IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received March 3, 2020, accepted March 17, 2020, date of publication March 30, 2020, date of current version April 16, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.2984277

Improving Flow Scheduling Scheme With Mix-Traffic in Multi-Tenant Data Centers

SHUO WANG, JIAO ZHANG, TAO HUANG, TIAN PAN, JIANG LIU, AND YUNJIE LIU

State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications (BUPT), Beijing 100876, China
Purple Mountain Laboratories, Nanjing 211111, China
Beijing Advanced Innovation Center for Future Internet Technology, Beijing 100124, China
Beijing Laboratory of Advanced Information Networks, Beijing 100876, China

Corresponding author: Shuo Wang (shuowang@bupt.edu.cn)

This work was supported in part by the National Key Research and Development Program of China under Grant 2018YFB1800500, in part by the National Natural Science Foundation of China under Grant 61902033 and Grant 61872401, in part by the Beijing Natural Science Foundation under Grant 4204105, in part by the China Postdoctoral Science Foundation under Grant 2018M641281, in part by the MoE-CMCC "Artificial Intelligence" Project under Grant MCM20190701, in part by the Research and Development Program in Key Areas of Guangdong Province under Grant 2018B010113001, and in part by the Chinese Academy of Engineering under Grant 2019-XY-5.

ABSTRACT Data centers need low-latency fabrics. Several flow scheduling schemes have been proposed to minimize the Flow Completion Time (FCT) based on Shortest Job First (SJF) heuristic. However, to mimic SJF, previous proposals sacrifice the generality (e.g., pFabric requires special hardware) or sacrifice the performance to guarantee the generality (e.g., PIAS loses some of pFabric's performance). Especially, in multi-tenant data centers, traffic patterns from different applications are mixed together and vary over time, thereby creating even more challenges. In this paper, we investigated that the performance of information-agnostic scheme could be further improved by leveraging the unique characteristics of different traffic types. Based on this investigation, we present Traffic Prediction based Flow Scheduling (TPFS), aiming at achieving near-optimal performance and good generality in multi-tenant data centers with the mix-traffic pattern. To achieve near-optimal performance, we design a two-stage machine learning algorithm to first automatically cluster flows with the similar flow size distribution and then predict the priorities of flows based on the clustering results. Besides, we implement TPFS in virtual switches, which exerts fine-grained flow scheduling over the arbitrary network stacks of tenants. Testbed evaluation and simulations show that TPFS outperforms the previous information-agnostic flow scheduling scheme PIAS and greatly reduces the tail latency of the network.

INDEX TERMS Flow scheduling, datacenter networks, scheduling.

I. INTRODUCTION

In data centers, applications, such as web search, advertising, social networking and retail, often generate small-size requests that need to be finished within microseconds [1]–[4]. Flows that fail to finish before their deadlines are abandoned, which will cause bandwidth waste, user experience degradation and finally reduce the revenue of providers. The root cause of the above problem lies in that the latency-sensitive short flows are often blocked by large flows generated by co-existing workloads, such as data mining [5], backup and video stream. However, today's data center transport protocols like TCP, [1], [6]–[8] are oblivious to the latency requirements of short flows and treat the two types of flows equally, which significantly increases the Flow Completion Time (FCT) of short flows.

Motivated by this, a number of novel data center transport designs have been proposed [4], [9]–[11]. Broadly speaking, most of them are built on the SJF heuristic that assigns short flows more bandwidth and high priority to minimize their completion times. To prioritize short flows, these proposals need to identify short flows and are aware of the size of flows. Therefore, most of the proposals, such as pFabric [4] and PDQ [10], assume the flow size information can be easily obtained from applications. Particularly, pFabric can achieve close to theoretically minimal latency over a variety of workloads. The performance, however, is gained at the cost of the assist of special hardware, packet header customization,

The associate editor coordinating the review of this manuscript and approving it for publication was Guangjie Han.

flow size information requirement, which greatly sacrifices the generality.

To mimic SJF without prior knowledge, some proposals [11], [12] leverage commodity switches coupled with priority queues to achieve good generality. For example, in PIAS [11], a flow is gradually demoted from higher-priority queues to lower priority-queues according to its sent bytes. While this can emulate SJF without knowing flow size, the good generality comes at the cost of losing performance that the average 99th FCT of the short flows is 20% worse than pFabric in web search workload [11] and is 40% worse than pFabric in mix-traffic (shown in Section VI).

