RateSheriff: Multipath Flow-aware and Resource Efficient Rate Limiter Placement for Data Center Networks

Songshi Dou*, Yongchao He†, Sen Liu‡, Wenfei Wu§, Zehua Guo*
*Beijing Institute of Technology, †Tsinghua University, ‡Fudan University, §Peking University

Abstract—**Emerging cloud services and applications request different Quality of Service (QoS) in Data Center Networks (DCNs). To meet these various requirements, programmable switch-based rate limiters are introduced to provide performance isolation and benefit from easy control and fast deployment. However, existing programmable switch-based rate limiters have two limitations: (1) multipath flows (*i.e.*, MultiPath TCP) cannot be precisely limited, and (2) rate limiter placement solutions in DCNs are missing. These limitations could lead to poor rate limiting performance and low bandwidth utilization. In this paper, we propose RateSheriff to improve rate limiting performance by providing multipath flow-aware and resource efficient rate limiter placement for programmable switch-enabled DCNs. We identify and associate subflows to a multipath flow by extracting and comparing specific packets and header fields. By solving the formulated resource efficient rate limiter placement problem, we can improve rate limiting performance and balance memory utilization among programmable switches in DCNs. Simulation results show that RateSheriff can correctly limit the rate of multipath flows, improve rate limiting performance by up to 46%, and improve memory balancing performance by up to 79% with low computation time, compared with baselines.**

I. Introduction

Popular modern cloud services bring emerging new applications. These applications could be deployed from tenants with different Quality of Service (QoS) (*e.g.*, latency and throughput). For example, web services [1] and video streaming [2] are latency-sensitive applications; Tritonsort [3] and Hadoop [4] are throughput-intensive applications. Some throughput-intensive ultra-low latency applications (*e.g.*, AR/VR [5], virtual gaming [6], and tactile Internet [7]) can even require both high throughput and low latency [8]. Hence, it is important for cloud providers to guarantee various QoS for different applications.

Data Center Networks (DCNs) are the infrastructure of the cloud. In DCNs, many applications or tenants share the same network fabric to transmit their data and compete with each other for the bottleneck bandwidth, leading to performance fluctuation. To meet different QoS requirements, rate limiters are proposed to realize performance isolation, such as server-based limiters (*e.g.*, Carousel [9], EyeQ [10], Google BwE [11], SENIC [12], and Silo [13]) and switch-based limiters (*e.g.*, SwRL [14] and Nimble [15]). Typically, switch-based limiters outperform server-based limiters in two aspects. First, switch-based rate limiters are control plane friendly because they only need to control a few switches instead of many end-hosts [14]. Second, switch-based limiters mainly depend on switches and can be quickly deployed and updated [15].

Corresponding Author: Zehua Guo

979–8–3503–9973–8/23/$31.00 © 2023 IEEE

Programmable networking techniques (*e.g.*, OpenFlow [16] and P4 [17]) enable dynamical network control on the network and bring new opportunities to efficiently design switch-based rate limiters for DCNs. Major cloud providers (*e.g.*, Aliyun [18]) start to deploy programmable devices (*e.g.*, Intel Tofino-based P4 switches [19]) in their DCNs and take advantage of flexible flow scheduling and monitoring from the programmable devices to improve network performance. Recent works on switch-based rate limiters can provide scalable and precise rate limitation, and work compatible with congestion control algorithms (*e.g.*, TCP Vegas) [15]. However, these programmable switch-based rate limiter designs do not consider the following two major issues.

First, how to accurately limit the transmission rate of multipath-enabled flows like MultiPath TCP (MPTCP)? MPTCP is widely used in today's DCNs [20], [21], [22]. In the public cloud, a physical server can be virtualized into many Virtual Machines (VMs) controlled by different tenants, and each tenant can generate either single-path flows (*e.g.*, TCP, UDP, and ICMP) or multipath flows (*e.g.*, MPTCP) from VMs. Thus, single-path and multipath flows could co-exist in DCNs. However, existing switch-based rate limiter designs mainly consider single-path flows and thus can only limit the rate of individual subflows of a multipath flow, which would reduce rate limiting accuracy for MPTCP flows and degrade the QoS.

Second, where to place rate limiters? The control plane solution for placing rate limiters in DCNs is missing and thus would affect the performance of the limiter (*i.e.*, the impact of varying flow rate on network performance). Ideally, limiters are preferably placed at edge switches that directly connect to the servers which generate flows. In this way, the rates of flows can be immediately limited when they enter the network, and bandwidth resource can be efficiently saved [23]. However, many other switch-based network applications or functions could be deployed at edge switches at same time, and they would demand a huge memory resource [24]. Given that each switch has a limited memory resource, the placement of rate limiters in DCNs could not only affect rate limiting performance but also memory resource efficiency, which are used for other network applications or functions.

In this paper, we propose RateSheriff, a multipath flow-aware and resource-efficient rate limiter placement solution for programmable DCNs. RateSheriff mainly consists of two modules: multipath flow identification and rate limiter placement. The multipath flow identification module extracts and compares special packets and header fields for identifying subflows and associating them to a multipath flow. Rate limiter placement module differentiates single-path flows and multipath path flows and places limiters at different switches to trade off the rate limiting performance and the memory usage balance. We formulate an optimization problem called Resource Efficient Rate Limiter Placement (RERLP) problem, which is a Mixed-Integer Nonlinear Programming (MINLP) and aims to maximize the total benefit of the rate limiter placement and balance memory utilization of programmable switches. To reduce the computation complexity, we reformulate the problem and propose an efficient heuristic solution to solve the problem of deciding the proper placement of rate limiters for different flows. Simulation results show that RateSheriff can correctly limit the rate of multipath flows, improve the rate limiting performance by up to 46%, and improve the memory balancing performance by up to 79% with low computation time, compared with baselines.

In summary, our paper makes the following contributions:

- We identify two limitations of existing programmable switch-based rate limiter designs. They cannot identify and limit multipath flows, and are lack of the consideration to rationally place rate limiters for control plane.
- We propose to identify MPTCP subflows and correctly associate them to their MPTCP connections on P4 switches.
- We formulate RERLP problem for placing rate limiters, which is MINLP. To reduce the complexity, we reformulate the problem to RERLP+ problem and provide rigorous proof of its complexity.
- We propose a heuristic solution to efficiently solve the RERLP+ problem and evaluate the performance of RateSheriff under a typical DCN.

The rest of the paper is organized as follows. In Section II, we introduce background, motivation, and challenges of this paper. Section III introduces how to efficiently identify MPTCP flows and where to place rate limiters. Section IV formulates RERLP problem and reformulates it to RERLP+ problem to reduce the complexity. Section V proposes a heuristic solution to efficiently solve the problem. We evaluate the performance of RateSheriff in Section VI. Section VII introduces related works and Section VIII concludes this paper.

II. Background and motivation

In this section, we introduce the background, motivation and potential challenges in this paper.

A. Programmable switch-based rate limiter

Rate limiters are proposed to realize performance isolation, and their designs can be categorized into server-based rate limiters and switch-based rate limiters. Server-based rate limiters consume extra server resource to deploy rate limiters and need to get access to end-hosts, while switch-based rate limiters promise efficient performance isolation and easy consistency control [14]. Furthermore, switch-based rate limiters exhibit high precision and throughput without enduring the request of scheduling or queuing resource [15].

In recent year, the design and application of programmable hardware develop rapidly. Programmable hardware provides more powerful programmability and enables programmable

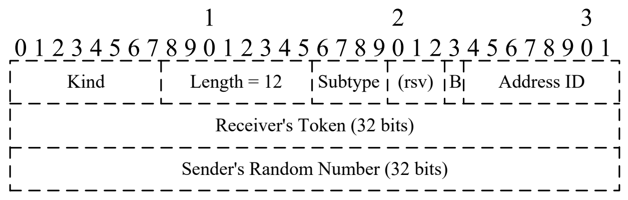


Fig. 1. `MP_JOIN` sample in the initial SYN of a subflow [32].

switches to provide flexible flow control, which gives us an opportunity to deploy a large number of rate limiters in the cloud. Switch-based rate limiters [14], [15] are proposed in both software switch's BehaVioral Models (*e.g.*, BMV2 [25]) and hardware programmable switches (*e.g.*, Intel Tofino switches). To achieve high scalability and rate limiting precision, SwRL [14] carefully customizes the token bucket rate limiter algorithm and adds arithmetic operations (*i.e.*, $\times$ and $\div$). However, existing programmable switch-based rate limiting solutions [14], [15] mainly focus on data plane implementation. How to properly place rate limiters for different types of flows in the DCN is an unsolved problem.

B. The traffic pattern of DCNs

DCNs' traffic can be generally classified into two types: long flows and short flows [26], [27]. Long flows are throughput-intensive with the sizes from 1MB to 1GB, while short flows are typically less than 1MB with low delay requirement [28]. Implementing rate limiters on either long flows or short flows increases their transmission time. The rate limiting would have a far greater impact on short flows than long flows since short flows often last only a very short time and demand strict flow completion time. On the contrary, long flows mainly focus on the throughput and do not require a demanding flow completion time, and more than 90% of the data is transmitted through these long flows [29]. Therefore, limiting the rate of long flows could save a large amount of bandwidth and improve the total performance of the whole network, and potentially reduce the queuing delay and the flow completion time of short flows [30], [31]. Major cloud providers (*e.g.*, Aliyun [23]) propose an FPGA-based long flows detection and rate limiting system to effectively realize tenant performance isolation.

C. The popularity of multipath transmission

Multipath transmission becomes popular in DCNs to meet increasing demands for cloud services and tenants by improving network resource utilization and add reliability efficiency [33]. MPTCP [32], [34] is one representative enabler of multipath transmission and can accommodate to various network scenarios (*e.g.*, multihomed servers, data centers, and mobile clients) [20]. Specifically, MPTCP brings great benefits to DCNs (*e.g.*, improving the robustness of data centers [21], [35], enhancing transmission reliability [36], and realize last hop fairness [37]). MPTCP is efficiently used to handle long flows in today's data centers since it can significantly improve long flows' throughput and reduce their flow completion time [38]. Thus, MPTCP flows can be the dominant traffic in DCNs.

However, existing programmable switch-based rate limiting solutions cannot correctly limit the rate of MPTCP flows. Typically, one MPTCP flow consists of several subflows, each of which can be forwarded on an individual path. If the subflows of an MPTCP flow cannot be correctly associated together, individually limiting each subflow's rate does not guarantee to properly limit the rate of MPTCP flows. Based on our observation, more than 80% of MPTCP flows are inaccurately rate limited if MPTCP flows and single-path flows are not differentiated from each other (Section VI-C).

D. *Emerging in-network computing*

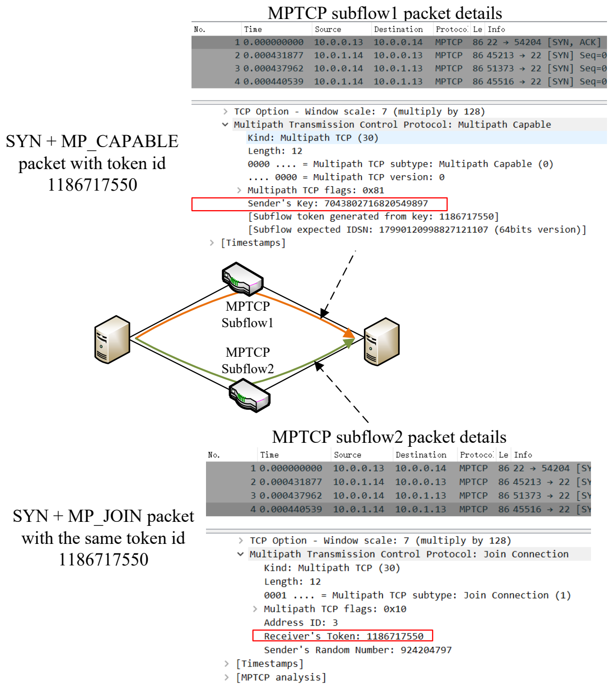
In-network computing is an emerging trend and attract the attention of industries and academia [39]. Many in-network applications (*e.g.*, key-value caching [40], coordination service [41], and gradient aggregation [42]) propose to deploy on programmable switches to improve network and application performance. To be specific, NetCache [40] provides efficient key-value cache by efficiently observing, indexing, caching, and storing hot key-value items in the switch. NetChain [41] offers low-latency in-network coordination service by storing data and process queries in the switch to eliminate the query processing and cut end-to-end latency. ATP [42] accelerates the training process of distributed machine learning by migrating the gradient aggregation in servers to programmable switches. It is expected that more applications will be deployed in programmable switches. However, deploying these in-network applications on programmable switches is restricted by the switch memory constraint [24]. Thus, how to efficiently and effectively utilize the limited memory resource of programmable switches to deploy emerging in-network applications is a crucial issue.

E. *Challenges*

Based on the above analysis, we find that designing a rate limiter placement solution is not trivial. We face two major challenges for designing an efficient rate limiter placement solution.

1) How to fast and accurately identify MPTCP flows. In DCNs, single-path flows and multipath flows could co-exist. In the view of switches, subflows of an MPTCP flow can be viewed as individual flows since they usually use different five-tuples [34]. Due to the traffic variation, the identification process should be quick enough so that the rate limiter placement could be done as soon as possible. Hence, how to propose a lightweight solution to accurately distinguish single-path flows and subflows of a multipath flow and associate subflows to a multipath flow is a challenge on programmable switches.

2) How to place rate limiters to realize high limiting performance and resource efficiency. The different placement of rate limiters in DCNs affects the performance of rate limiting. If we place the rate limiter for each flow at the edge switch, the limiter can achieve the best limiting performance since the rate of the flow can be immediately limited when it enters the network, and bandwidth resource can be saved efficiently.


MPTCP subflow1 packet details
SYN + MP_CAPABLE packet with token id 1186717550
MPTCP Subflow1
MPTCP Subflow2
MPTCP subflow2 packet details
SYN + MP_JOIN packet with the same token id 1186717550

```
No.  Time         Source     Destination  Protocol Le Info
   1 0.000000000  10.0.0.13  10.0.0.14    MPTCP    86 22 → 54204 [SYN, ACK]
   2 0.000431877  10.0.1.14  10.0.0.13    MPTCP    86 45213 → 22 [SYN] Seq=0
   3 0.000437962  10.0.0.14  10.0.1.13    MPTCP    86 51373 → 22 [SYN] Seq=0
   4 0.000440539  10.0.1.14  10.0.1.13    MPTCP    86 45516 → 22 [SYN] Seq=0

  > TCP Option - Window scale: 7 (multiply by 128)
  ∨ Multipath Transmission Control Protocol: Multipath Capable
      Kind: Multipath TCP (30)
      Length: 12
      0000 .... = Multipath TCP subtype: Multipath Capable (0)
      .... 0000 = Multipath TCP version: 0
    > Multipath TCP flags: 0x81
      Sender's Key: 7043802716820549897
      [Subflow token generated from key: 1186717550]
      [Subflow expected IDSN: 17990120998827121107 (64bits version)]
> [Timestamps]
```

```
No.  Time         Source     Destination  Protocol Le Info
   1 0.000000000  10.0.0.13  10.0.0.14    MPTCP    86 22 → 54204 [SY
   2 0.000431877  10.0.1.14  10.0.0.13    MPTCP    86 45213 → 22 [SY
   3 0.000437962  10.0.0.14  10.0.1.13    MPTCP    86 51373 → 22 [SY
   4 0.000440539  10.0.1.14  10.0.1.13    MPTCP    86 45516 → 22 [SY

  > TCP Option - Window scale: 7 (multiply by 128)
  ∨ Multipath Transmission Control Protocol: Join Connection
      Kind: Multipath TCP (30)
      Length: 12
      0001 .... = Multipath TCP subtype: Join Connection (1)
    > Multipath TCP flags: 0x10
      Address ID: 3
      Receiver's Token: 1186717550
      Sender's Random Number: 924204797
  > [Timestamps]
  > [MPTCP analysis]
```

Fig. 2. Multipath flow identification.

However, limited memory resource on edge switches could be competed by many different in- network applications. Furthermore, many critical factors (*e.g.*, flow rate and flow type) could also hugely affect the overall performance of the rate limiting. Thus, how to trade off balanced memory utilization and good rate limiting performance is also challenging.

III. MPTCP FLOW IDENTIFICATION AND LIMITER PLACEMENT

In this section, we analyze MPTCP flows' two specific processes, propose a solution for programmable switches to identify MPTCP flow, and discuss several key considerations for placing rate limiters.

A. *The process of establishing an MPTCP flow*

The process of establishing an MPTCP flow mainly consists of two parts: the establishment of an MPTCP connection and the join of subflows.

1) The establishment of an MPTCP connection. MPTCP operations are transmitted in the optional fields of the TCP header [32]. In the "Kind Number" of TCP option field as shown in Fig. 1, 30 denotes that a flow belongs to an MPTCP connection[1]. To establish an MPTCP connection, we need to first establish a subflow connection following the typical three-way handshake for establishing a single-path TCP connection, and `MP_CAPABLE` option is activated in each packet of the connection's first subflow. During the three-way handshake, the connection's sender and receiver exchange their keys.

2) The join of subflows. Once an MPTCP connection is established, new subflows can be added to the connection. Each new subflow starts with a TCP SYN packet. However,

[1]In this paper, we use a connection and a flow interchangeably.

```
parser parse_tcp {
    extract(tcp);
    extract(tcp_option);
    return select(latest.kind) {
        30 : parse_mptcp;
        default: ingress;}}
parser parse_mptcp {
    extract(mptcp);
    return select(latest.subtype) {
        0: parse_capable;
        1: parse_join;
        default: ingress;}}
parser parse_capable {
    extract(capable);
    extract(stat);
    return ingress;}
parser parse_join {
    extract(join);
    extract(stat);
    return ingress;}
```

Fig. 3. Main code of MPTCP option parsing for P4 switches.

unlike the first subflow using MP_CAPABLE, the new subflow uses MP_JOIN in MPTCP option, which is shown in Fig. 1. MP_JOIN is used to find the right MPTCP connection to join. In the initial SYN of the new subflow, the packet contains receiver's token, which is generated from the key exchanged in the MP_CAPABLE handshake [43]. The token enables new subflows to join the right MPTCP connection.

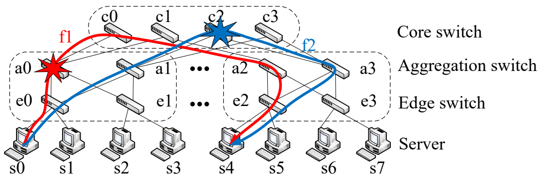
B. Programmable switch-based MPTCP connection identification

P4 switches are able to parse and extract information from TCP option fields [44], [45]. We propose an MPTCP flow identification solution on P4 switches. The solution can be broadly divided into two parts. For the first part, we parse and extract the key that a sender and a receiver share with each other through the MP_CAPABLE MPTCP option during establishing the first subflow of an MPTCP connection. Then, we generate a token from the key for this connection and store the token and the flow's ID. The second part is to associate subflows to the right MPTCP connection. We parse and extract the receiver's token in the packet of MP_JOIN MPTCP option when a new subflow joins the MPTCP connection. Then, we compare the new token with the stored tokens to associate this subflow to its MPTCP connection.

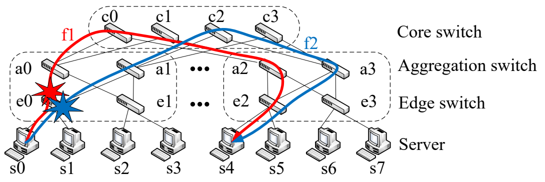
We validate our multipath flow identification solution on a real MPTCP connection-based experiment. Fig. 2 shows the experiment and its Wireshark screenshot. We generate one MPTCP flow containing two subflows between the sender and the receiver. Subflow 1 delivers packets with MP_CAPABLE option, and subflow 2's packets contain MP_JOIN option. In this figure, the token generated from the key in subflow 1's packet is the same as the token extracted from the packet of subflow 2. Thus, this experiment validates our solution's functionality. We implemented this module in a P4 switch. Fig. 3 shows the main code of MPTCP option parsing for P4 switches.

C. Considerations for rate limiter placement

After associating subflows to their right MPTCP connection, we are able to distinguish between both MPTCP flows and single-path flows. Compared with single-path flows, how to place rate limiters for MPTCP flows is a more complicated problem since we have some extra concerns (*e.g.*, the consistency of MPTCP flows) when placing rate limiters. Thus, we will discuss several major concerns of rate limiter placement for MPTCP flows in detail.



(a) Case 1



(b) Case 2

Fig. 4. Two motivation examples to illustrate the considerations of rate limiter placement for MPTCP connections. Note that separately placing the rate limiter for individual subflow could lead to inaccurate rate limiting.

We have two considerations for placing rate limiters for MPTCP flows: (1) separately limiting individual subflow could bring inaccurate limitations on the rate of their MPTCP connection, and (2) the placement of rate limiters affects the limiting performance. Note that each bi-directional flow is deemed as two individual flows with different five tuples, and the rate limiter is separately placed for each flow, thus the placement for each flow would not have any impact on their reverse flow. We use two motivation examples in Fig. 4 to illustrate our considerations. In Fig. 4, MPTCP connection m ($s_0 \rightarrow s_4$) has two subflows (*i.e.*, f_1 and f_2) forwarded on two different paths in a 3-layer 4-pod DCN, and m is expected to be limited to 200 Mb/s.

In Fig. 4(a), rate limiters of subflows f_1 and f_2 are placed at aggregation switch a_0 and core switch c_2, respectively. In this case, the rates of f_1 and f_2 can only be individually limited. The network congestion states change significantly. Assume f_1's rate reaches 150 Mb/s, and f_2's rate reaches 70 Mb/s. If f_1 and f_2's limiters set to 100 Mb/s, m can only work at 170 Mb/s (*i.e.*, 100 Mb/s from f_1 and 70 Mb/s from f_2), over-restricted m's rate. If f_1 and f_2's limiters are set to 150 Mb/s, m's rate becomes 220 Mb/s, which exceeds the expected rate limitation.

In Fig. 4(b), rate limiters of f_1 and f_2 are placed at edge switch e_0. Since f_1 and f_2 belong to the same MPTCP, and their limiters are placed in the same switch, we can associate the two flows' limiters together and let m restrict to its ideal rate 200 Mb/s. Compared with Fig. 4(a), rate limiters accurately limit the rate two subflows since they are placed at the same switch. Thus, two subflows do not incur the waste of bandwidth utilization in the network.

In summary, based on the above-mentioned considerations and motivation examples, we find that rate limiters can accurately limit the rate of subflows from the same MPTCP connection if they are placed at the same switch. In the following section, we will formulate our problem to decide the proper placement of each flow with high overall benefit performance and balanced memory usage.

IV. Problem Formulation

In this section, we formulate the RERLP problem to efficiently place rate limiters in DCNs. To efficiently solve the proposed RERLP problem, we reduce the computation complexity by reformulating it to the RERLP+ problem.

A. System description

A DCN consists of N programmable switches. The set of switches are $\mathcal{S} = \{s_1, ..., s_i, ..., s_N\}$. Single-path flows and MPTCP flows co-exist in the DCN. The set of L single-path flows from switches $\mathcal{S}$ is $F = \{f_1, ..., f_j, ..., f_L\}$, while the set of MPTCP subflows from switches $\mathcal{S}$ is $F = \{f_{L+1}, ..., f_j, ..., f_K\}$. If flow f_j's forwarding path traverses switch s_i, we have $\alpha_{ij} = 1$; Otherwise $\alpha_{ij} = 0$. We use the binary variable x_{ij} to denote the relationship between the rate limiter of flow f_j and switch s_i. If the rate limiter of flow f_j is placed at switch s_i, we have $x_{ij} = 1$; Otherwise $x_{ij} = 0$.

B. Constraints

Our problem has four constraints, which are detailed as follows.

1) Rate limiter placement location constraint: The rate limiter can only be placed at the switch which the flow's path traverses:

$$x_{ij} \leq \alpha_{ij}, \forall i \in [1, N], \forall j \in [1, K]. \tag{1}$$

2) Rate limiter placement number constraint: Each flow's rate limiter can be only placed at one switch. That is:

$$\sum_{i=1}^{N} x_{ij} = 1, \forall j \in [1, K]. \tag{2}$$

3) MPTCP subflow constraint: For each MPTCP subflow from the same MPTCP connection, the rate limiter can be only placed at the same switch. We use $p_{jj'}$ to denote the relationship between MPTCP subflows. If MPTCP subflows f_j and $f_{j'}$ are from the same MPTCP connection, we have $p_{jj'} = 1$; Otherwise $p_{jj'} = 0$. The constraint can be formulated as follows:

$$p_{jj'} \leq \sum_{i=1}^{N} x_{ij} * x_{ij'}, \forall j, j' \in [L+1, K], j' \neq j. \tag{3}$$

4) Memory usage constraint: We use u_i to denote the memory usage of switch s_i, and m_j to denote the memory consumption of placing flow f_j's rate limiter. That is:

$$u_i = \sum_{j=1}^{K} x_{ij} * m_j, \forall i \in [1, N]. \tag{4}$$

The current memory usage at each switch should not exceed the switch's memory capacity M_i. It can be written as follows:

$$u_i \leq M_i, \forall i \in [1, N]. \tag{5}$$

C. Objective functions

Our problem has two objectives. The first objective is to maximize the overall performance benefit of rate limiter placement. The overall performance benefit is generally determined by rate limiter placement and flow rate difference. First, we find that a rate limiter works best if it is placed at the switch close to the flow's source server, because the impact of varying flow rate on network performance (*e.g.*, link utilization) is minimized. In this way, the rates of flows can be immediately limited when they enter the network, and bandwidth resource can be efficiently saved [23]. Note that congestion control algorithms can also help to limit the flow rate if the rate limiter is not deployed at the switch close to the flow's source server. However, the congestion control schemes at end-hosts (*i.e.*, flows) take at least one RTT to react to the rate limiter's operation, which means that excessive extra traffic will be injected into the DCN before the rate limiter literally limits the transmission rate of a flow, especially under highly dynamic micro-burst DCN traffic [46].

Second, each flow could have a different flow rate limitation, and flows with high rate difference before and after placing the rate limiter play a more important role than flows with low rate difference. Thus, we prefer to place rate limiters with high rate difference than the ones with low rate difference close to the flows' senders. We use r_j to denote the rate difference before and after the rate limiter of flow f_j is placed and use β_{ij} to denote the performance benefit if we place the rate limiter of flow f_j at switch s_i. Thus, the overall performance of rate limiter placement is formulated as follows:

$$obj_1 = \sum_{j=1}^{K} \sum_{i=1}^{N} x_{ij} * r_j * \beta_{ij}.$$

Recall that many other in-network applications or functions could be deployed at switches at same time, and they would demand a huge memory resource [24]. If some switches' memory is fully occupied, we cannot successfully place rate limiters for flows at these switches, and the network performance would decrease. Thus, the second objective is to let switches have balanced memory utilization. We use h to denote the maximum memory utilization of all switches, which is:

$$h = \max(u_i), \forall i \in [1, N]. \tag{6}$$

Thus, the second objective is:

$$obj_2 = h.$$

Typically, formulating one problem by combining the two objectives into one objective is a good choice since it only needs to solve one problem. However, the two objectives are opposite. Our first objective is to maximize the overall benefit while our second objective aims to minimize maximum memory utilization. Thus, to combine the two objectives together, we change the second objective to maximize the negative value of maximum memory utilization. Thus, the objective function of the problem is formulated as follows:

$$obj = obj_1 - \lambda * obj_2 = \sum_{j=1}^{K}\sum_{i=1}^{N} x_{ij} * r_j * \beta_{ij} - \lambda * h,$$

where $\lambda \geq 0$ is a positive constant to give different weights of the two objectives.

D. Problem formulation

Our problem is to maximize the overall performance benefit of rate limiter placement and balance memory utilization of programmable switches across the network by rationally placing rate limiters. The RERLP problem is formulated as follows:

$$\begin{aligned} \max_{x,h} \quad & \sum_{j=1}^{K}\sum_{i=1}^{N} x_{ij} * r_j * \beta_{ij} - \lambda * h \\ \text{s.t.} \quad & (1)(2)(3)(4)(5)(6), \\ & h \geq 0, x_{ij} \in \{0,1\}, \forall i \in [1,N], \forall j, j' \in [1,K], j' \neq j, \end{aligned} \tag{P1}$$

where $\{x_{ij}\}$ and $\{x_{ij'}\}$ are binary design variables, h is a positive design variable, $\{\alpha_{ij}\}$ and $\{p_{jj'}\}$ are binary constants, λ, $\{m_j\}$, $\{r_j\}$, $\{\beta_{ij}\}$, and $\{M_i\}$ are positive constants.

E. Problem reformulation

In RERLP problem, the objective function is linear, variables are binary integers, and constraints are nonlinear. Thus, this problem is an MINLP. The high complexity of the RERLP problem comes from Eq. (3) since two binary variables are multiplied in the equation. The proposed MINLP is hard to solve due to its high complexity. Therefore, to efficiently solve the problem, we need to reformulate our proposed problem.

Based on the previous discussion, about 90% of the data is transmitted through long flows in a typical DCN [29], and MPTCP plays a critical role during the data transmission of these long flows since MPTCP can efficiently improve the throughput [38]. To greatly save the network bandwidth and improve the total performance of the whole network, MPTCP flows should be considered with a higher priority than single-path flows when placing rate limiters. Therefore, in order to control the traffic efficiently under bursty DCN traffic and reduce the computation complexity, we propose to pre-place rate limiters for each MPTCP flow at the edge close to senders.

By separating the placement of rate limiters for MPTCP flows and single-path flows, Eq. (3) would no longer exist, and Eqs. (1), (2), (4), (5), and (6) can be respectively reformulated as follows:

$$x_{ij} \leq \alpha_{ij}, \forall i \in [1,N], \forall j \in [1,L], \tag{7}$$

$$\textstyle\sum_{i=1}^{N} x_{ij} = 1, \forall j \in [1,L], \tag{8}$$

$$u_i = \textstyle\sum_{j=1}^{L} x_{ij} * m_j, \forall i \in [1,N], \tag{9}$$

$$u_i + U_i \leq M_i, \forall i \in [1,N], \tag{10}$$

$$h = \max(u_i + U_i), \forall i \in [1,N], \tag{11}$$

where U_i is the current memory usage of switch s_i after the placement of MPTCP flows.

Thus, we can reformulate our proposed RERLP problem to the new RERLP+ problem, which is a Mixed-Integer Linear Programming (MILP) as follows:

$$\begin{aligned} \max_{x,h} \quad & \sum_{j=1}^{L}\sum_{i=1}^{N} x_{ij} * r_j * \beta_{ij} - \lambda * h \\ \text{s.t.} \quad & (7)(8)(9)(10)(11), \\ & h \geq 0, x_{ij} \in \{0,1\}, \forall i \in [1,N], \forall j \in [1,L]. \end{aligned} \tag{P2}$$

V. Solution

In this section, we exhibit rigorous proof of the RERLP+ problem to be NP-hard and propose an efficient heuristic solution to solve the problem.

A. Complexity Analysis

Theorem 1. *For a special case with two conditions: (1) the second objective is not considered, and (2) the rate limiter is only placed at the switch which the flow traverses, the RERLP+ problem is NP-hard.*

Proof: We first introduce the *Generalized Assignment Problem (GAP)* [47]. The GAP aims to maximize the value of assigning L jobs to N agents such that each job is precisely assigned to one agent subject to resource availability restrictions on the agents. A typical formulation of the GAP is shown below:

$$\begin{aligned} \max_{x} \quad & \sum_{j=1}^{L}\sum_{i=1}^{N} c_{ij} * x_{ij} \\ \text{s.t.} \quad & \sum_{j=1}^{L} a_{ij} * x_{ij} \leq b_i, \forall i \in [1,N], \\ & \sum_{i=1}^{N} x_{ij} = 1, \forall j \in [1,L], \\ & x_{ij} \in \{0,1\}, \forall i \in [1,N], \forall j \in [1,L], \end{aligned} \tag{P3}$$

where c_{ij} is the value of having agent i do job j, a_{ij} is the resource required by agent i to do job j, and b_i is the resource availability of agent i. Binary variable x_{ij} equals 1 if agent i performs job j, otherwise it equals 0. It has been proved when assigning multiple jobs to an agent and ensuring each job is performed exactly by one agent, the GAP is NP-hard [48].

We then prove for a special case of conditions (1) and (2), problem (P2) and the GAP are equivalent problems. Given condition (1) that the second objective is not considered and condition (2) that the rate limiter is only placed at the switch which the flow traverses, we do not need the obj_2 and the constraint (7).

We can use ω_{ij} to denote the benefit of placing rate limiter at switch s_i for flow f_j. Thus, we have:

$$\omega_{ij} = r_j * \beta_{ij}, \forall i \in [1,N], \forall j \in [1,L].$$

By combining Eqs. (9) and (10) and using G_i to denote $M_i - U_i$, we have:

$$\sum_{j=1}^{L} m_j * x_{ij} \leq G_i, \forall i \in [1,N]. \tag{12}$$

Following the above two conditions, our RERLP problem can be reformulated as follows:

$$\begin{aligned} \max_{x} \quad & \sum_{j=1}^{L}\sum_{i=1}^{N} \omega_{ij} * x_{ij} \\ \text{s.t.} \quad & (8)(12), \\ & x_{ij} \in \{0,1\}, \forall i \in [1,N], \forall j \in [1,L]. \end{aligned} \tag{P4}$$

Problem (P4) aims to maximize the benefit of placing rate limiters at N switches for L flows such that the rate limiter for each flow is precisely placed at one switch subject to the availability of memory on the switches. We can treat switch s_i and flow f_j in problem (P4) as agent i and job j in the GAP. By this construction, it is easy to prove that there exists the maximum benefit of placing rate limiters by selecting proper switches in $\mathcal{S}$ of flows F, if and only if there exists the optimal solution of the GAP by assigning L jobs to N agents. The construction can be done in polynomial time. In problem (P4), the placement of rate limiters for flows at switches could be many to one. Since the GAP is NP-hard when one agent has multiple jobs, and each job is performed exactly by one agent [48], problem (P4) is NP-hard. □

Problem (P4) is a special case of the RERLP problem and is NP-hard. Therefore, we can draw the following conclusion:

Theorem 2. *The RERLP problem is NP-hard.*

B. Heuristic solution

The typical solution of the RERLP+ problem is to get its optimal result using existing IP optimization solvers. However, the RERLP+ problem is NP-hard with high complexity. As the network size increases, the solution space could increase significantly, and finding a feasible solution could cost a long time or perhaps is impossible. To this end, we design the heuristic algorithm to solve the problem and achieve the trade-off between performance and time complexity.

The key idea of the heuristic algorithm is to test and place the rate limiter for each flow by following descending order of the rate difference before and after the rate limiter and ensuring the memory usage of each switch is balanced. Details are summarized in Algorithm 1. We generate P_j to denote the set of switches for potential rate limiter placement of flow f_j in the descending order of their performance benefit β_{ij}, where $P_j = \{s_i \mid i \in [1,N]\}$. Note that our solution is designed for programmable DCNs, and each TCP flow is typically forwarded on a single path. If the network condition changes due to either network congestion or node/link failures and flows have to reroute to accommodate network variation, our proposed solution will recalculate the placement strategy according to the current network state and further re-place all rate limiters.

In line 1 of Algorithm 1, we initialize $\mathcal{X}$ to be empty at the beginning of the algorithm. In line 2, we sort all flows in the flow set F in the descending order of their flow rate difference before and after the rate limiter. The sorting operation helps us to test the potential placement of rate limiters based on their benefit performance. In line 3, we generate the positive constant Avg_Mem, which is a positive constant denoting the ideal memory usage of each switch, where Avg_Mem = $\sum_{j=1}^{L} m_j/N$ + 1. In lines 5-9, we set the value of ideal memory capacity Cap to make sure that the memory usage of each switch is balanced after the placement. If the maximum current usage of switches $\max(U_i)$ is greater than the ideal memory usage Avg_Mem, it means that a given amount of space has already been occupied by the placement of MPTCP flows, thus the ideal memory usage of each flow could only be limited to $\max(U_i)$; Otherwise, Cap is set to be the value of Avg_Mem.

Algorithm 1 Heuristic solution

```
Input: $F$, $N$, $L$, $P_j$, $m_j$, $M_i$, $U_i$, $\beta_{ij}$, $Cap$;
Output: $\mathcal{X}$;
 1: $\mathcal{X} = \emptyset$;
 2: Sort all flows in the set $F$ in the descending order of their
    flow rate difference before and after the rate limiter;
 3: Generate $Avg\_Mem$;
 4: // set the memory utilization capacity of each switch to
    ensure the memory usage is balanced.
 5: if $\max(U_i) \geq Avg\_Mem$ then
 6:     $Cap$ = $\max(U_i)$;
 7: else
 8:     $Cap$ = $Avg\_Mem$;
 9: end if
10: // test potential placement based on the descending order
    of their flow rate difference.
11: for $f_{j_0} \in F$ do
12:     for $s_{i_0} \in P_{j_0}$ do
13:         // rate limiter is placed at switch $s_{i_0}$ for flow $f_{j_0}$.
14:         if $U_{i_0} + m_{j_0} \leq Cap$ and $M_{i_0} \geq m_{j_0}$ then
15:             $M_{i_0} = M_{i_0} - m_{j_0}$, $U_{i_0} = U_{i_0} + m_{j_0}$;
16:             $F \leftarrow F \setminus f_{j_0}$, $\mathcal{X} \leftarrow \mathcal{X} \cup (i_0, j_0)$;
17:             break;
18:         end if
19:     end for
20: end for
21: // place rate limiters for the rest of flows by relaxing the
    constraint of balancing memory usage.
22: if $F! = \emptyset$ then
23:     for $f_{j_0} \in F$ do
24:         for $s_{i_0} \in P_{j_0}$ do
25:             if $M_{i_0} \geq m_{j_0}$ then
26:                 $M_{i_0} = M_{i_0} - m_{j_0}$, $U_{i_0} = U_{i_0} + m_{j_0}$;
27:                 $F \leftarrow F \setminus f_{j_0}$, $\mathcal{X} \leftarrow \mathcal{X} \cup (i_0, j_0)$;
28:                 break;
29:             end if
30:         end for
31:     end for
32: end if
33: return $\mathcal{X}$;
```

In lines 11-32, we test all potential placements of each flow based on the descending order of their flow rate difference.

From lines 11 to 20, we test the potential placement at switch s_{i_0} for flow f_{j_0}. If this placement satisfies the constraints in Eqs. (9) and (10) and does not exceed the ideal memory capacity Cap, it is a feasible placement, and we remove flow f_{j_0} from the flow set F, update the available memory capacity and current memory usage of each switch, and confirm this placement for f_{j_0} at switch s_{i_0} in $\mathcal{X}$, as shown in lines 14-18. From lines 22 to 32, if some flows are still not placed with rate limiters, we then relax the constraint of ideal memory capacity Cap and test them again; Otherwise, all flows are placed with rate limiters or fully tested and we cannot place any new rate limiter, and the testing ends. In line 33, the algorithm returns the result and stops.

VI. Evaluation

A. Simulation setup

In a DCN, each link has the same bandwidth, and each server sends a certain number of flows to other servers. In our simulation, we use a 3-layer 8-pod fat-tree network [49], which consists of 16 core switches, 32 aggregation switches, and 32 edge switches. The bandwidth of each link is 40 Gbps. Following previous work [22], our simulations generate 100K TCP flows randomly between end-hosts, including single-path TCP flows and subflows for MPTCP flows. Each MPTCP flow consists of 2-4 subflows. RateSheriff is scalable and not sensitive to different communication patterns between end-hosts since it deploys limiters on programmable switches rather than end-hosts. Among all flows, the rates of 20% flows are selected from [2, 4, 6, 8, 10] Mbps. The previous work [14] has evaluated the functionality of switch-based limiters when the target rate varies from 1 Mbps to 1000 Mbps. Thus, we set the target rate to the minimum functional rate (i.e., 1 Mbps). The benefit of placing a rate limiter is set proportionally to the distance for the flow's source server to the switch that is placed with the limiter. Note that global network information can be collected through in-band network telemetry solutions (*e.g.*, HPCC [18] and PINT [50]). Though additional headers, which carry the status of switches, will be introduced in this case, the overall overhead is small and still under control according to these works.

B. Comparison algorithms

We compare RateSheriff with 5 baseline algorithms.

1) Nearest: all rate limiters are placed at the edge switch, which is close to the flow's source server. This solution can achieve the maximum benefit of rate limiter placement but performs the worst in terms of memory utilization balancing.
2) MultiPath-Oblivious (MP-Oblivious): this solution only focuses on improving limiter placement benefit and balancing memory utilization without differentiating single-path flows and MPTCP flows.
3) Optimal: it is the optimal solution of the RERLP+ problem that maximizes the benefit of rate limiter placement and achieves balanced memory usage.

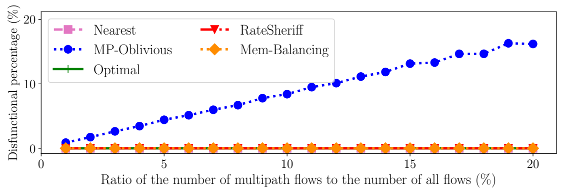


Fig. 5. The percentage of inaccurate MPTCP flows to all flows.

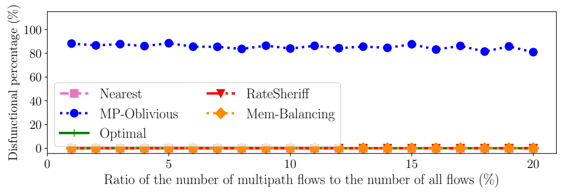


Fig. 6. The percentage of inaccurate MPTCP flows to all MPTCP flows.

4) Memory-Balancing (Mem-Balancing): this solution can differentiate single-path flows and MPTCP flows but only aims to balance memory utilization.
5) Without rate limiter: this solution presents link utilization in the DCN without placing any rate limiters.
6) RateSheriff: it differentiates single-path flows and MPTCP flows, and places rate limiters for MPTCP flows at the selected edge switches and for single-path flows at switches decided by Algorithm 1.

Note that Optimal, MP-Oblivious, and Mem-Balancing are formulated as optimization problems solved by the optimization solver GUROBI [51].

C. Performance analysis

We compare the performance of RateSheriff with other algorithms under the different traffic compositions. Specifically, the ratio of the number of multipath flows to the number of all flows changes from 1% to 20%. We also evaluate RateSheriff's functionality and computation efficiency. Following our two objectives, we use two metrics: overall benefit performance, which reflects obj_1 and maximum memory utilization, which reflects obj_2.

1) MPTCP flow identification overhead. Recall that our multipath flow identification module parses and extracts specific header fields from packets to identify subflows and associate these flows to an MPTCP connection. We conducted 1000 experiments on a P4 switch and collected the time consumed by this module. Results show that this module only consumes about 283ns on average, which is a really short processing time. Note that our proposed MPTCP flow identification scheme is the first to identify and associate MPTCP subflows on P4 switches.

2) Rate limiting accuracy. Fig. 5 shows rate limiting inaccuracy of all flows. If the rate limiters of MPTCP's subflows are placed at different switches, the MPTCP connection's rate could be limited inaccurately. In this figure, only MP-Oblivious encounters the problem of inaccurate rate limitation on MPTCP flows because MP-Oblivious does not differentiate between single-path flows and MPTCP flows when placing rate limiters. The inaccurate percentage increases gradually with the growth of multipath flow's proportion in all flows. Fig. 6 presents the result of the rate limiting inaccuracy of

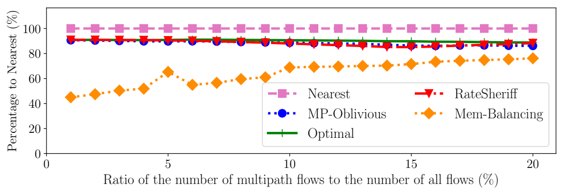


Fig. 7. Overall benefit normalized to Nearest.

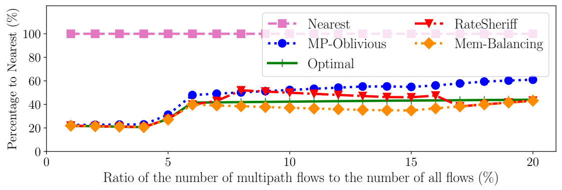


Fig. 8. Maximum memory usage normalized to Nearest.

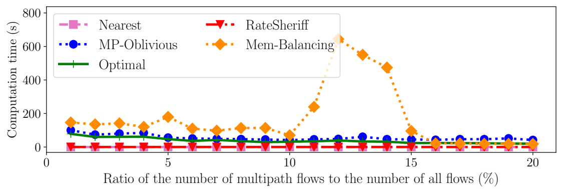


Fig. 9. Computation time.

MPTCP flows. In this figure, with MP-Oblivious, more than 80% of MPTCP flows are inaccurately rate limited. This result also verifies that differentiating single-path flows and multipath flows plays a crucial role when limiting flows' rates.

3) Overall benefit performance. Fig. 7 shows the overall benefit of different algorithms normalized to Nearest. Nearest performs the best since all the rate limiters are placed at edge switches close to flows' senders. Optimal, MP-Oblivious, and RateSheriff exhibit comparative results, which are about 90% of the Nearest. Mem-Balancing performs the worst since it does not consider the overall benefit at all. RateSheriff's performance is 46% higher than Mem-Balancing at most. Note that the major factor, which affects the overall benefit, changes as the number of multipath flows changes. In Fig. 8, when the ratio is lower than 6%, the major factor is the memory balancing performance since the network only has a small number of multipath flows, and the overall benefit relies on the limiter placement of single-path flows among all switches. However, if the ratio exceeds 6%, the overall benefit performance becomes the major factor. This is because many rate limiters are placed at edge switches, and the benefit of placing rate limiters on multipath flows directly decides the overall benefit.

4) Memory balancing performance. Fig. 8 shows the percentage of maximum memory usage normalized to Nearest. Nearest performs the worst since all rate limiters are placed at edge switches. Optimal and RateSheriff's performance is marginally worse than Mem-Balancing because they trade-off between overall benefit and balanced memory usage. Compared with Nearest, RateSheriff improves memory balancing performance up to 79%. Mem-Balancing performs best in all situations. The reason is that Mem-Balancing only aims to balance memory utilization to realize overall memory balancing performance for all switches.

5) Computation time. To show the computation efficiency, we evaluate the computation time on RateSheriff with other algorithms. Fig. 9 shows the computation time of all algorithms. Note that Nearest just places all limiters are edge switches and does not introduce any computation cost. Optimal, MP-Oblivious, and Mem-Balancing are solved by the optimization solver, and the computation time of these three schemes can reach by up to 600s. As the network size increases and with more flows forwarding in the network, finding a feasible solution with optimization solvers may need a long time or even be impossible. Except Nearest, RateSheriff performs the best and only consumes about 0.2s on average, because it is a heuristic algorithm with low complexity. Note that the computation time of RateSheriff is far less than the flow completion time of long flows in today's DCNs [27].

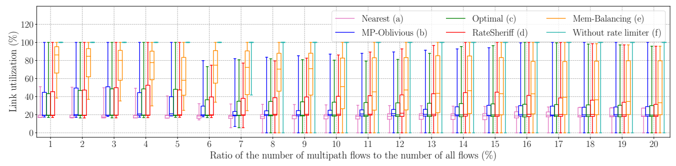


Fig. 10. Link utilization performance. (a) to (f) are used to denote the six solutions from left to right in each scenario.

6) Link utilization performance. Fig. 10 shows the link utilization performance of different algorithms. Nearest outperforms other schemes because all rate limiters are placed at edge switches. In this way, the rates of flows can be immediately limited when they enter the network, and bandwidth resource are efficiently saved. RateSheriff's performance is better than other algorithms except Nearest. In some cases, RateSheriff outperforms Optimal because Optimal aims to jointly optimize the overall benefit performance and memory balancing and thus does not perform optimally for this metric.

VII. Related work

Existing works on rate limiter can be categorized into the following two groups:

1) Server-based rate limiters. SENIC [12] deploys rate limiters on SmartNICs to realize scalable hardware-based rate limitation. It can support 10s of thousands of rate limiters and achieves good performance. Carousel [9] is a software-based rate limiting solution. It is able to limit tens of thousands of flows on a single server, but resource consumption of the server is inevitable. Google BwE [11] is proposed for WAN bandwidth allocation. It has been proved that BwE offers isolation among competing services and achieves higher link utilization. EyeQ [10] proposes to provide efficient bandwidth guarantees in DCNs. The rate limiting of EyeQ is achieved at the edge to provide bandwidth allocation. Network bandwidth, packet delay, and burstiness are jointly considered in Silo [13]. Silo is designed for multi-tenants DCNs.

2) Switch-based rate limiters. Switch-based rate limiters are newly proposed. To the best of our knowledge, there are only two existing works [14], [15] for switch-based rate limiters. SwRL [14] proposes to deploy rate limiters on programmable switches instead of on end-host software or end-host NICs. SwRL presents high scalability, which can support up to 1 million rate limiters on a single programmable switch.

Nimble [15] also focuses on on-switch rate limiters. It is TCP friendly, scalable (100K rate limiters per switch), and with high throughput (100 Gbps line rates). Recall that many programmable switch-based applications could be deployed in switches at the same time to realize different functions, which would request a huge memory resource. Thus, scalability performance would largely degrade under real-world deployment.

VIII. Conclusion

In this paper, we propose RateSheriff, a multipath flow-aware and resource efficient rate limiter placement solution for programmable DCNs. RateSheriff limits MPTCP flows' rates and balances memory usage among programmable switches in the DCN. The evaluation results show that RateSheriff can correctly limit the rate of multipath flows, and improve rate limiting performance and memory balancing performance when placing rate limiters in the DCN. In the future, we will consider other factors related to rate limiter placement, such as directly quantifying the impact of limiter placement on link utilization in the problem formulation, and other types of multipath flows (*e.g.*, multipath QUIC).

Acknowledgment

This work was supported by the National Natural Science Foundation of China under Grant 62002019 and Grant 62002066, Zhejiang Lab Open Research Project under Grant K2022QA0AB02, SongShan Laboratory Fund under Grant YYJC022022009, Beijing Institute of Technology Research Fund Program for Young Scholars, and Natural Science Foundation of Shanghai under Grant 23ZR1404900.

References

[1] A. Singh, J. Ong, A. Agarwal *et al.*, "Jupiter rising: A decade of clos topologies and centralized control in google's datacenter network," *ACM SIGCOMM CCR*, vol. 45, no. 4, pp. 183–197, 2015.

[2] K. Du, A. Pervaiz, X. Yuan *et al.*, "Server-driven video streaming for deep learning inference," in *ACM SIGCOMM'20*.

[3] A. Rasmussen *et al.*, "Tritonsort: A balanced and energy-efficient large-scale sorting system," *ACM TOCS*, vol. 31, no. 1, pp. 1–28, 2013.

[4] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The hadoop distributed file system," in *IEEE MSST'10*.

[5] Q. He, Z. Dong, F. Chen *et al.*, "Pyramid: enabling hierarchical neural networks with edge computing," in *ACM WWW'22*.

[6] B. Boudaoud *et al.*, "Gaming at warp speed: Improving aiming with late warp," in *ACM SIGGRAPH'21 Emerging Technologies*.

[7] N. Promwongsa, A. Ebrahimzadeh, D. Naboulsi *et al.*, "A comprehensive survey of the tactile internet: State-of-the-art and research directions," *IEEE COMST*, vol. 23, no. 1, pp. 472–523, 2020.

[8] M. Diarra, W. Dabbous *et al.*, "Ran-aware proxy-based flow control for high throughput and low delay embb," in *ACM MSWiM'21*.

[9] A. Saeed, N. Dukkipati, V. Valancius *et al.*, "Carousel: Scalable traffic shaping at end hosts," in *ACM SIGCOMM'17*.

[10] V. Jeyakumar, M. Alizadeh, D. Mazières *et al.*, "{EyeQ}: Practical network performance isolation at the edge," in *NSDI'13*.

[11] A. Kumar *et al.*, "Bwe: Flexible, hierarchical bandwidth allocation for wan distributed computing," in *ACM SIGCOMM'15*.

[12] S. Radhakrishnan, Y. Geng, V. Jeyakumar *et al.*, "{SENIC}: Scalable {NIC} for {End-Host} rate limiting," in *NSDI'14*.

[13] K. Jang, J. Sherry, H. Ballani, and T. Moncaster, "Silo: Predictable message latency in the cloud," in *ACM SIGCOMM'15*.

[14] Y. He, W. Wu, X. Wen, H. Li, and Y. Yang, "Scalable on-switch rate limiters for the cloud," in *IEEE INFOCOM'21*.

[15] V. S. Thapeta *et al.*, "Nimble: Scalable tcp-friendly programmable in-network rate-limiting," in *ACM SOSR'21*.

[16] N. McKeown *et al.*, "Openflow: enabling innovation in campus networks," *ACM SIGCOMM CCR*, vol. 38, no. 2, pp. 69–74, 2008.

[17] P. Bosshart *et al.*, "P4: Programming protocol-independent packet processors," *ACM SIGCOMM CCR*, vol. 44, no. 3, pp. 87–95, 2014.

[18] Y. Li, R. Miao, H. H. Liu *et al.*, "Hpcc: High precision congestion control," in *ACM SIGCOMM'19*.

[19] "Intel tofino series programmable ethernet switch asic," https://www.intel.com/content/www/us/en/products/network-io/programmable-ethernet-switch/tofino-series.html, 2022.

[20] D. a. Wischik, "Design, implementation and evaluation of congestion control for multipath {TCP}," in *NSDI'11*.

[21] C. Raiciu *et al.*, "Improving datacenter performance and robustness with multipath tcp," *ACM SIGCOMM CCR*, vol. 41, no. 4, pp. 266–277, 2011.

[22] S. Liu *et al.*, "Reducing traffic burstiness for mptcp in data center networks," *Elsevier JNCA*, vol. 192, p. 103169, 2021.

[23] E. Song *et al.*, "A cloud-scale per-flow backpressure system via fpga-based heavy hitter detection," in *ACM SIGCOMM'21 Posters*.

[24] D. Kim *et al.*, "Tea: Enabling state-intensive network functions on programmable switches," in *ACM SIGCOMM'20*.

[25] "Behavioral model (bmv2)," https://github.com/p4lang/behavioral-model, 2022.

[26] M. Al-Fares, S. Radhakrishnan, B. Raghavan *et al.*, "Hedera: dynamic flow scheduling for data center networks," in *NSDI'10*.

[27] W. Bai, L. Chen, K. Chen, D. Han, C. Tian, and H. Wang, "{Information-Agnostic} flow scheduling for commodity data centers," in *NSDI'15*.

[28] J. Zhao, J. Liu, H. Wang, and C. Xu, "Multipath tcp for datacenters: From energy efficiency perspective," in *IEEE INFOCOM'17*.

[29] A. Greenberg, J. R. Hamilton, N. Jain *et al.*, "Vl2: A scalable and flexible data center network," in *ACM SIGCOMM'09*.

[30] M. Zadnik and M. Canini, "Evolution of cache replacement policies to track heavy-hitter flows," in *ACM/IEEE ANCS'10 Posters*.

[31] J. Lu *et al.*, "A two-stage heavy hitter detection system based on cpu spikes at cloud-scale gateways," in *IEEE ICDCS'21*.

[32] A. Ford, C. Raiciu, M. Handley *et al.*, "Tcp extensions for multipath operation with multiple addresses," *RFC 6824*, 2013.

[33] M. Li *et al.*, "Multipath transmission for the internet: A survey," *IEEE COMST*, vol. 18, no. 4, pp. 2887–2925, 2016.

[34] A. Ford *et al.*, "Architectural guidelines for multipath tcp development," *IETF, Informational RFC*, vol. 6182, pp. 2070–1721, 2011.

[35] C. Raiciu, C. Pluntke, S. Barre *et al.*, "Data center networking with multipath tcp," in *ACM HotNets'10*.

[36] J. Sun *et al.*, "Dcˆ 2-mtcp: Light-weight coding for efficient multi-path transmission in data center network," in *IEEE IPDPS'17*.

[37] M. Kheirkhah and M. Lee, "Amp: An adaptive multipath tcp for data center networks," in *IFIP Networking'19*.

[38] M. Kheirkhah, I. Wakeman, and G. Parisis, "Mmptcp: A multipath transport protocol for data centers," in *IEEE INFOCOM'16*.

[39] Y. Tokusashi, H. T. Dang, F. Pedone *et al.*, "The case for in-network computing on demand," in *ACM EuroSys'19*.

[40] X. Jin, X. Li, H. Zhang, R. Soulé *et al.*, "Netcache: Balancing key-value stores with fast in-network caching," in *ACM SOSP'17*.

[41] X. Jin, X. Li, H. Zhang, N. Foster *et al.*, "{NetChain}:{Scale-Free}{Sub-RTT} coordination," in *NSDI'18*.

[42] C. Lao, Y. Le, K. Mahajan *et al.*, "{ATP}: In-network aggregation for multi-tenant learning," in *NSDI'21*.

[43] D. Eastlake and T. Hansen, "Us secure hash algorithms (sha and sha-based hmac and hkdf)," RFC 6234, May, Tech. Rep., 2011.

[44] B. Pit-Claudel, Y. Desmouceaux, P. Pfister *et al.*, "Stateless load-aware load balancing in p4," in *IEEE ICNP'18*.

[45] M. Ghasemi, T. Benson, and J. Rexford, "Dapper: Data plane performance diagnosis of tcp," in *ACM SOSR'17*.

[46] D. Shan, F. Ren, P. Cheng *et al.*, "Micro-burst in data centers: Observations, analysis, and mitigations," in *IEEE ICNP'18*.

[47] M. L. Fisher, R. Jaikumar, and L. N. Van Wassenhove, "A multiplier adjustment method for the generalized assignment problem," *Management science*, vol. 32, no. 9, pp. 1095–1103, 1986.

[48] M. R. Gary and D. S. Johnson, "Computers and intractability: A guide to the theory of np-completeness," 1979.

[49] M. Al-Fares *et al.*, "A scalable, commodity data center network architecture," *ACM SIGCOMM CCR*, vol. 38, no. 4, pp. 63–74, 2008.

[50] R. Ben Basat, S. Ramanathan, Y. Li *et al.*, "Pint: Probabilistic in-band network telemetry," in *ACM SIGCOMM'20*.

[51] "Gurobi optimization," http://www.gurobi.com, 2022.