# RLPlacer: A Deployment Model for Distributed Rate Limiting

Paper #516 (1570576181)

***Abstract*—Providing guaranteed performance and satisfying the Service Level Agreements (SLAs) are critical to the business and prosperity of any cloud provider. In a multi-tenant environment, rate limiting is an indispensable and widely used method to provide bandwidth and performance guarantees. Leveraging the rate limiting functionality on the programmable network devices is increasingly popular due to its scalability and efficiency. However, the resources for rate limiting on the network devices are limited and thus needs to be shared cross flows. A careful rate limiter allocation can achieve much more efficiency network wide. This paper presents RLPlacer, a system that optimizes the deployment of rate limiters in the cloud to support all flows with limited number of shared rate limiters on programmable switches. Our experiment on real topologies and flow datasets shows a factor of 5 reduction in terms of resource needed while achieving nearly the same performance as the per-flow rate limiting method.**

## I. Introduction

The network infrastructure in the modern cloud environment is shared cross multiple tenants and their applications. Providing bandwidth guarantees to cloud applications is critical to ensure the predictable performance and thus satisfy the Service Level Agreements (SLAs). Supporting network virtualization with bandwidth guarantees has become an indispensable feature for public cloud providers. Over the last decade, a large body of research has been put on abstractions, programming models, controllers, and enforcement mechanisms to achieve the goal of efficient bandwidth guarantees.

Regardless of how the bandwidth quota is computed, rate limiter is the main enforcement mechanism to shape the flows' rate according to the available bandwidth and ensure performance isolation. Recently, researchers have been improving rate limiter performance from different perspectives. Some focus on rate limiter performance in terms of reducing the per-packet queuing delay [1], [2] or minimizing the flow completion time [3], [4]. Some focus on improving the fairness of bandwidth allocation [5], [6] and the network utilization [7]. In most cases, the bandwidth is computed and enforced in a distributed manner [8]–[13].

A common rate limiter solution is to implement the rate limiting function in the end host, with software-based [14]–[16] or NIC-based [17] mechanism. However, it may introduce additional latency [2] and CPU overhead [17] to the host, which can be even worse with higher network speed. Most of the end host approaches apply the per-class or per-user based rate limiting. But finer-grained control (i.e., at the flow level) is needed for more efficient resource allocation. Therefore, recently there are proposals on building rate limiters on the network switch itself [10], [18]. *Making the network as a logical giant rate limier* can ease the programming complexity and improve the overall efficiency.

One major challenge for the network based rate limiter is that commodity switches can only support at most 8 classes and are not easy to be reprogrammed. With the emergence of programmable switches [19]–[21] and data plane programming language, e.g., P4 [22], it is possible to implement the rate limiting functions on the programmable switches [23]–[26]. For example, Tofino [19] provides programmable `Meter` which can be used for bandwidth allocation.

While network based rate limiting is promising, one unsolved problem is how to allocate the network wide resources across all flows. Each switch has limited resources (TCAM and SRAM) to hold all the rules for all flows. We need a mechanism to distribute the per-flow rate limiter to all switches in the network.

We present RLPlacer to solve the cloud rate limiters placement and allocation problem. There are two main ideas of this work: (1) an intelligent placement of rate limiters to maximize the amount of flows that it can cover, and (2) a method to multiplex rate limiter within the switch to support multiple flows. On the one hand, we first prove that cloud rate limiters placement problem is a NP-Complete problem by formulating the problem as an Integer Linear Programming (ILP) problem. We then develop an algorithm to place rate limiters in an incremental manner without disruptively changing the previous assignment. On the other hand, the number of flows is far more than the number of meters available on the switch [17], thus multiplexing is needed. We propose the hypothesis that it is possible to share the same rate limiter across multiple flows due to the nature of flow size and duration distribution. We first verify this hypothesis using the flow data from real world. We formulate the sharing using a theoretical formulation. Specifically, assume there are $x$ flows sharing the same pre-configured parameter *rate*, RLPlacer allocate $y$ ($<< x$) rate limiters to serve for these flows to guarantee the least bandwidth saturation, e.g., $\geq 98\%$. Even so, conflict may still occur. Then we propose to use SDN based approach to resolve conflicts and dynamically assign rate limiters.

The main contributions of this paper are as follows.

- This is the first work that proposes the multiplexing of rate limiters in the network cross multiple flows. We propose the hypothesis that multiplexing is possible, formulate the multiplexing as a probabilistic model, and verify it using real world flow data.
- We are the first work to propose the rate limiter placement and flow assignment problem. We formulate the problem

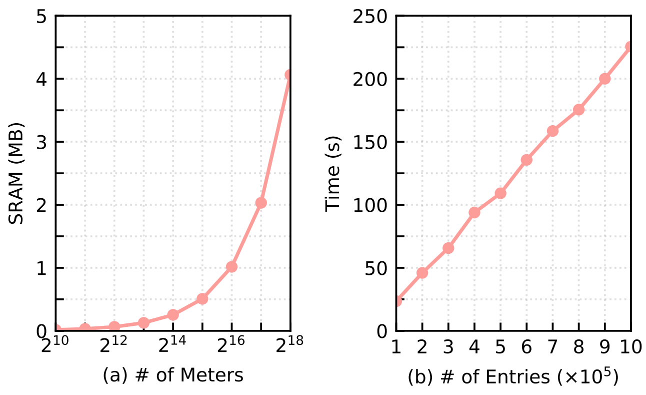

Fig. 1. Memory usage and Entry loading time of P4 switch

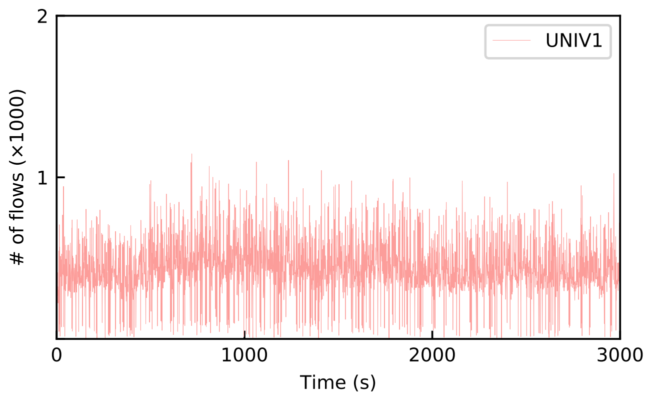

Fig. 2. Flow number varying with time, statistics time interval = 1s

as the integer linear programming and prove it is an NP-Complete problem. We then develop an algorithm to find the approximate solution. To consider real world deployment scenario, our algorithm assigns rate limter incrementally.

- We develop RLPlacer, the first system for the deployment and multiplexing of rate limiters in the cloud, which makes distributed bandwidth allocation system more scalable and improves the utilization of rate limiters to support more flows.
- We evaluate RLPlacer on typical data center topology with real packet traces and prove RLPlacer has high performance in distributed per-flow rate limiting.

## II. Overview of RLPlacer

In this section, we show the motivation for RLPlacer, including two parts for the needs for the deployment of rate limiters in the cloud, then followed by the design of RLPlacer.

### A. Motivations

**Need for rate limiters placement.** Suppose one rate limiting rule is associated with each flow, then rate limiters placement is equal to distributed these rules to network devices. Firstly, the scale of rate limiting rules would be so large in the cloud that only distributed bandwidth allocation can support. To develop a bandwidth allocation system that can serve for a data center, there will be more than billions of configuration rules in the system. Distributing these rules among networks nodes is the simplest way to balance the load of network devices. There are two approaches to implement distributed bandwidth allocation, one is to enforce rate limiting at each end host and configure them individually, the other is to do it with switches. Although the former is more prevalent, programmable switches make it a better place to enforce rate limiting at switches, which is more flexible, high performance to handle high speed flows. Here we focus on the latter implementation. Besides, One flow only traverses through a subset of all switches and there may be no point of intersection between two paths. What's more, the available resource such as SRAM and TCAM of one node is limited, and one flow's rate can be controlled when enforcing rate limiting only at one node of its path. For these reasons, there is a need for the algorithm of distributed deployment of rate limiters.

**Need for rate limiters multiplexing.** Ideally, to achieve per-flow rate limiting, it needs to assign a dedicated rate limiter to each flow. However, based on the following observations, we argue that this is not the only approach to per-flow rate limiting. Firstly, the number of rate limiters supported by one switch is limited. Take the `P4 Meter` of Tofino [19] for example, which is the core of a rate limiter when develop rate limiter at programmable switches, Figure 1(a) depicts the storage usage with the increase of Meters. Besides, due to the fact that in the PISA architecture of P4, one Match Action Unite only provides a small number of Meters, too many rate limiters will increase the complexity of scheduling rate limiters among Match Action Units. Secondly, though there are about 50 MB SRAM available in Tofino, only a fraction of them are left for developing rate limiters. Thirdly, the probability of occurrence of all flows at the same time is very low, there is no need assigning each flow a dedicated rate limiter. For instance, we analyze four public datasets [27]–[30], Table II-A shows the results, there are 556601 and 1091248 flows in UNIV1 and DEFCON respectively, whereas the max occurrence probability of flows at the same time within each second is less than $0.2059\%$ and $1.0152\%$ respectively. A dedicated rate limiter for each flow is uneconomic for it being idle during most of time.

TABLE I
Max Occurrence Probability of flows at the same time

| Dataset | Flow Number | Peak (%) | |
|---|---|---|---|
| | | Interval=10ms | Interval=1s |
| UNIV1 | 556601 | 0.1653 | 0.2059 |
| FIRST | 445618 | 0.0873 | 0.1093 |
| MACCDC | 34333734 | 0.0553 | 0.1091 |
| DEFCON | 1091248 | 0.9408 | 1.0152 |

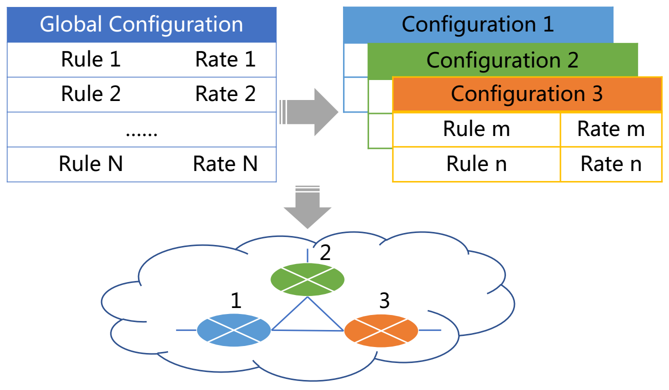

Fig. 3. Rate limiters placement

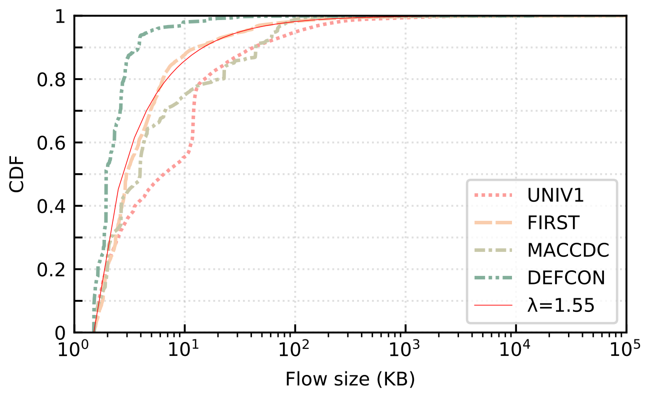

Fig. 5. The statistical results of flow length of different datasets

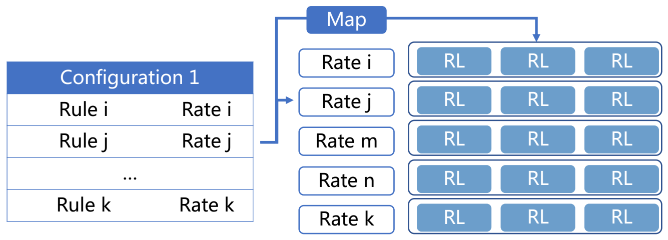

Fig. 4. Overview of rate limiters multiplexing

## B. *Design of RLPlacer*

The design of RLPlacer is decomposed into two parts, one is rate limiters placement, the other is rate limiters multiplexing. The main idea of RLPlacer is dividing the whole configuration rule space into many non-intersect subspaces, which then being handled by one switch.

Figure 3 depicts the architecture of rate limiters placement. The whole configuration space is abstracted as a global configuration table, of which each entry represents a flow and its corresponding rate, e.g, each entry may has the format: $< srcIP, dstIP, rate >$. For an individual flow, it only needs to enforce rate limiting for the flow at one point in its path so that the throughput would be controlled. Thus, each entry of the Global Configuration is only needed to be owned by one switch. To this end, we divide Global Configuration into many disjoint subconfigurations. Next, each subconfiguration is populated into a switch. All switches work together to enforce the distributed rate limiting.

Inside each switch, RLPlacer adopts multiplexing to reduce the rate limiters number. This idea is based on two observations. In practice, the range of the rate of rate limiting is discrete and limited, and many flows share the same rate. Moreover, the number of flows appearing in a tiny time interval is much smaller the total. Take dataset *UNIV1* for example, as illustrated in Figure 2, the concurrent flows are nearly no more than 1,000 per second, though there are 556601 flow in it. This indicates that the most of rate limiters are idle if each flow has one rate limiter. We propose a simple but effective mechanism to improve the utilization of rate limiters while achieving the goal that each flow seems to have a dedicated rate limiter. As shown in Figure 4, each switch has a configuration table and maintains an array of rate limiters. Suppose there are $x$ flows in the configuration table sharing the same parameter $rate\ j$, RLPlacer would allocate $y$ rate limiters for them. At the arrival of one of these flows, we first obtain which set of rate limiters it should through based on its $rate$, then map this flow to one of rate limiters. A more effective method may be selecting a idle rate limiter for a new flow dynamically, but for simplicity, here we hash one flow to a rate limiter according to its five tuple.

## III. Problem Formulation

Before revealing the detail of RLPlacer, here we define the rate limiters placement problem and rate limiters multiplexing problem formally. Our theoretical analysis proves that rate limiters placement is at least a NP-Complete problem and there is a unique solution for rate limiters multiplexing problem.

### A. *Preliminaries*

**Rate Limiters Placement.** Rate limiters placement means the process that how to distribute rate limiting rules associated with each flow to network nodes. Consider a network topology with a set of nodes connected by undirected links. These nodes represent either the physical or virtual switches $S$. Suppose each entry in the Global Configuration denotes a flow $< srcIP, dstIP, rate >$. There would be many flows passing through the network from time to time. Since not all the flows need to be limited, we don't consider the free flows that are not in the limiting targets here. Thus, we see the maximum bandwidth capacity for each link as the difference of real capacity and the occupied bandwidth by the free flows. Consider the limited flows as a set $F$. For each flow $f_i \in F$, suppose there are $K$ corresponding paths $p_i = \{p_{i1}, \ldots, p_{ij}, \cdots, p_{iK}\}$ and a limited flow rate $r_i$. From the topology, we can derive the binary variables $y_{ijm}$ and $z_{ijn}$ denoting whether path $p_{ij}$ traverses the link $l_m \in L$ and whether $p_{ij}$ passes by the switch $s_n \in S$ respectively. Also, binary variable $g_{in}$ denotes if the rate limiter of flow

$f_i$ is placed on switch $s_n$. Table II lists all the variables and constants for the model.

**Rate Limiters Multiplexing.** We propose rate limiters multiplexing to achieve per-flow rate limiting with as few rate limiters as possible. The key idea of rate limiters multiplexing is to multiplex rate limiters for flows in time given the fact that most of flows are non-overlapping in time. We give an example of flow's appearance in Figure 6 based on the analysis from the four public datasets, the vertical line represents a packet and we can see that at any time there only a fraction of flows are active, which means that if more rate limiters than the maximum number of concurrent flows are given, we can find a scheduling policy for per-flow rate limiting with these rate limiters. Formally, suppose there are $x$ flows in total in the rate limiting configuration of one switch, to calculate the minimum number (denoted as $y$) of rate limiters needed to achieve higher than the $e$ (i.e. 98%) bandwidth saturation, we give the following assumptions based on the observations from the statistics results of the four public datasets.

- We depict the flow length of four public datasets, the results are shown in Figure 5, which can be fitted by

$$F_s(x) = 1 - e^{-1.55 \times \lg(x+0.5)} \tag{1}$$

Therefore, we assume that the distribution probability of flow size (in packets) $P(S = i) = F_s(i) - F_s(i-1)$, where

$$F_s(i) = \begin{cases} 0 & ,i < 0 \\ 1 - e^{-\lambda \lg(i+1)} & ,i \geq 0 \end{cases} \tag{2}$$

Then, the expected flow size

$$E(s) = \lim_{n \to \infty} (n - \sum_{i=1}^{n-1} F_s(i)) \tag{3}$$

- At any given moment $t$, the occurrence probability of flow $f_i$ is $p_i(t)$ and thus only a subset of all flows will occur at the same time. Then the distribution of time $t$ ($t \geq 0$) that has elapsed before the flow becoming active is stated as

$$F_n(t) = e^{-\int p_i(t)dt}(\int p_i(t)e^{\int p_i(t)dt}dt - 1) \tag{4}$$

Further more, if the arrival of packets can be described by a Poission Process ($\lambda_i$) when the flow is active [31], [32], the distribution of the time intervals $t$ ($\geq 0$) between two successive packets is given by

$$F_T^i(t) = \frac{\frac{1-e^{-\lambda_i t}}{\lambda_i}E(s) + F_n(t)\int tF_n'(t)dt}{\frac{1}{\lambda_i}E(s) + \int tF_n'(t)dt} \tag{5}$$

- Suppose $x$ flows with $n$ different *rate* need $y$ rate limiters to achieve at least $e$ expected bandwidth saturation for all flows, obviously $y = h^e(x, n)$ is a monotone-nondecreasing function. Let $X(= X_1, \cdots, X_x)$ denotes the total number of configuration entries on one switch,

$$h^e(|X|, 1) \leq h^e(|X|, n) = \sum_{i=1}^{x} h^e(X_i, 1) \leq |X| \tag{6}$$

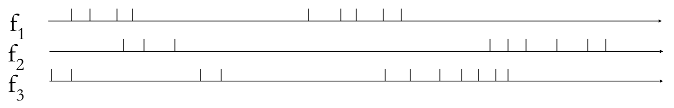

Fig. 6. Packets' arrival time of different flows

## *B. Theoretical Analysis*

**Lemma 1.** Rate limiters placement problem is NP-Complete.

**Proof.** This can be proven by formulated the placement problem as an Integer Linear Programming (ILP) problem. To find the best solution to the rate limiter placement problem, we consider the maximum occupied memory on all switches as the objective.

$$\textbf{\textit{Objective:}} \quad \min \max_{s_n \in S}(\sum_{i=1}^{|F|}\sum_{j=1}^{K} x_{ij}z_{ijn}g_{in}) \tag{7}$$

**Capacity Constraints.** For each link and switch, the occupied bandwidth cannot exceed its physical limit. Note that the physical limit here refers to the difference between the real cable capacity and the occupied bandwidth by the free flows. A formal expression is as follows:

$$\begin{gathered} \forall\ l_m \in L,\ \sum_{i=1}^{|F|} r_i \sum_{j=1}^{K} x_{ij}y_{ijm} \leq C_m \\ \forall\ s_n \in S,\ \sum_{i=1}^{|F|} r_i \sum_{j=1}^{K} x_{ij}z_{ijn} \leq C_n \end{gathered} \tag{8}$$

**Memory Constraints.** The insufficient number of memory on switches is the key problem.

$$\forall\ s_n \in S,\ \sum_{i=1}^{|F|}\sum_{j=1}^{K} x_{ij}z_{ijn}g_{in} \leq M_n \tag{9}$$

**Feasibility Constraints.**

1. A necessary condition is that each flow has a path

$$\forall\ f_i \in F,\ \sum_{j=1}^{K} x_{ij} = 1 \tag{10}$$

2. The rate limiter of one flow only needs to be placed in one switch.

$$\forall\ f_i \in F,\ \sum_{n=1}^{|S|} g_{in} = 1. \tag{11}$$

3.Besides, it has to make sure that the limiter is on the determined path for each flow

$$\forall\ f_i \in F\ and\ j \in [1, K],\ \sum_{n=1}^{|S|} g_{in}z_{ijn} = x_{ij} \tag{12}$$

Note that when $K$ is set to be 1, these constraints build up an ILP. Because ILP is NP-Complete, the optimal solution to rate limiter placement is at least NP-Complete and can't be solved in polynomial time in theory.

TABLE II
SYMBOLS AND NOTIONS FOR RATE LIMITER PLACEMENT PROBLEM

| Type | Symbol | Meaning |
|---|---|---|
| Inputs | $S$ | The set of switches, $S = \{s_1, s_2, \cdots, s_n, \cdots\}$ |
| | $L$ | The set of links, $L = \{l_1, l_2, \cdots, l_m, \cdots\}$ |
| | $F$ | The set of flows, $F = \{f_1, f_2, \cdots, f_i, \cdots\}$ |
| | $R$ | The set of pre-configure rates, flow $f_i$ has rate $r_i$, $R = \{r_1, r_2, \cdots, r_i, \cdots\}$ |
| | $K$ | Each flow has $K$ paths, e.g., $f_i$ has paths $\{p_{i1}, p_{i2}, \cdot, p_{ij}, \cdots\}$ |
| | $y_{ijm}$ | A binary value to indicate if the path $p_{ij}$ traverse through link $l_m$ |
| | $z_{ijn}$ | A binary value to indicate whether the path $p_{ij}$ goes through switch $s_n$ |
| Constants | $C_m$ | The bandwidth capacity for link $l_m$ |
| | $C_n$ | The bandwidth capacity for switch $s_n$ |
| | $M_n$ | The memory capacity on switch $s_n$ |
| Variables | $x_{ij}$ | A binary variable to indicate whether the flow $f_i$ would choose path $p_{ij}$ |
| | $g_{in}$ | A binary variable to indicate whether the rate limiting entry for flow $f_i$ would be placed on switch $s_n$ |

**Lemma 2.** For $x$ flows sharing different rates, at least $y$ rate limiters are needed to achieve $e$ bandwidth saturation.

$$e = \frac{1 - g(x,y)}{\mu} \cdot t \int_0^{\infty} F_T^i(t)dt \tag{13}$$

where $\mu$ is the average processing time of each packet and $\sigma = g(x,y) \in (0,1)$ is the unique solution to $\sigma = (\frac{1}{\sigma} - 1)H(\frac{1}{\sigma})$.

**Proof.** Assume $y$ independent and identical rate limiters are allocated to $x$ flows that share different parameters *rate* and each flow is randomly mapped to one of the rate limiters, thus each rate limiter is expected to serve for $\lceil \frac{x}{y} \rceil$ flows. Suppose the packets' arrival event of one flow is a steady-state process, the stochastic process of $\lceil \frac{x}{y} \rceil$ flows traversing through the same rate limiter has a steady state. For the $i$th flow, let $F_T^i(t)$ be the distribution of the time intervals between two successive packets. Choose one rate limiter as research object, the distribution of time intervals between two successive packets for the rate limiter is

$$F_A(t) = 1 - \prod_{i=1}^{\lceil \frac{x}{y} \rceil} (1 - F_T^i(t)) \tag{14}$$

where $F_T^i(t), i = 1, \cdots, \lceil \frac{x}{y} \rceil$ are independent. At the arrival of $k$th packet, suppose there are $L_k$ packets in the buffer, when the system reaches the steady state, the probability of $n$ packets in buffer is

$$a_n = \lim_{k \to \infty} P(L_k = n) \tag{15}$$

During the time between $k$th and $k+1$th packet, if $D_{k+1}$ are processed by the rate limiter, we can obtain that

$$L_{k+1} = L_k + 1 - D_{k+1} \tag{16}$$

Then $\{L_k\}_{k=0}^{\infty}$ is a Markov chain and the transition matrix is given by

$$P = \begin{bmatrix} p_{0,0} & \beta_0 & 0 & \cdots & \cdots \\ p_{1,0} & \beta_1 & \beta_0 & 0 & \cdots \\ p_{2,0} & \beta_2 & \beta_1 & \beta_0 & 0 \\ p_{3,0} & \beta_3 & \beta_2 & \beta_1 & \beta_0 \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{bmatrix}$$

where

$$\begin{aligned} \beta_i &= \int_{\mu i}^{\mu(i+1)} f_A(t)dt = F_A(\mu(i+1)) - F_A(\mu i) \\ p_{i,0} &= 1 - \sum_{j=0}^{i} \beta_j = \sum_{j=i+1}^{\infty} \beta_j \end{aligned} \tag{17}$$

and the distribution of $\{a_n\}_{n=0}^{\infty}$ is constrained by

$$\begin{aligned} a_0 &= \sum_{i=0}^{\infty} a_i p_{i,0} \qquad , n = 0 \\ a_n &= \sum_{i=0}^{\infty} a_{n-1+i} \beta_i \quad , n > 0 \end{aligned} \tag{18}$$

Assume $a_n = A\sigma^n$ ($\sigma \in (0,1)$), $H(z) = \sum_{n=-\infty}^{+\infty} h[n]z^{-n}$ is the $Z$-transform of $F_A(t)$ when the sampling period $T = \mu$, from Equation 18 we can obtain

$$\begin{aligned} \sigma &= \frac{1}{\sigma} \sum_{i=1}^{\infty} \sigma^i F_A(\mu i) - \sum_{i=0}^{\infty} \sigma^i F_A(\mu i) \\ &= (\frac{1}{\sigma} - 1)H(\frac{1}{\sigma}) \end{aligned} \tag{19}$$

Next, we show above equation has unique solution when $\sigma \in (0,1)$. Let

$$\begin{aligned} f(\sigma) &= H(\frac{1}{\sigma}) - \frac{\sigma^2}{1-\sigma} \\ &= F_A(\mu)\sigma + \sum_{i=2}^{\infty} (F_A(\mu i) - 1)\sigma^i \end{aligned} \tag{20}$$

Then the following equation holds when $\sigma \in (0, \frac{F_A(\mu)}{1+F_A(\mu)})$

$$f(\sigma) > F_A(\mu)\sigma - \sum_{i=2}^{\infty} \sigma^i = \sigma(F_A(\mu) - \frac{\sigma}{1-\sigma}) > 0 \tag{21}$$

Besides

$$\lim_{\sigma \to 1} f(\sigma) = F_A(\mu) + \sum_{i=2}^{\infty} (F_A(\mu i) - 1) \leq F_A(\mu) + F_A(2\mu) - 1 \tag{22}$$

$F_A(\mu) + F_A(2\mu) > 1$ indicates the average arrival time of packets is less than $\mu$, which would lead to the unlimited growth of queue. Based on our assumption that the system has a steady state, this situation does not happen. Thus, there must be at least one solution to Equation (19) according to mean value theorem. Obviously, $f'(\sigma)$ is nonotone decreasing and $f'(0) > 0$ and $f'(1) < 0$, so there exists $\beta \in (0,1)$ to satisfy $f'(\sigma) = 0$. When $\sigma \in (0,\beta)$, $f(\sigma)$ is monotone increasing and $f(\beta) > f(0) > 0$, and when $\sigma \in (\beta,1)$, $f(\sigma)$ is monotone decreasing. Given that $f(1) < 0$, Equation (19) has the unique solution in $(\beta, 1)$.

Based on above conclusions, $\sigma$ is only decided by input $x$ and $y$, so there exists on equation holding $\sigma$=g(x,y). Because of the standard of probability, we have $A = 1 - \sigma$ and thus

$$a_n = (1-\sigma)\sigma^n \tag{23}$$

Due to the system has a steady state, the expected processing time of each packet is in proportion to the expected processing time of the flow, then the throughput percentage $e$ of the $i$th flow getted from the rate limiter is given by

$$\begin{aligned} e &= \frac{E(\ arrival\ time\ interval\ of\ ith\ flow)}{E(\ processing\ time)} \\ &= \frac{1-g(x,y)}{\mu} \cdot t \int_0^\infty F_T^i(t)dt \end{aligned} \tag{24}$$

The above proving process indicates that once we learned a minimum $y$ at any given $x$ and $e$ in practice, we can directly apply the ratio of $x$ and $y$ in the same environment.

**Algorithm 1:** Greedy Rate Limiters Placement

```
1  while $F$ != $nil$ do
2      start:
3      $f_i \leftarrow F$.getAndRemoveEntry()
4      sortByLength($\{p_{ij}\}$, ascending)
5      for $j$ in $1 \cdots K$ do
6          $s_n \leftarrow p_{ij}$.head()
7          while $s_n$ != $nil$ do
8              $s_n$.populateEntry($f_i$)
9              if Not violateConstraints() then
10                 goto start
11             $s_n$.removeEntry($f_i$)
12             $s_n \leftarrow p_{ij}$.next($s_n$)
13 for $s_n \in S$ do
14     if $s_n$.entryNum() $> \frac{|F|}{|S|}$ then
15         for $p_{ij}$ rate limited on $s_n$ do
16             $s_{n'} \leftarrow$ minEntryNum($p_{ij}$)
17             if $s_{n'}$.entryNum() $< \frac{|F|}{|S|}$ then
18                 $s_n$.removeEntry($f_i$)
19                 $s_{n'}$.populateEntry($f_i$)
20             if $s_n$.entryNum() $\leq \frac{|F|}{|S|}$ then
21                 break
```

## IV. Algorithm

In this section, we give two algorithms to address rate limiters placement problem. There exist many algorithms to solve ILP, thus when $K$ is set to be 1, rate limiters placement can be solved easily. But when more than one paths are associated with a flow, as well as taking the topology of data center into consideration, there is a need for new algorithms to rate limiters placement problem. Here we propose two heuristic algorithm for the problem.

**Algorithm 2:** Incremental Rate Limiters Placement

```
1  for $j$ in $1 \cdots K$ do
2      $s_n \leftarrow p_{ij}$.head()
3      while $s_n$ != $nil$ do
4          $s_n$.populateEntry($f_i$)
5          if Not violateConstraints() then
6              $s_{n'} \leftarrow p_{ij}$.head()
7              $P_{in} = s_{n'}$.memory
8              while $s_{n'}$ != $nil$ do
9                  if $s_{n'}$.memory $< P_{in}$ then
10                     $P_{in} = s_{n'}$.memory
11                 $s_{n'} \leftarrow p_{ij}$.next()
12         $s_n$.removeEntry($f_i$)
13         $s_n \leftarrow p_{ij}$.next()
14 n $\leftarrow$ min ($P_{in}$)
15 $s_n$.populateEntry($f_i$)
```

### A. Greedy Deployment

First, we begin with an intuitive understanding algorithm. In data center, the topology does not have complex structure, thus making rate limiters placement easier to be solved than general one that defined on any graph. Algorithm 1 shows the logic of placing rate limiting rules greedily. The algorithm consists of two parts, position selection (line 1 — 12) and rebalance operations (line 13 — 21). Intuitively, we get entry one by one from the set of flows $F$ and find the first suitable place for the flow to populate the corresponding rate limiting rule. The criterion to judge whether a node is suitable to enforce rate limiting for the flow is the objective must get minimum value when it is selected and the selected path must be as short as possible. Specifically, we enumerate all paths of the flow by length in ascending order and then find the first node that do not violated the constraints while minimizing the objective. On the other hand, after finding a feasible solution the rate limiters placement problem, it is necessary to rebalance the distribution of rate limiting rules for that the memory usage of each node may vary significantly. Moreover, Figure 1(b) shows that populating entries is a time-consuming process, thus populating two many entries would waste too much time. Therefore, we adopt rebalance operations to balance the load of each node. To this end, we migrate rate limiting rules among nodes to balance the distribution. Before the migration, we set

$\frac{|F|}{|S|}$ as threshold to decide whether a rule can be populated into one node or should be removed from one node. The reason of choosing $\frac{|F|}{|S|}$ is each node would acquire the same amount of rate limiting rules under ideal circumstances. For each node that owns over $\frac{|F|}{|S|}$ rate limiting rules, we try to find another node , of which the total number of rate limiting rules is less than $\frac{|F|}{|S|}$, that does not violate constraints for each flow rate limited at the node. The migrating process repeats until no rate limiting rule can be migrated or the memory usage of the node has less than or equal to $\frac{|F|}{|S|}$. By the two phases of rate limiters placement, the placement problem can achieve a much closer to global optimal one.

### B. Incremental Deployment

While Algorithm 1 is more intuitive and easier to be implemented, it is not practical to adjust the distribution of rate limiting rules continuously when new rules are added. We propose to place rate limiting rules incrementally to solve the problem. The incremental rate limiters placement mechanism is shown in Algorithm 2. When a new flow needs to be rate limited, for each path associated with the flow, we enumerate the node of the path and verify whether the constraints would be violated if rate limited at the node. If not, record the node and its memory usage state and then verify the next node. Compared to Algorithm 1, the incremental algorithm always find the node with the lowest memory usage rather than the optimal position of the first feasible path (Algorithm 1). Moreover, once choosing a node to enforce rate limiting for a flow, it won't be changed forever.

## V. Evaluation

We evaluate RLPlacer in this section. For the part of rate limiters placement, we simulate our algorithms on two typical data center topologies. Specifically, we compare our algorithms with solving the ILP by python library *PULP* when only one path is associated with each flow. To evaluate the performance of rate limiting multiplexing, we use the four aforementioned datasets to measure the mean bandwidth saturation of all flows.

### A. Implementation

We create two virtual data center topologies, 4-post and 5-stage, following [33], of which has 5 clusters, 124 switches, 1000 hosts, 1504 edges and 5 clusters, 205 switches, 1000 hosts, 3025 edges respectively. We generate host pairs (10,000 $\sim$ 50,000) randomly between hosts, the ration of inter-cluster host pair and inner-cluster host pair are 2:8 and 8:2. We prototype RLPlacer with python and carry out experiments on a 64-bit server with 3.30GHz CPU, 64 GB RAM, 10 cores and 20 logical cores. Our evaluation is carried out with four public data traces [27]–[30]

### B. Rate Limiter Placement

Firstly, we assign each flow only one path to compare our algorithm with python library PULP, which is usually used to solve ILP problem. Figure 7 depicts the results, from which we can get the following observations: (1) The objectives defined

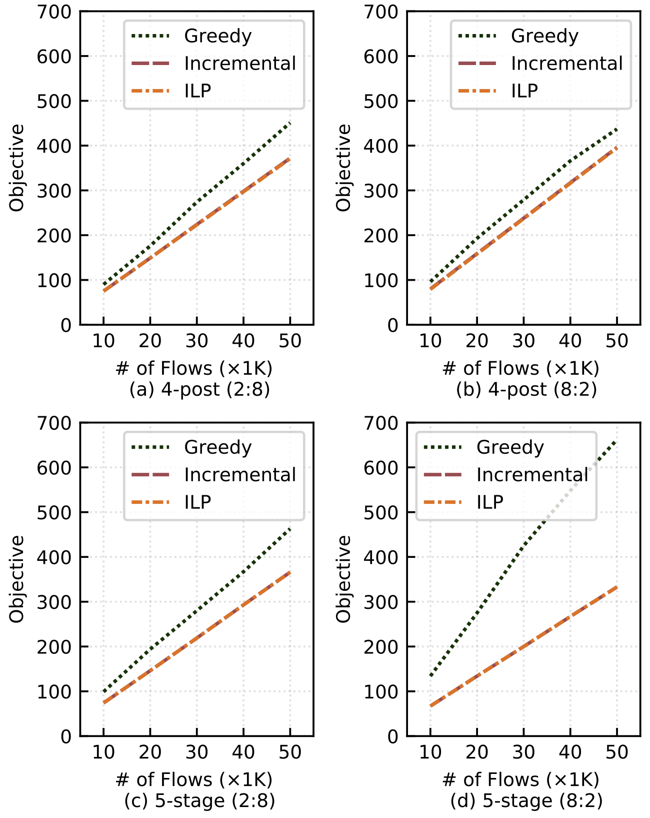

Fig. 7. Comparison of Algorithm 1 and Algorithm 2 with PULP (Objective = $\min \max_{s_n \in S}(\sum_{i=1}^{|F|}\sum_{j=1}^{K} x_{ij} z_{ijn} g_{in})$)

in Equation 7 of *Greedy Placement*, *Incremental Placement* and *PULP* increase linearly with the number of flows as expected, which means that the distribution of rate limiting rules is evenly with the flows number; (2) Different data center topologies have little effect on the final objective; (3) Placing rate limiters incrementally can get nearly as good performance as optimal one under all circumstances, e.g., the objectives of Incremental and PULP are 372 and 371 respectively in Figure 7(a); (4) Incremental strategy performs better than greedy strategy by at least 20%.

Further more, we explore the effect of the number of each flows' paths on the distribution of rate limiters rules. We repeat all the above experiments by setting flows number to be 10,000 and then vary the paths number from 1 to 10. The results are shown in Figure 8. When scaling to multipath scenario, the performance of greedy strategy is unstable and thus unpredictable. By contrast, no matter what the number of paths is, incremental strategy achieve low and steady objective on both topology, which indicates the load of each node is more balance.

### C. Rate Limiter Multiplexing

We use two metrics, the mean bandwidth saturation of all flows and the percentage of flows that have achieved expected

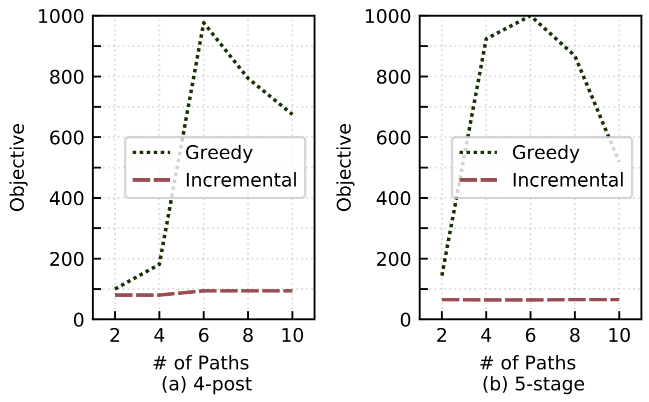

Fig. 8. The effect of paths number on the objective of Algorithm 1 and Algorithm 2 (Objective = $\min \max_{s_n \in S}(\sum_{i=1}^{|F|}\sum_{j=1}^{K} x_{ij} z_{ijn} g_{in})$)

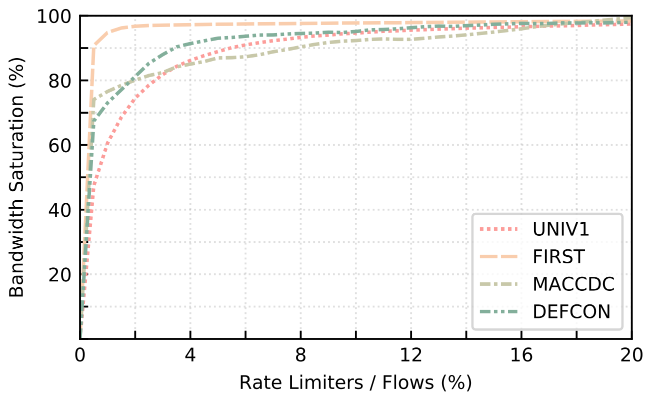

Fig. 9. Mean bandwidth saturation of all flows

throughput, to measure the performance of RLPlacer in rate limiters multiplexing.

Figure 9 shows the mean bandwidth saturation of all flows in rate limiters multiplexing experiments. We fix the total number of flows and then vary rate limiters number from 0 to 20% of flows number. The mean throughput increases with the increase of the number of rate limiters. When the ratio of flows number and rate limiters number is 5:1, the mean bandwidth saturation can achieve up to 97.47%, 98.39%, 99.43%, 98.03% respectively on *UNIV1*, *FIRST*, *MACCDC* and *DEFCON*.

We give the bandwidth saturation of each flow in Figure 10. There are 95.05%, 96.47%, 90.18%, 84.10% of flows in *UNIV1*, *FIRST*, *MACCDC* and *DEFCON* achieve exactly expected throughput. The results mean that for these flows, they has the rate limiter all to themselves during the time period when they are active. But they are still some flows can not achieve the expected throughput. We analyze the results and find the root cause is that these flows are very small, which means they are sensitive to queuing latency. For example, of all the flows in *DEFCON* that cannot achieve the expected throughput, more than 90% of flows are less than 3.874 KB. There are two approaches can overcome the issue. The simplest way is to develop a per-flow rate limiting system, which is not feasible currently in that either the number of rate limiters is constrained [17], [34] or would bring additional overhead like high latency [2], [35]. Instead of mapping flows to rate limiters statically, a dynamic mapping scheme can be adopted based on the techniques such as SDN or machine learing.

## VI. Related work

We categorize current bandwidth allocation work into two classes, one aims at improving bandwidth allocation by optimizing the implementation at end hosts, while the other focus on improving bandwidth allocation scheme of the entire data center to achieve objectives such as fairness, convergence, and etc, which is implemented mainly in the form of distributed rate limiting.

**End hosts rate limiting.** The optimization of rate limiters at end hosts is achieved by two approaches as follows:

- **software-based rate limiter.** Most of work enforces rate limiting by software-based rate limiter. Based on Linux Qdisc, [36], [37] can achieve per-class or per-user rate limiting by assigning users with different priorities and then applying corresponding queue disciplines to them. But due to the high CPU and performance overhead of traditional software rate limiters, [15] proposes to dequeue packets using time and bind shapers to each CPU core in traffic shaping to lower CPU overhead and avoid head of line blocking while supporting up to tens of thousands of traffic classes per server. Further more, [2], [3] optimize software-based rate limiters to reduce queuing delay and flow completion time by lowering the queue length and even just jumping queues.
- **hardware-based rate limiter.** Another important way to implement bandwidth allocation is by hardware. Given that the control is not complex in bandwidth allocation, thus it can be offloaded in programmable switch [38]. One of the restrictions of hardware rate limiters is its limited number of rate limiters, [17] makes it possible to provide more than 10,000 rate limiters by one NIC for the first time.

**Distributed rate limiting.** Referring rate limiters as Black-Boxes, some other studies focus on the bandwidth allocation scheme in the entire data center. Naturally, it is implemented as distributed bandwidth allocation, which can be realized by coordinating either end hosts or switches.

- **End hosts based bandwidth allocation.** Currently, the most prevalent method to achieve distributed bandwidth allocation is to configure rate limiting rules at each end host to serve as a global single rate limiter [8], [9], [14], [15]. These work targets to support a larger scale of network and guarantee the fairness and convergence of flows.
- **Switch-based bandwidth allocation.** All the time, switches play a key role in bandwidth allocation [15], [18], [22], [39]. [10] proposes a distributed rate limiter

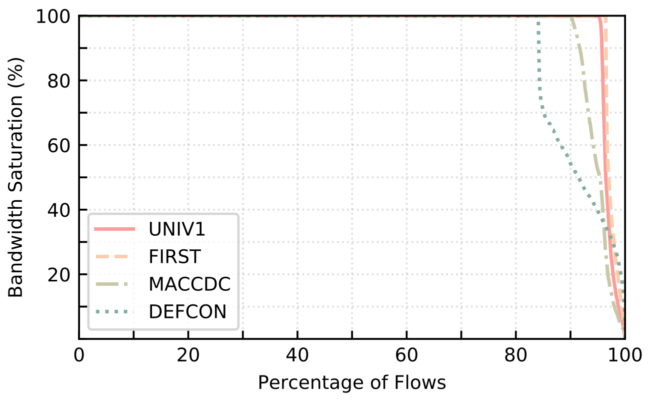

Fig. 10. Cumulative distribution function of each flows' throughput

with switches to enforce global rate limiting. Recently, [22], [40] provide flexible and scalable `Meter` in programmable switches to achieve bandwidth allocation.

## VII. Discussion

In practical deployment, the rate limiter placement in RLPlacer can be enforced globally for all tenants; the rate limiter multiplexing can be enabled according to the tenant QoS requirement and payment. If the tenant is not strict on the QoS (e.g., the free-trial tenant, or the CPU-intensive tenant), the multiplexing can be turned on; if the requirement is strict, the multiplexing can be turned off; the QoS service level in the middle is to allow tenants (or operators) to switch between shared (multiplexing) rate limiters and dedicated ones.

## VIII. Conclusion

In this paper, we have shown a deployment model, including two algorithms to instruct the rate limiters placement and a framework to multiplex rate limiters to improve the rate limiters' utilization, of rate limiters in the cloud. Specifically, the incremental strategy performs nearly as good as optimal solution in practice and multiplexing strategy can achieve the same effect of per-flow rate limiting but only needs no more than 20% rate limiters.

In the future, we hope to improve rate limiters multiplexing by leveraging SDN-based mechanism, so that flows can be mapped to rate limiters dynamically to reduce the potential queuing latency for mice flows.

## References

[1] V. Arun and H. Balakrishnan, "Copa: Practical delay-based congestion control for the internet," in *15th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 18)*, 2018, pp. 329–342.

[2] K. He, W. Qin, Q. Zhang, W. Wu, J. Yang, T. Pan, C. Hu, J. Zhang, B. Stephens, A. Akella *et al.*, "Low latency software rate limiters for cloud networks," in *Proceedings of the First Asia-Pacific Workshop on Networking*. ACM, 2017, pp. 78–84.

[3] M. P. Grosvenor, M. Schwarzkopf, I. Gog, R. N. Watson, A. W. Moore, S. Hand, and J. Crowcroft, "Queues dont matter when you can {JUMP} them!" in *12th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 15)*, 2015, pp. 1–14.

[4] M. Alizadeh, S. Yang, M. Sharif, S. Katti, N. McKeown, B. Prabhakar, and S. Shenker, "pfabric: Minimal near-optimal datacenter transport," in *ACM SIGCOMM Computer Communication Review*, vol. 43, no. 4. ACM, 2013, pp. 435–446.

[5] L. Chen, Y. Feng, B. Li, and B. Li, "Efficient performance-centric bandwidth allocation with fairness tradeoff," *IEEE Transactions on Parallel and Distributed Systems*, vol. 29, no. 8, pp. 1693–1706, 2018.

[6] Y. Cheng, S. Xiao, J. Liu, F. Guo, R. Qin, B. Li, and X. Yuan, "An approximate bandwidth allocation algorithm for tradeoff between fairness and throughput in wsn," *Wireless Networks*, vol. 24, no. 6, pp. 2165–2177, 2018.

[7] P. Zhang, Y. Gang, X. Huang, S. Zeng, and K. Xie, "Bandwidth allocation with utility maximization in the hybrid segment routing network," *IEEE Access*, 2019.

[8] A. Kumar, S. Jain, U. Naik, A. Raghuraman, N. Kasinadhuni, E. C. Zermeno, C. S. Gunn, J. Ai, B. Carlin, M. Amarandei-Stavila *et al.*, "Bwe: Flexible, hierarchical bandwidth allocation for wan distributed computing," *ACM SIGCOMM Computer Communication Review*, vol. 45, no. 4, pp. 1–14, 2015.

[9] K. Nagaraj, D. Bharadia, H. Mao, S. Chinchali, M. Alizadeh, and S. Katti, "Numfabric: Fast and flexible bandwidth allocation in datacenters," in *Proceedings of the 2016 ACM SIGCOMM Conference*. ACM, 2016, pp. 188–201.

[10] B. Raghavan, K. Vishwanath, S. Ramabhadran, K. Yocum, and A. C. Snoeren, "Cloud control with distributed rate limiting," in *ACM SIGCOMM Computer Communication Review*, vol. 37, no. 4. ACM, 2007, pp. 337–348.

[11] P. Soares, J. Santos, N. Tolia, D. Guedes, and Y. Turner, "Gatekeeper: Distributed rate control for virtualized datacenters," Technical Report HP-2010-151, HP Labs, Tech. Rep., 2010.

[12] C. Liang and F. R. Yu, "Distributed resource allocation in virtualized wireless cellular networks based on admm," in *2015 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. IEEE, 2015, pp. 360–365.

[13] J. Guo, F. Liu, X. Huang, J. C. Lui, M. Hu, Q. Gao, and H. Jin, "On efficient bandwidth allocation for traffic variability in datacenters," in *IEEE INFOCOM 2014-IEEE Conference on Computer Communications*. IEEE, 2014, pp. 1572–1580.

[14] A. Shieh, S. Kandula, A. G. Greenberg, and C. Kim, "Seawall: Performance isolation for cloud datacenter networks." in *HotCloud*, 2010.

[15] A. Saeed, N. Dukkipati, V. Valancius, C. Contavalli, A. Vahdat *et al.*, "Carousel: Scalable traffic shaping at end hosts," in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*. ACM, 2017, pp. 404–417.

[16] V. Jeyakumar, M. Alizadeh, D. Mazieres, B. Prabhakar, and C. Kim, "Eyeq: Practical network performance isolation for the multi-tenant cloud," in *Presented as part of the*, 2012.

[17] S. Radhakrishnan, Y. Geng, V. Jeyakumar, A. Kabbani, G. Porter, and A. Vahdat, "{SENIC}: Scalable {NIC} for end-host rate limiting," in *11th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 14)*, 2014, pp. 475–488.

[18] A. S. Khope, A. A. Saleh, J. E. Bowers, and R. C. Alferness, "Elastic wdm crossbar switch for data centers," in *2016 IEEE Optical Interconnects Conference (OI)*. IEEE, 2016, pp. 48–49.

[19] "https://barefootnetworks.com/products/brief-tofino/."

[20] "https://www.intel.com/content/www/us/en/trademarks/intel-flexpipe.html."

[21] "https://www.cavium.com/xpliant-packet-trakker-programmable-telemetry-solution.html."

[22] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese *et al.*, "P4: Programming protocol-independent packet processors," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 3, pp. 87–95, 2014.

[23] X. Jin, X. Li, H. Zhang, R. Soulé, J. Lee, N. Foster, C. Kim, and I. Stoica, "Netcache: Balancing key-value stores with fast in-network caching," in *Proceedings of the 26th Symposium on Operating Systems Principles*. ACM, 2017, pp. 121–136.

[24] R. Kundel, J. Blendin, T. Viernickel, B. Koldehofe, and R. Steinmetz, "P4-codel: Active queue management in programmable data planes," in *2018 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*. IEEE, 2018, pp. 1–4.

[25] N. Zilberman, Y. Audzevich, G. Kalogeridou, N. Manihatty-Bojan, J. Zhang, and A. Moore, "Netfpga: Rapid prototyping of networking

devices in open source," in *ACM SIGCOMM Computer Communication Review*, vol. 45, no. 4. ACM, 2015, pp. 363–364.

[26] Y. Li, R. Miao, C. Kim, and M. Yu, "Flowradar: A better netflow for data centers," in *13th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 16)*, 2016, pp. 311–324.

[27] "https://www.first.org/."

[28] "http://maccdc.org/."

[29] "https://www.oooverflow.io/dc-ctf-2018-finals/."

[30] T. Benson, A. Akella, and D. A. Maltz, "Network traffic characteristics of data centers in the wild," in *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*. ACM, 2010, pp. 267–280.

[31] E. Vanini, R. Pan, M. Alizadeh, P. Taheri, and T. Edsall, "Let it flow: Resilient asymmetric load balancing with flowlet switching," in *14th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 17)*, 2017, pp. 407–420.

[32] C. Li, C. Kai, B. Wei, and M. Alizadeh, "Scheduling mix-flows in commodity datacenters with karuna," in *Acm Sigcomm Conference*, 2016.

[33] N. Farrington and A. Andreyev, "Facebook's data center network architecture," in *2013 Optical Interconnects Conference*. Citeseer, 2013, pp. 49–50.

[34] Z. Fu, Z. Liu, J. Gao, W. Zhou, W. Xu, and J. Li, "Coral: A multi-core lock-free rate limiting framework," in *2017 International Conference on Computing, Networking and Communications (ICNC)*. IEEE, 2017, pp. 638–642.

[35] S. Hu, W. Bai, K. Chen, C. Tian, Y. Zhang, and H. Wu, "Providing bandwidth guarantees, work conservation and low latency simultaneously in the cloud," *IEEE Transactions on Cloud Computing*, 2018.

[36] "http://man7.org/linux/man-pages/man8/tc-htb.8.html."

[37] "https://elixir.bootlin.com/linux/v3.6/source/net/sched/sch_tbf.c."

[38] G. Lu, C. Guo, Y. Li, Z. Zhou, T. Yuan, H. Wu, Y. Xiong, R. Gao, and Y. Zhang, "Serverswitch: a programmable and high performance platform for data center networks." in *Nsdi*, vol. 11, 2011, pp. 2–2.

[39] "https://www.cisco.com/c/en/us/support/docs/smb/switches/cisco-esw2-series-advanced-switches/smb4079-quality-of-service-qos-queue-settings-on-esw2-350g-switches.html."

[40] "https://p4.org/."