Furthermore, multi-tenant data centers are increasingly deployed in large providers, such as Google, Amazon and Microsoft, to support a large variety of applications. By our investigation, there are mainly two challenges: 1) traffics from different applications are mixed together and dynamically vary along the time, making the priority policy mismatches with the underlying traffic; 2) the network stack is managed by users, making previous proposals that modify TCP or kernels challenging to be deployed in multi-tenant data centers [3], [4], [9]–[12].

In this paper, we raise a question: is it possible to achieve near-optimal performance and obtain good generality simultaneously even in more challenging multi-tenant data centers?

We observe that the main challenge of using SJF based schemes is the difficulty in obtaining flow size information. Instead of simply assuming that flow information is completely agnostic, if the flow size can be estimated according to historic flow trace, achieving near-optimal performance with good generality is possible. Thus, our key insight to solve the problem is that the flow size should be estimated before the start of flows to further reduce the latency.

This paper presents Traffic Prediction based Flow Scheduling (TPFS), a new transport design aimed at achieving near-optimal performance and good generality even in more challenging multi-tenant data centers with mix-traffic. At a high-level, TPFS predicts flow priority by using the proposed two-stage machine learning algorithm and schedule flows according to their priorities.

The first stage of TPFS is predicting flow size distributions. We have investigated that the traffic is predictable, and there are a lot of schemes designed for traffic prediction and classification [13], [14]. However, several challenges arise when employing these schemes to predict the size of flow. Since existing prediction algorithms are not perfect, most of them can only predict the type of application and have potential prediction errors. Our scheduling schemes have to tolerate the imperfect prediction results. Thus, TPFS only assumes the prediction algorithm can give a correct range of flow size and schedules flows based on the flow size distribution. Second, using different features may have different prediction results, TPFS searches the best features to find the best prediction results and merges the traffic with similar features using the DBSCAN clustering algorithm to reduce the number of types of traffic.

The second stage of TPFS is predicting flow priority. The simplest way is to assign flow priority according to the average size in the flow size distribution. However, this heuristic scheme will increase the tail latency since the short flows are given the same priority as the larger size flows. To solve this problem, a flow should be gradually demoted from the highest priority to the lowest priority when it sends more data. However, large flows also generate traffic (called bad load) in the higher priority queues, which increases the queueing delay. Our analysis finds that throughput-sensitive applications only have a small fraction of short flows while latency-sensitive applications contain a large number of short flows. Thus, TPFS sacrifices the performance of short flows in throughput-sensitive applications to decrease the queueing delay by removing the bad load. More generally, when there are several types of applications with the different fraction of short flows, we have to determine to sacrifice the performance of which type of application for achieving the optimal performance. We formulate the problem as an optimization problem and give the maximum percentage of short flows that we can sacrifice for increasing the performance.

On the other hand, motivated by AC $\not\downarrow$ DC [7], TPFS is implemented in the virtual switches to address the difficulty in deployment. Further, since traffic will pass through virtual switches, TPFS can easily monitor flows and collect historic flow information of all Virtual Machines(VMs).

We have implemented TPFS and built a small-scale testbed with 8 hosts to evaluate its performance with realistic workloads from four different applications. Furthermore, we also implement TPFS in ns-2 simulator to perform large-scale simulations. The testbed experiments show that TPFS outperforms PIAS more than 30% for short flows. Our simulation results show that the performance gap between TPFS and pFabric is within 5% for short flows, and TPFS reduces the tail latency up to 30% compared to PIAS.

The main contributions of TPFS are:

- We investigate the performance of pure information agnostic schemes can be further improved by predicting flow size.
- We propose TPFS, a prediction based flow scheduler to optimize the FCT for flows in data center networks.

The rest of paper is organized as follows. Section II-A briefly overviews the background of our work and shows the motivation. Section III to Section IV show the architecture of TPFS and design the scheduling algorithm. Section V and Section VI extensively evaluate our algorithm through testbed experiments and simulations. