authorization before transmitting scheduled packets. ProactMP does not use explicit acknowledgments. It allows the Response message to serve as an acknowledgment

for the request, and the GRANT packet to serve as an acknowledgment for the data sent from the sender. The receiver requests the transmission of scheduled bytes by sending GRANT packets, prompting the sender to release scheduled bytes upon receiving any GRANT.

ProactMP congestion control in granted phase relies on the cooperation between sender and receiver. As depicted in Fig. 3, to effectively support credit control (pCC) at the sender, ProactMP first performs essential credit allocation (pCA) at the receiver. On the one hand, the pCA component at the receiver reasonably allocates credits according to its acceptance ability, aiming to ensure that the bandwidth of ToR downlink is fully utilized with no queue buildup, thus laying the foundation for congestion control at the sender. On the other hand, the primary role of the pCC component at the sender is to allocate the total credits contained in a received GRANT packet to each subflow, so as to achieve parallel transmission and congestion migration between multiple subflows, thereby enhancing the effective throughput and robustness of large flow transmission.

With the above thought, pCA and pCC components are designed to work as follows:

pCA component: Each time the receiver receives data, it allocates the amount of data that can be sent (i.e., the total credits) in the next round according to the current idle bandwidth, and returns this amount of data, represented as $credits_{all}$, to the sender along with the GRANT packet. When allocating the total available credits, the receiver considers connection k as a whole, without distinguishing specific subflows. The calculation method of $credits_{all}$ is given by Algorithm 1, which involves the design of an overcommitment-based credit allocation method in ProactMP. A detailed discussion of this method is provided in Section III-C4.

pCC component: After receiving a GRANT packet, pCC component at the sender undertakes the task of credit allocation between subflows. Suppose connection k consists of n subflows, and transmits data between hosts A and B. Every subflow $i \in k$ has a weight w_i $(i = 1, 2, \ldots, n)$, which is initialized to 0 when a subflow is newly established. The RTT of subflow i is defined as the time interval between the last data sent and the GRANT packet obtained this time. Upon receiving a GRANT packet, the sender becomes aware of the status of each subflow and can identify the specific subflow to which this GRANT belongs. On each received GRANT belonging to subflow i, w_i is increased by 1. The main role of pCC is to allocate the total credits to all subflows of a connection each time the sender receives a GRANT packet. Suppose the sender receives a GRANT packet at time t, which contains the total allocated credits $credits_{all}$. So for any subflow $r \in k$, the credits allocated to subflow r is:

$$G_t^r = credits_{all} \cdot \frac{w_r / {RTT_r}^2}{\sum_{i=1}^{n} w_i / {RTT_i}^2}, \tag{2}$$

where G_t^r indicates the offset that subflow r is allowed to send, and it will send G_t^r bytes of data in the next round. When calculating G_t^r according to Eq. 2, the sender has received at least one GRANT packet, therefore we have $\sum_{i=1}^{n} w_i / {RTT_i}^2 > 0$.

Algorithm 1: Credit Allocation Method based on Overcommitment at the Receiver

```
Upon receiving a Response message / DATA packet from any sender:
begin
    T_next ← the time of receiving the response;
    credits_all ← the allocated credits of the response;
    // update the remaining quantity of credits
    credits_rem += credits_all;
    if credits_rem ≥ 0 then
        // the receiver still has spare credits
        credits_all = credits_rem + f(T_next, T_last);
        credits_rem -= credits_all;
    else if 0 > credits_rem ≥ −OWDbytes then
        // there is no available credit, but has free
          bandwidth for overcommitment
        credits_all = f(T_next, T_last);
        credits_rem -= credits_all;
    else
        // stop overcommitting
        Withholding credits until
          credits_rem ≥ −OWDbytes;
    T_last = T_next;
    return credits_all
```

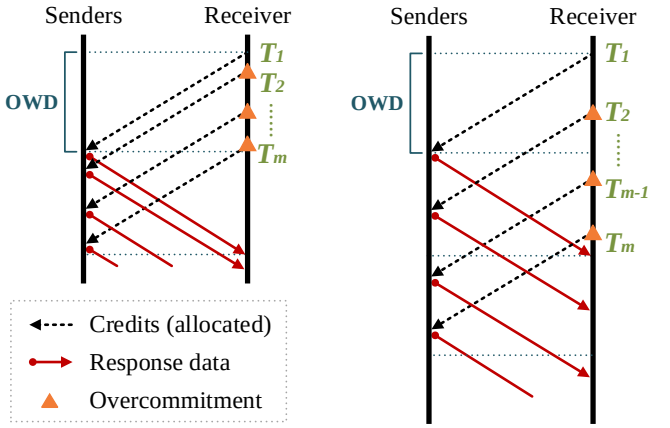
4) Overcommitment of ProactMP: A receiver of ProactMP guides active flows to transmit the appropriate amount of data by allocating credits, and can stop the transmission of a flow by withholding credits. Suppose we keep all incoming flows active at all times, just like TCP and most other existing protocols. In that case, ProactMP is likely to suffer from higher buffer occupancy, as well as higher latency from the credit allocation that loops frequently between flows, both of which contribute to high tail latency.

In some proactive transport designs (e.g., pHost [11]), each receiver can only allow one active flow at a time. But simulation results [13] show that the network utilization would be low under high load if only one active flow is allowed. In addition, due to the delay of credits transmission, a receiver is unable to authorize other incoming flows while waiting for the response from a particular sender. If the sender does not respond to the authorized credits or does not respond immediately, the ToR downlink has to remain idle even if it can be used by other senders.

Therefore, considering the fact that a receiver has no idea whether one particular sender will respond to credits, and suffers from low bandwidth utilization due to waiting for responses from the sender, we use an overcommitment-based credit allocation method in ProactMP to ensure full utilization of the ToR downlink, whose pseudo-code is given by Algorithm 1.

The receiver conducts credit allocation upon receiving a Response message / DATA packet from any sender. We define a function $f(T_b, T_a)$ to calculate the idle credits generated in the process of waiting for responses. Assume that the receiver

This article has been accepted for publication in IEEE Transactions on Network and Service Management. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TNSM.2024.3399028



(a) Overcommitment within one OWD (b) Overcommitment out of one OWD

Fig. 4. Allocations of credits at a receiver with our overcommitment-based method given by Algorithm 1.

allocates credits at time T_a and T_b respectively, so the idle credits generated from T_a to T_b can be calculated as:

$$f(T_b, T_a) = \frac{\min\{T_b - T_a, OWD_{base}\}}{OWD_{base}} \cdot OWDbytes. \quad (3)$$

The receiver needs to overcommit in moderation, otherwise, excessive overcommitment will consume too much buffer space in the ToR and result in congestion. We evaluate the availability of credits based on both the remaining quantity and allocated quantity of credits. The availability of credits determines how many credits a receiver can currently advance to an incoming active flow. Algorithm 1 specifies three different credit availability levels:

① The receiver still has spare credits (line 7∼10). It can use all of the remaining credits and the idle credits generated in the process of waiting for responses after the last allocation.

② There are no spare credits (line 11∼14). At this point, all the available credits reserved by the receiver have been allocated, while free bandwidth is still available on the ToR downlink. Therefore, the idle credits generated in the process of waiting for responses can still be overcommitted to improve the utilization of ToR downlink.

③ Overcommitting more credits to senders might result in queue buildup (line 15∼17). New flows can only be authorized to send data once any sender successfully responds to the previously allocated credits.

Theoretically, the overcommitment-based credit allocation method provided by Algorithm 1 can achieve a maximum of 100% utilization of ToR downlink in each RTT.

Assume that within one one-way delay (OWD), a receiver allocates credits to multiple senders at time T_1, T_2, $\cdots$, $T_m (m \geq 2)$ respectively, i.e., $T_1 - T_m < 1 \cdot OWD$. As shown in Fig. 4(a), overcommitment is involved in all the allocations except for time T_1. So the allocated credits $credits_{all}$ and remaining credits $credits_{rem}$ at different time points can be represented as:

$$\begin{aligned}
&T_1 \begin{cases} credits_{all} = OWDbytes = C_1, \\ credits_{rem} = 0, \end{cases} \\
&T_2 \begin{cases} credits_{all} = f(T_2, T_1) = C_2, \\ credits_{rem} = -f(T_2, T_1), \end{cases} \\
&\cdots \\
&T_m \begin{cases} credits_{all} = f(T_m, T_{m-1}) = C_m, \\ credits_{rem} = -\sum_{i=2}^{m} f(T_i, T_{i-1}). \end{cases}
\end{aligned} \quad (4)$$

Then the total quantity of credits C_{total} allocated in this OWD is:

$$\begin{aligned}
C_{total} &= \sum_{i=1}^{m} C_i \\
&= OWDbytes + \frac{T_m - T_1}{OWD_{base}} \cdot OWDbytes. \quad (5)
\end{aligned}$$

So the maximum downlink utilization U we can achieve is:

$$U_{\max} = \frac{\lim\limits_{T_m - T_1 \to OWD_{base}} C_{total}}{RTTbytes} = \frac{2 \cdot OWDbytes}{RTTbytes} = 100\%, \quad (6)$$

where $RTTbytes$ is the amount of data that can exactly cover the RTT between sender and receiver without introducing additional delays, and there is $RTTbytes = 2 \cdot OWDbytes$.

Therefore, the downlink utilization approaches 100% as $T_m - T_1$ approaches OWD_{base}. Even if certain senders do not respond to credits, the ToR downlink still remains available for other senders to use.

Similarly, assume the time interval between T_1 and T_m is greater than one OWD but less than one RTT. As shown in Fig. 4(b), there is $1 \cdot OWD < T_1 - T_m < 1 \cdot RTT$. According to Algorithm 1, the receiver overcommits credits at time T_2, $\cdots$, T_{m-1}, and T_m to maintain full utilization of the downlink. So we still have:

$$\begin{aligned}
C_{total} &= C_1 + \frac{\min\{(T_m - T_1), OWD_{base}\}}{OWD_{base}} \cdot OWDbytes \\
&= 2 \cdot OWDbytes = RTTbytes, \quad (7)
\end{aligned}$$

and the downlink utilization U is:

$$U = RTTbytes / RTTbytes = 100\%, \quad (8)$$

In summary, ProactMP with overcommitment can achieve up to 100% downlink utilization without exceeding the downlink's capacity, which is a significant improvement over the utilization of no overcommitment. And such improvement brought by overcommitment remains unaffected by the number of active flows or the timing of receiving responses from senders. Even in scenarios with only one active ProactMP connection, overcommitment can still be self-triggered by multiple subflows contained in this connection.

It is worth mentioning that we use $RTT/2$ to calculate the OWD in this paper. In practice, the delay of the forward path and the delay of the return path are not always equal, which leads to the inability to accurately estimate the OWD using $RTT/2$. But ProactMP's multipath capabilities and function of overcommitment protect it from the impact of inaccurate OWD estimates: Even if the delay of the forward path and the return path are not equal, ProactMP ultimately ensures the allocation of $2 \cdot OWDbytes = RTTbytes$ credits to the sender, thus fully utilizing the downlink.

5) Loss Recovery of ProactMP: We expect lost packets to be rare in ProactMP. This is because the two main causes of packet loss, namely link failure and buffer overflow, are very uncommon in datacenter networks with ProactMP. Firstly, modern datacenter networks adopt multiple methods to deal with network faults, which can almost completely mask the effect of packet loss caused by link failure or corruption for transport layer protocols [29]. Secondly, ProactMP adopts a proactive control method to reduce buffer usage and effectively avoid buffer overflows. Therefore, we optimize lost-packet handling in ProactMP to improve its efficiency in common cases of no packet loss.

ProactMP delegates the task of detecting and notifying lost packets to the receiver. ProactMP does not use any explicit acknowledgment for lost packets, but uses a timer-based mechanism to detect lost packets periodically. According to the offset and length in each received DATA packet, the current range of received bytes and missing bytes can be obtained from the list. If the receiver does not receive any packet of a message within a certain period of time (T_{out}, set to several RTTs), it will send a RESEND packet to the sender, which contains the first range of missing bytes in the list. And the sender will retransmit the specified bytes.

The loss recovery of ProactMP also addresses the loss of GRANT packets. When a GRANT packet is lost during transmission, the sender will not send any data due to lack of credits, resulting in the receiver receiving no additional packet of this message and continuing waiting. When the waiting time exceeds T_{out}, it triggers ProactMP's loss recovery and sends a RESEND packet to the sender. The sender can continue to transmit data after receiving the RESEND packet.

We also simplify the response of ProactMP congestion control to packet loss. ProactMP congestion control has no need to identify the occurrence of packet loss or give feedback to lost packets; consequently, packet loss does not affect credit allocation at the receiver and the sender.

IV. Performance Evaluation

We implement ProactMP as a module in the Linux system, and deploy it in our testbed for datacenters. The ProactMP implementation utilizes the RPC framework provided by Dubbo [30] and uses the key networking features in the Linux kernel (e.g., TCP/IP).

A. Test Setup

The topology of our testbed is depicted in Fig. 5. Our testbed adopts the classic Clos [7] (e.g., Fat-Tree [15]) DCN topology, which can implement scenarios such as single-path transmission, multipath transmission, shared bottleneck and numerous traffic concurrency. We add ProactMP traffic and the traffic of contrastive schemes in the left half of Fig. 5, and add background traffic in the right half. We have 3 server hosts (S1, S2, S3) connected to 2 client hosts (C1, C2) through a core switch using 6Gbps links. Each host is a GIGABYTE GB-BSi7HA-6500 machine, with 2.5GHz 64-bit 2-Core Intel Core i7-6500U processor, 4GT/s, 4MB cache, 32GB 2133MT/s DDR4 RAM, and a Intel i219LM 1GbE NIC.

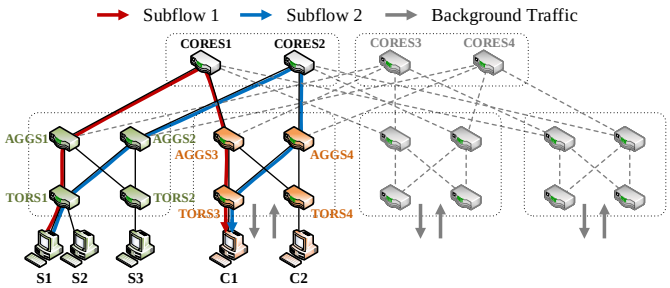


Fig. 5. Topology of our self-constructed testbed for datacenter networks.

The base RTT of our testbed is 2µs, which the minimum RTT between two hosts (e.g., S1 and C1) passing through a core switch. So in the ProactMP implementation for 1Gbps ToR downlink bandwidth, $OWDbytes$ is set to 125B.

Various transport protocols and congestion control algorithms are deployed in the hosts. We choose TCP variants (DCTCP [6], Cubic [31], Vegas [32], BBR [33]), MPTCP variants (Olia [34], Balia [35], wVegas [36]) and a proactive transport protocol (Homa [19]) as the contrastive schemes. All the above schemes have Linux implementations, and are deployed in each host. All the hosts are running on Linux ubuntu 18.04 OS with kernel version 4.19.234 [37]. The path manager of MPTCP is set to "ndiffports", and the scheduler is set to "min-RTT", the detailed description of configuring MPTCP can be seen in [38].

B. Transmission Performance for Different Flow Sizes

DCN is filled with flows of various sizes. We analyze five workloads, including the Web search workload for DCTCP analysis [6] (W1), the workload of the Google search application (W2), the aggregate workload of all applications in a Google datacenter (W3) from literature [13], the workload on a collection of memcached servers at Facebook [39] (W4), and the workload on the Hadoop cluster at Facebook [40] (W5). Large flows dominate W1, flows larger than 1MB account for 30% and the maximum size reaches more than 50MB. In W2, W3, and W5, flows with sizes ranging from 100B to 1KB account for about 60%, while 10% of the flows in W5 are larger than 100KB. In W4, nearly 40% of the flows are smaller than 10B, and more than 95% are smaller than 1KB. Therefore, we choose flows with sizes of 20B, 200B, 1KB, 50KB, 500KB, 10MB, 50MB, and 500MB to verify the performance of different schemes when transmitting small flows and large flows. With no background traffic, we have C1 request a flow of a given size from S1, which then transmits the flow separately with the support of different schemes. All of the multipath schemes utilize two subflows, which pass through spine switches CORES1 and CORES2, respectively, while the single-path schemes only use one path that passes through CORES1. The distribution of flow completion times (FCTs) is shown in Fig. 6, which is the time duration from the request being sent to the message being completed. We can see that ProactMP has the lowest or close to the lowest FCT no matter transmitting what size of flows, its FCT is also

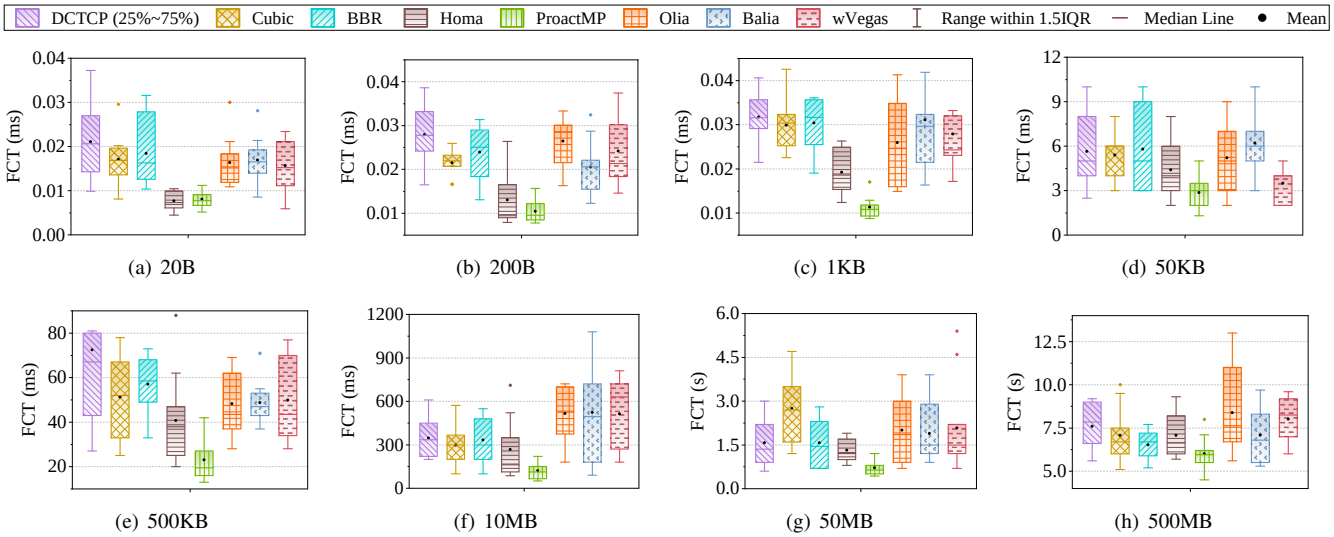


Fig. 6. Distribution of FCTs for flows of different sizes.

more stable than the other schemes and rarely experiences large fluctuations.

When the flow size is 20B, the FCTs of ProactMP and Homa have similar distribution patterns, and this is because they both adopt burst strategy in the first RTT, and 20B size of data is enough to be transmitted within $OWDbytes$ in the first RTT. As the flow size increases to 200B, ProactMP and Homa need more than one RTT to complete the transmission. Although ProactMP has a lower bound of FCT similar to Homa, it can generally guarantee a lower and more stable FCT than Homa. Compared with TCP variants and MPTCP variants, ProactMP significantly improves the FCT of small flows because its bursting in the pre-grant phase can make good use of the idle bandwidth in the first RTT.

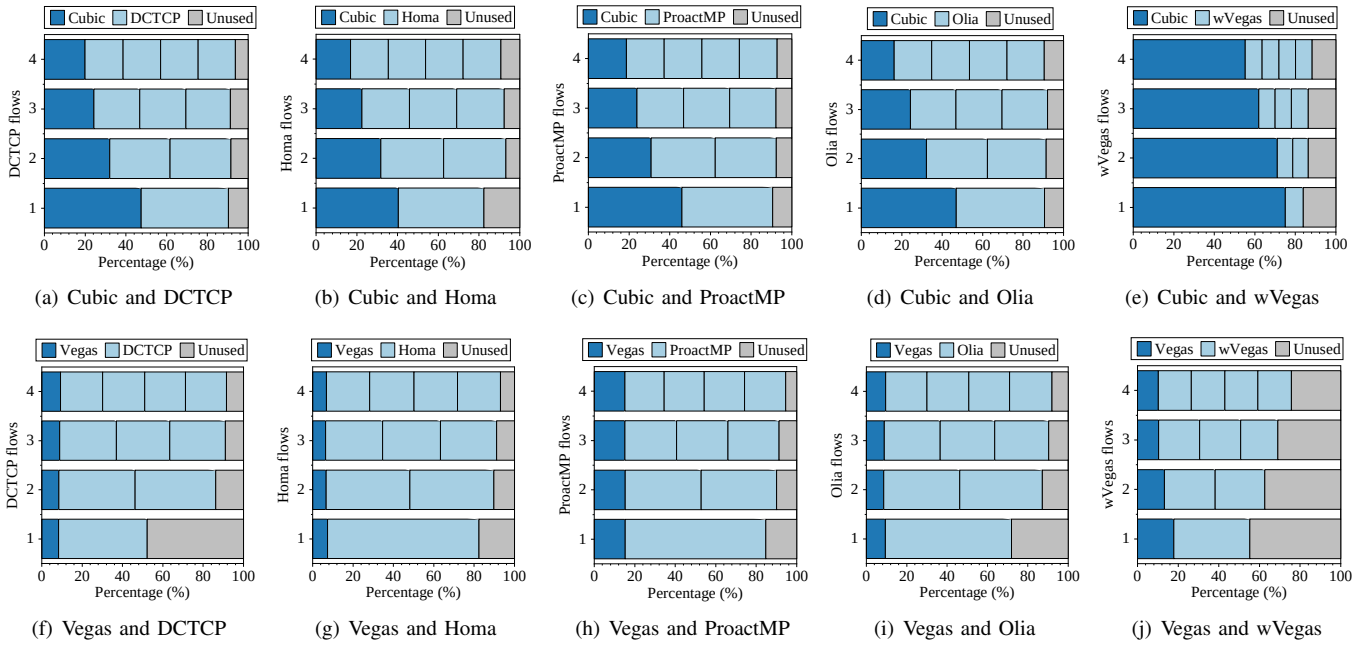
When transmitting large flows, the contrast between the FCTs of Homa and ProactMP becomes more pronounced. As the flow size increases to more than 500KB, Homa's performance gradually approaches that of TCP variants, and does not demonstrate obvious superiority in transmitting large flows larger than 10MB, while ProactMP achieves shorter FCTs. The different overcommitment methods adopted by Homa and ProactMP lead to their performance gap in practical use. Specifically, Homa deploys a priority-based overcommitment mechanism that overcommits at most one flow for each available priority level. This means that Homa's overcommitment does not take effect when there is only one Homa flow in the network or all flows are given the same priority. Therefore, Homa does not achieve notable performance when there is no competition. Instead, ProactMP's overcommitment mechanism calculates its credits based on the current available capacity, which is free from the restriction of the number of priorities or flows, thus achieving higher bandwidth utilization and shorter FCTs in non-competitive situations. MPTCP variants are limited by the design concept of ensuring fairness [41], which makes them obtain similar FCTs to TCP variants despite utilizing multiple subflows, and inferior to DCTCP and BBR in some cases.

Overall, compared with TCP variants, ProactMP improves the efficiency in completing small flows by 133%∼135% and large flows by 17%∼175%. And compared with MPTCP variants, ProactMP improves the efficiency in completing small flows by 102%∼128% and large flows by 30%∼154%. Therefore, it is conducive to supporting the transmission of complex data traffic in DCNs.

C. Bandwidth Utilization and Fairness

Modern datacenters support a wide range of protocols, making it necessary to consider whether ProactMP can coexist gracefully with existing protocols. The primary reason existing protocols are difficult to coexist gracefully and fairly is that they inevitably interact with each other via queuing in the network [42]. As TCP is a dominant transport protocol in production DCNs, we take TCP as an example. TCP variants include loss-based TCP (e.g., Cubic) and delay-based TCP (e.g., Vegas). Loss-based TCP adopts an aggressive window adjustment strategy, which continuously increases its $cwnd$ until packet loss occurs, so it occupies the queue quickly and squeezes the throughput of other protocols. Delay-based TCP, on the other hand, adjusts its $cwnd$ differently. Delay-based TCP is sensitive to network conditions, once the bottleneck queue is built and RTT increases accordingly, it decreases its $cwnd$. As a result, delay-based TCP has limited competitiveness, and its throughput can be easily exhausted due to excessive concession.

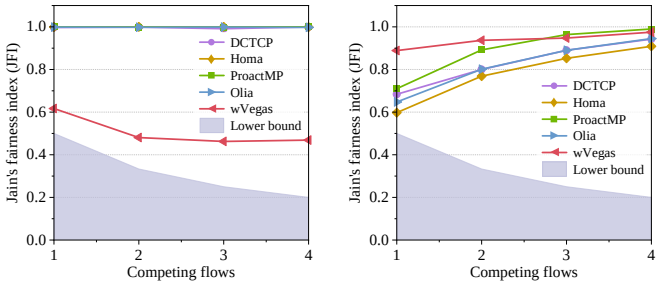
In order to provide a demonstrable effect for comparison, we choose an extremely aggressive scheme (Cubic) and an extremely conservative scheme (Vegas) as the background traffic, respectively. We use iPerf [43] to generate traffic, and test the throughput of different protocols at increasingly higher network loads to measure their ability to utilize network bandwidth. The results of network sharing between different protocols and network utilization limits are shown in Fig. 7. We let a Cubic flow or a Vegas flow go through the same bottleneck link of 1Gbps with a certain number of competing flows, and vary the number of competing flows from 1 to 4.



(a) Cubic and DCTCP (b) Cubic and Homa (c) Cubic and ProactMP (d) Cubic and Olia (e) Cubic and wVegas

(f) Vegas and DCTCP (g) Vegas and Homa (h) Vegas and ProactMP (i) Vegas and Olia (j) Vegas and wVegas

Fig. 7. Network sharing of single-path schemes and multipath schemes with Cubic and Vegas.



(a) Cubic and competing flows (b) Vegas and competing flows

Fig. 8. JFIs of Cubic/Vegas with a certain number of competing flows.

The colored rectangles in each figure represent the percentage of bandwidth occupied by different flows, while the gray rectangles represent the percentage of unused bandwidth. To evaluate the fairness of resource allocation among traffic flows, we also calculate the Jain's fairness index (JFI) [44] of different cases. The JFI curves along with the number of competing flows are shown in Fig. 8.

When competing with an aggressive Cubic flow, DCTCP, Homa, ProactMP and Olia all obtain JFIs close to 1 and get their fair share of bandwidth while achieving high link utilization of over 90%. But no protocol achieves 100% utilization, they all waste network bandwidth to some extent in certain cases. Homa wastes the most bandwidth when there is only one competing flow, and its unused bandwidth gradually increases as the number of flows increases. This is because Homa uses priority levels for flow control, and when all of the scheduled priority levels are allocated, senders will not be able to respond, resulting in the idleness of the receiver's downlink. Such waste of bandwidth becomes increasingly serious as the overall network load increases. But ProactMP's overcommitment mechanism overcomes this problem, and achieves the highest bandwidth utilization while ensuring fairness. By contrast, as a delay-based MPTCP CC scheme, wVegas is far less competitive than the others. wVegas obtains JFIs close to the lower bound, which means the bandwidth resources are almost monopolized by the Cubic flow. Even as the number of competing flows increases, wVegas's bandwidth share and link utilization still show no significant improvement.

When competing with a conservative Vegas flow, all protocols show strong aggressiveness. Even wVegas, a delay-based scheme like Vegas, achieves twice the throughput as Vegas. As the number of competing flows increases, DCTCP, Homa, Olia, and wVegas achieve higher network utilization, but always fail to share bandwidth fairly with Vegas. We can see from Fig. 7(j) that both Vegas and wVegas have poor capacity to utilize bandwidth. Even in the case of multiple active flows, they are still unable to overcommit their ToR downlinks. This observation also indicates that the improvement in bandwidth utilization in other test groups has nothing to do with Vegas. Overall, Vegas's conservative window adjustment strategy makes it challenging for ProactMP and other protocols to share bandwidth with Vegas with perfect fairness. However, as shown in Fig. 8(b), with the increase of flow number of ProactMP, ProactMP's JFI continues to increase to close to 1 and outperforms the fairness provided by delay-based wVegas. Although the JFI of ProactMP is lower than that of wVegas when the number of competing flows is less than 3, this is mainly due to Vegas's poor capacity to utilize bandwidth. ProactMP can achieve high bandwidth utilization at the downlink without compressing Vegas through a reasonable credits control strategy at the receiver, and gradually achieves a fair share of the bandwidth with conservative flows.

This article has been accepted for publication in IEEE Transactions on Network and Service Management. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TNSM.2024.3399028

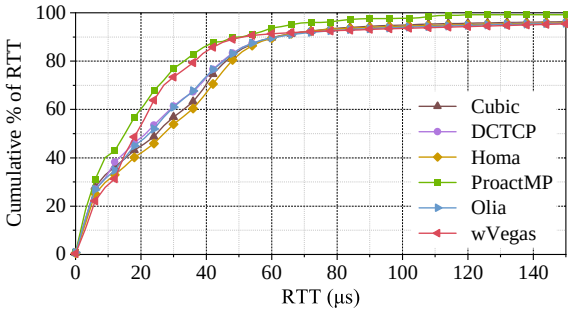
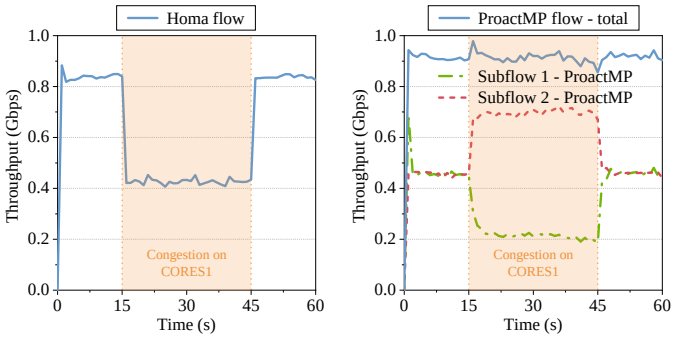


Fig. 9. Cumulative distribution of RTT of a 10-to-1 incast.



(a) Homa (b) ProactMP

Fig. 10. Throughput curves of a Homa flow and two ProactMP subflows when congestion occurs on CORES1.

D. Delay Performance during Incast Conditions

Incast is a common problem in DCNs, which is usually caused by the many-to-one and many-to-many communication mode. When a host sends a request to a group of nodes (e.g., server cluster or storage cluster), all the nodes in this cluster will receive the request and respond almost simultaneously. As a result, multiple nodes will send data streams to this host at the same time, leading to a "micro burst of flows". The incast problem will be particularly more apparent when using TCP in an environment of high bandwidth and low latency [45]. To study the impact of incast on different protocols, we conduct 10-to-1 incast experiments in our testbed. We set host C2 as one client node and let it send 10 requests simultaneously to different servers. Finally, all respond flows will be concentrated in the downlink of TORS4, whose bandwidth is 1Gbps.

TABLE II
MEAN, MAXIMUM AND TAIL RTT OF EACH PROTOCOL, AND THE AVERAGE THROUGHPUT OF TEN FLOWS

Protocol	100B latency (μs)	Mean (μs)	Max (μs)	$99th$-p RTT (μs)	Average throughput (Mbps)
Cubic	3.54	89.61	141060	459	83.71
DCTCP	3.71	126.56	197714	516	86.77
Homa	2.28	101.87	197300	490	80.24
Olia	3.70	132.33	150923	492	83.66
wVegas	3.69	55.21	163912	408	75.13
ProactMP	2.09	48.16	136438	119	93.85

We count the RTT distribution of 6 different protocols in 10 seconds during incast, and display the cumulative distribution of RTT in Fig. 9. Table II shows the basic performance of 6 protocols, where 100B latency is measured end-to-end at the application level with 100-byte requests and responses, $99th$-p RTT is the 99th percentile of RTT. The RTT of ProactMP for short flows (i.e., 100B latency) is 8% lower than that of Homa, and more than 40% lower than those of TCP or MPTCP.

We believe that a protocol with longer RTTs has experienced longer queuing delay, and thus has a weaker ability to handle incast. ProactMP has the shortest mean RTT, about 80% of its RTTs are lower than 33μs, with more than 97% of its RTTs lower than 100μs, and its proportion of RTTs within 10μs is the highest among all protocols. Olia behaves similarly to DCTCP. The RTT of delay-based wVegas has the most centralized distribution pattern. We can see from Fig. 9 that RTTs within the range of 10∼30μs account for about 50% of the total. However, as a reactive transport protocol, the window adjustment strategy of wVegas inevitably has a certain lag and cannot send the optimal amount of data according to the usage of ToR downlink. Therefore, wVegas is still impossible to avoid buffer overflow and timeout caused occasionally by excessive data transmission, resulting in ultra-long RTTs accounting for 4.5%. Similar problems are observed in other reactive protocols driven by the sender, such as Cubic, DCTCP, and Olia. However, ProactMP reduces the proportion of ultra-long RTT to about 2.5% and improves tail RTT by 71%∼77% compared to previous sender-driven protocols. ProactMP also handles incast better than Homa. It improves tail RTT by 76% and mean RTT by 53% compared to Homa. Although Homa is also a receiver-driven protocol like ProactMP, its flow control strategy based on network priorities is less effective against incast. This is because when all of the scheduled priority levels are allocated, the receiver of Homa will be unable to overcommit even if its downlink has idle bandwidth, while most senders experience longer latency due to waiting for responses. Therefore, the performance of Homa tends to be inferior to Cubic overall.

E. Load Balancing and Congestion Migration

ProactMP aims to provide better transmission performance for proactive transport protocols by utilizing multipath capabilities. To verify ProactMP's ability to balance load and migrate congestion, we change the load on a spine block to simulate non-congestion and congestion scenarios, and compare the performance of proactive transport with and without multipath capabilities. We maintain a 60-second transmission between S1 and C1. ProactMP utilizes two subflows, which pass through spine switches CORES1 and CORES2 respectively. While Homa, a single-path scheme for comparison, uses only one path that passes through CORES1. During the transmission, we add 50% network load on CORES1 within 15s $\sim$ 45s and then remove it after 45s, while the load on CORES2 remains unchanged in 60 seconds; so as to simulate the scenario where a certain spine block is congested and becomes the bottleneck that limits the overall

link speed. The throughput curves along with time of one Homa flow and two ProactMP subflows are depicted in Fig. 10.

When there is no congestion on CORES1 and CORES2, the spine block is not the bottleneck to limit the link speed. As shown in Fig. 10, controlled by the receiver (C1), Homa achieves an average throughput of 0.8387Gbps, while ProactMP fairly distributes the load between two subflows and achieves an average total throughput of 0.9235Gbps. Both Homa and ProactMP achieve high downlink utilization.

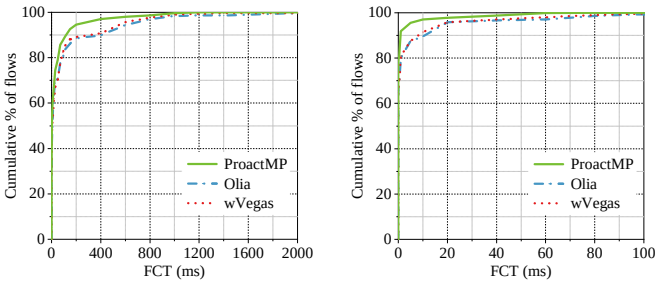
However, as the load increases and congestion occurs on CORES1, the throughput of Homa is obviously limited by the spine switch. Although the downlink of C1 still has a lot of idle bandwidth, Homa's average throughput only reaches 0.4265Gbps, and is not recovered until 45s, which means that congestion on the spine block has become the bottleneck limiting Homa's throughput and downlink utilization. Unfortunately, Homa uses only one single path and thus cannot eliminate this bottleneck, resulting in its average throughput in 60 seconds being reduced to only 0.6326Gbps.

On the contrary, although the throughput of ProactMP on subflow 1 also decreases with the occurrence of congestion on CORES1, the sender (S1) of ProactMP can detect the decline of path quality on subflow 1 by observing the difference between the RTTs of subflows. It can be seen in Fig. 10(b) that ProactMP quickly transfers part of the load on subflow 1 to subflow 2 as congestion occurs, and reduces the amount of data sent on subflow 1 to relieve the congestion on CORES1. During the congestion on CORES1, ProactMP ensures that each subflow experiences the same degree of congestion by balancing load among multiple subflows, thus still achieving an average total throughput of 0.9134Gbps, which is not significantly lower than that with no congestion. ProactMP's average total throughput in 60 seconds reaches to 0.9185Gbps, an increase of 45.2% compared to that of Homa, which means that ProactMP successfully eliminates the bottleneck generated by spine blocks by utilizing multipath capabilities and improves the overall performance.

F. Performance under Realistic DCN Traffics

In this section, we use realistic datacenter network traffics to verify the performance of ProactMP in practical applications, and evaluate whether ProactMP's performance in practical environments meets the expectations. We adopt two realistic DCN workloads, W1 [6] and W5 [40]. We randomly inject traffic into the network based on the flow size and flow distribution of W1 and W5, respectively, to simulate the network load of realistic datacenters, and control the total number of flows in the network to 300. We calculate the FCTs of ProactMP and two MPTCP variants under different workloads, the cumulative distribution function (CDF) of FCTs for different schemes is shown in Fig. 11.

As shown in Fig. 11, ProactMP achieves the shortest overall FCTs under both W1 and W5 workloads. Compared with Olia and wVegas, it ensures that more small flows are completed in a shorter time and achieves shorter tail FCT for large flows. Under workload W1, ProactMP reduces the average FCT by 34.3% compared to Olia and 24.05% compared to wVegas.



(a) Workload W1 (b) Workload W5

Fig. 11. Cumulative distribution function of FCTs for different schemes under two realistic DCN workloads.

Under workload W5, ProactMP reduces the average FCT by 53.74% compared to Olia and 46.94% compared to wVegas. wVegas cannot guarantee the fast completion of small flows smaller than 10KB, but its delay-based window adjustment strategy reduces the buffer occupancy and overflow, thus still achieving an average FCT of over 12.8% shorter than Olia. Overall, ProactMP can adapt to the complex and ever-changing network traffics in practical applications, and provides a better user experience than typical multipath schemes.

On the whole, compared with other protocols, ProactMP can ensure a lower queuing delay and a more stable RTT when incast occurs, and avoid the serious consequences caused by excessive buffer occupation and overflow. It also ensures that all participating flows can achieve the highest average throughput during incast, thus realizing a reasonable balance between delay and throughput. When the load increases or congestion occurs on a spine block, ProactMP can still ensure the connection throughput and improve the overall performance of the network by achieving load balancing and congestion migration between multiple subflows. In addition, ProactMP achieves the lowest average FCT when dealing with realistic DCN traffics, and ensures that most flows can be completed with shorter FCTs.

V. Conclusion

In this paper, we proposed ProactMP, a proactive multipath transport protocol for datacenters. ProactMP is designed to be connectionless, it transmits a certain amount of data blindly in the first RTT, so as to better support small flows in low-latency networks and achieve the best delay for small flows even under high loads. ProactMP deploys a coupled congestion control mechanism with overcommitment to ensure efficient and correct data transmission through multiple paths, which globally adopts a receiver-driven rate control strategy and locally allocates the transmit quantity of data at the sender. Therefore, it guarantees that each connection or subflow has an appropriate number of bytes in flight in the network, and thus achieves load balance and avoids buffer overflow. We implemented ProactMP in the Linux system and conducted a series of experiments in our datacenter testbed to verify its effectiveness. Experimental results show that ProactMP achieved better flow completion time, link utilization, fairness and latency than TCP variants, MPTCP variants, and a leading proactive transport protocol.

ACKNOWLEDGMENT

This work is supported in part by the National Natural Science Foundation of China (NSFC) under Grants No. 62302472 and No. 62201540, the Fundamental Research Funds for the Central Universities under grant No. WK2100000039, and Youth Innovation Promotion Association of Chinese Academy of Sciences (CAS) under Grant No. Y202093.

REFERENCES

[1] A. Singh, J. Ong, A. Agarwal, G. Anderson, A. Armistead, *et al.*, "Jupiter rising: A decade of Clos topologies and centralized control in Google's datacenter network," in *Proceedings of the 29th ACM Conference on Special Interest Group on Data Communication (SIGCOMM)*, pp. 183–197, ACM, 2015.

[2] L. Poutievski, O. Mashayekhi, J. Ong, A. Singh, M. M. B. Tariq, *et al.*, "Jupiter evolving: Transforming google's datacenter network via optical circuit switches and software-defined networking," in *Proceedings of the 36th Conference of the ACM Special Interest Group on Data Communication (SIGCOMM)*, pp. 66–85, ACM, 2022.

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

[4] Y. Chen, R. Griffith, J. Liu, R. H. Katz, and A. D. Joseph, "Understanding TCP incast throughput collapse in datacenter networks," in *Proceedings of the 1st ACM Workshop on Research on Enterprise Networking (WREN)*, pp. 73–82, ACM, 2009.

[5] I. Cho, K. Jang, and D. Han, "Credit-scheduled delay-bounded congestion control for datacenters," in *Proceedings of the 31st Conference of the ACM Special Interest Group on Data Communication (SIGCOMM)*, pp. 239–252, ACM, 2017.

[6] M. Alizadeh, A. G. Greenberg, D. A. Maltz, J. Padhye, P. Patel, *et al.*, "Data center TCP (DCTCP)," in *Proceedings of the 24th ACM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM)*, pp. 63–74, ACM, 2010.

[7] C. Clos, "A study of non-blocking switching networks," *The Bell System Technical Journal*, vol. 32, no. 2, pp. 406–424, 1953.

[8] M. A. Qureshi, Y. Cheng, Q. Yin, Q. Fu, G. Kumar, *et al.*, "PLB: Congestion signals are simple and effective for network load balancing," in *Proceedings of the 36th Conference of the ACM Special Interest Group on Data Communication (SIGCOMM)*, pp. 207–218, ACM, 2022.

[9] Y. Li, R. Miao, H. H. Liu, Y. Zhuang, F. Feng, *et al.*, "HPCC: High precision congestion control," in *Proceedings of the 33rd ACM Conference on Special Interest Group on Data Communication (SIGCOMM)*, pp. 44–58, ACM, 2019.

[10] V. Addanki, O. Michel, and S. Schmid, "PowerTCP: Pushing the performance limits of datacenter networks," in *Proceedings of the 19th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, pp. 51–70, USENIX, 2022.

[11] P. X. Gao, A. Narayan, G. Kumar, R. Agarwal, S. Ratnasamy, and S. Shenker, "pHost: Distributed near-optimal datacenter transport over commodity network fabric," in *Proceedings of the 11th ACM Conference on Emerging Networking Experiments and Technologies (CoNEXT)*, pp. 1–12, ACM, 2015.

[12] M. Handley, C. Raiciu, A. Agache, A. Voinescu, A. W. Moore, *et al.*, "Re-architecting datacenter networks and stacks for low latency and high performance," in *Proceedings of the 31st Conference of the ACM Special Interest Group on Data Communication (SIGCOMM)*, pp. 29–42, ACM, 2017.

[13] B. Montazeri, Y. Li, M. Alizadeh, and J. K. Ousterhout, "Homa: A receiver-driven low-latency transport protocol using network priorities," in *Proceedings of the 32nd Conference of the ACM Special Interest Group on Data Communication (SIGCOMM)*, pp. 221–235, ACM, 2018.

[14] A. Ford, C. Raiciu, M. J. Handley, O. Bonaventure, and C. Paasch, "TCP extensions for multipath operation with multiple addresses," 2013. RFC 6824, IETF, Accessed on May., 2024.

[15] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," in *Proceedings of the 22nd ACM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM)*, pp. 63–74, ACM, 2008.

[16] A. G. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, C. Kim, *et al.*, "VL2: A scalable and flexible data center network," in *Proceedings of the 23rd ACM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM)*, pp. 51–62, ACM, 2009.

[17] C. Guo, G. Lu, D. Li, H. Wu, X. Zhang, *et al.*, "BCube: A high performance, server-centric network architecture for modular data centers," in *Proceedings of the 23rd ACM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM)*, pp. 63–74, ACM, 2009.

[18] D. Li and J. Wu, "On the design and analysis of data center network architectures for interconnecting dual-port servers," in *Proceedings of the 33rd IEEE Conference on Computer Communications (INFOCOM)*, pp. 1851–1859, IEEE, 2014.

[19] J. K. Ousterhout, "A Linux kernel implementation of the Homa transport protocol," in *Proceedings of the 2021 USENIX Annual Technical Conference (ATC)*, pp. 99–115, USENIX, 2021.

[20] A. M. Abdelmoniem and B. Bensaou, "T-RACKs: A faster recovery mechanism for TCP in data center networks," *IEEE/ACM Transactions on Networking*, vol. 29, no. 3, pp. 1074–1087, 2021.

[21] C. Raiciu, S. Barré, C. Pluntke, A. Greenhalgh, D. Wischik, and M. Handley, "Improving datacenter performance and robustness with multipath TCP," in *Proceedings of the 25th ACM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM)*, pp. 266–277, ACM, 2011.

[22] R. Mittal, V. T. Lam, N. Dukkipati, E. R. Blem, H. M. G. Wassel, *et al.*, "TIMELY: RTT-based congestion control for the datacenter," in *Proceedings of the 29th ACM Conference on Special Interest Group on Data Communication (SIGCOMM)*, pp. 537–550, ACM, 2015.

[23] Y. Zhu, H. Eran, D. Firestone, C. Guo, M. Lipshteyn, *et al.*, "Congestion control for large-scale RDMA deployments," in *Proceedings of the 29th ACM Conference on Special Interest Group on Data Communication (SIGCOMM)*, pp. 523–536, ACM, 2015.

[24] J. Perry, A. Ousterhout, H. Balakrishnan, D. Shah, and H. Fugal, "Fastpass: A centralized "zero-queue" datacenter network," in *Proceedings of the 28th Conference of the ACM Special Interest Group on Data Communication (SIGCOMM)*, pp. 307–318, ACM, 2014.

[25] K. Liu, Z. Jiang, J. Zhang, H. Wei, X. Zhong, *et al.*, "Hostping: Diagnosing intra-host network bottlenecks in RDMA servers," in *Proceedings of the 20th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, pp. 15–29, USENIX, 2023.

[26] I. Cho, A. Saeed, S. J. Park, M. Alizadeh, and A. Belay, "Protego: Overload control for applications with unpredictable lock contention," in *Proceedings of the 20th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, pp. 725–738, USENIX, 2023.

[27] S. Hu, W. Bai, G. Zeng, Z. Wang, B. Qiao, *et al.*, "Aeolus: A building block for proactive transport in datacenters," in *Proceedings of the 2020 Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication (SIGCOMM)*, pp. 422–434, ACM, 2020.

[28] J. Ousterhout, "The Homa transport protocol - Confluence." https://homa-transport.atlassian.net/wiki/spaces/HOMA/overview, 2023.

[29] T. Qu, R. Joshi, M. C. Chan, B. Leong, D. Guo, and Z. Liu, "SQR: In-network packet loss recovery from link failures for highly reliable datacenter networks," in *Proceedings of the 27th IEEE International Conference on Network Protocols (ICNP)*, pp. 1–12, IEEE, 2019.

[30] "Apache Dubbo - Building enterprise microservices with Dubbo!." https://dubbo.apache.org/en. Accessed on May., 2024.

[31] S. Ha, I. Rhee, and L. Xu, "CUBIC: A new TCP-friendly high-speed TCP variant," *ACM SIGOPS Operating Systems Review*, vol. 42, no. 5, pp. 64–74, 2008.

[32] L. S. Brakmo, S. W. O'Malley, and L. L. Peterson, "TCP Vegas: New techniques for congestion detection and avoidance," in *Proceedings of the 1994 ACM Conference on Communications Architectures, Protocols and Applications (SIGCOMM)*, pp. 24–35, ACM, 1994.

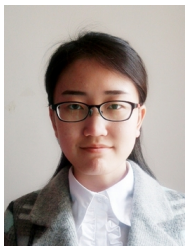
[33] N. Cardwell, Y. Cheng, C. S. Gunn, S. H. Yeganeh, and V. Jacobson, "BBR: Congestion-based congestion control," *ACM Queue*, vol. 14, no. 5, pp. 20–53, 2016.

[34] R. Khalili, N. Gast, M. Popovic, *et al.*, "Opportunistic linked-increases congestion control algorithm for MPTCP," 2013. draft-khalili-mptcp-congestion-control-05, IETF, Accessed on May., 2024.

[35] A. Walid, Q. Peng, J. Hwang, and S. Low, "Balanced linked adaptation congestion control algorithm for MPTCP," 2016. draft-walid-mptcp-congestion-control-04, IETF, Accessed on May., 2024.

[36] Y. Cao, M. Xu, and X. Fu, "Delay-based congestion control for multipath TCP," in *Proceedings of the 20th IEEE International Conference on Network Protocols (ICNP)*, pp. 1–10, IEEE, 2012.
[37] "Multipath TCP in the Linux kernel." https://www.multipath-tcp.org. Accessed on May., 2024.
[38] "Configure MPTCP with several tunables." https://multipath-tcp.org/pmwiki.php/Users/ConfigureMPTCP. Accessed on May., 2024.
[39] B. Atikoglu, Y. Xu, E. Frachtenberg, S. Jiang, and M. Paleczny, "Workload analysis of a large-scale key-value store," in *Proceedings of the 12th ACM SIGMETRICS/PERFORMANCE Joint International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS)*, pp. 53–64, ACM, 2012.
[40] A. Roy, H. Zeng, J. Bagga, G. Porter, and A. C. Snoeren, "Inside the social network's (datacenter) network," in *Proceedings of the 29th ACM Conference on Special Interest Group on Data Communication (SIGCOMM)*, pp. 123–137, ACM, 2015.
[41] C. Raiciu, M. Handley, and D. Wischik, "Coupled congestion control for multipath transport protocols," 2013. RFC 6356, IETF, Accessed on May., 2024.
[42] V. Olteanu, H. Eran, D. Dumitrescu, A. Popa, C. Baciu, *et al.*, "An edge-queued datagram service for all datacenter traffic," in *Proceedings of the 19th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, pp. 761–777, USENIX, 2022.
[43] J. Dugan, S. Elliott, B. A. Mah, J. Poskanzer, *et al.*, "iPerf - The ultimate speed test tool for TCP, UDP and SCTP." https://iperf.fr/. Accessed on May., 2024.
[44] R. Jain, "The art of computer systems performance analysis: Techniques for experimental design, measurement, simulation and modelling," *SIGMETRICS Performance Evaluation Review*, vol. 19, no. 2, pp. 5–11, 1991.
[45] G. Kumar, N. Dukkipati, K. Jang, H. M. G. Wassel, X. Wu, *et al.*, "Swift: Delay is simple and effective for congestion control in the datacenter," in *Proceedings of the 2020 Conference of the ACM Special Interest Group on Data Communication (SIGCOMM)*, pp. 514–528, ACM, 2020.

Rui Zhuang received her B.S. degree from the Department of Information Engineering, Hefei University of Technology (HFUT), in July, 2019. She is currently working toward the Ph.D degree in the School of Cyber Science and Technology, University of Science and Technology of China (USTC). Her research interests include future Internet architecture design, transmission optimization and data center network.



Jiangping Han received her bachelor's degree and the Ph.D. degree both from the Department of Electronic Engineering and Information Science (EEIS), USTC, in 2016 and 2021, respectively. She is currently a Post-Doctoral fellow with the School of Cyber Science and Technology, USTC. From Nov. 2019 to Oct. 2021, She was a visiting scholar with the School of Computing, Informatics, and Decision Systems Engineering, Arizona State University. She is currently a Post-Doctoral researcher with the School of Cyber Science and Technology, USTC. Her research interests include future Internet architecture design and transmission optimization.

Kaiping Xue (M'09-SM'15) received his bachelor degree from the Department of Information Security, University of Science and Technology of China (USTC), in 2003 and received his Ph.D. degree from the Department of Electronic Engineering and Information Science (EEIS), USTC, in 2007. From May 2012 to May 2013, he was a postdoctoral researcher with the Department of Electrical and Computer Engineering, University of Florida. Currently, he is a Professor in the School of Cyber Science and Technology, USTC. His research interests include next-generation Internet architecture design, transmission optimization and network security. He serves on the Editorial Board of several journals, including the IEEE Transactions on Dependable and Secure Computing (TDSC), the IEEE Transactions on Wireless Communications (TWC), and the IEEE Transactions on Network and Service Management (TNSM). He has also served as a (Lead) Guest Editor for many reputed journals/magazines, including IEEE Journal on Selected Areas in Communications (JSAC), IEEE Communications Magazine, and IEEE Network. He is an IET Fellow and an IEEE Senior Member.

Jian Li (M'20-SM'23) received his bachelor's degree from the Department of Electronics and Information Engineering, Anhui University, in 2015, and received doctor's degree from the Department of Electronic Engineering and Information Science (EEIS), University of Science and Technology of China (USTC), in 2020. From Nov. 2019 to Nov. 2020, he was a visiting scholar with the Department of Electronic and Computer Engineering, University of Florida. From Dec. 2020 to Dec. 2022, he was a Post-Doctoral researcher with the School of Cyber Science and Technology, USTC. He is currently an associate researcher with the School of Cyber Science and Technology, USTC. He serves as an Editor of China Communications. His research interests include quantum networking, wireless networks, and next-generation Internet architecture.

Qibin Sun (F'11) received the Ph.D. degree from the Department of Electronic Engineering and Information Science (EEIS), University of Science and Technology of China (USTC), in 1997. From 1996 to 2007, he was with the Institute for Infocomm Research, Singapore, where he was responsible for industrial as well as academic research projects in the area of media security, image and video analysis, etc. He was the Head of Delegates of Singapore in ISO/IEC SC29 WG1(JPEG). He worked at Columbia University during 2000–2001 as a Research Scientist. Currently, he is a professor in the School of Cyber Security, USTC. His research interests include multimedia security, network intelligence and security and so on. He led the effort to successfully bring the robust image authentication technology into ISO JPEG2000 standard Part 8 (Security). He has published more than 120 papers in international journals and conferences. He is an IEEE Fellow.

Jun Lu received his bachelor's degree from southeast university in 1985 and his master's degree from the Department of Electronic Engineering and Information Science (EEIS), University of Science and Technology of China (USTC), in 1988. Currently, he is a professor the School of Cyber Science and Technology, USTC. His research interests include theoretical research and system development in the field of integrated electronic information systems. He is an Academician of the Chinese Academy of Engineering (CAE).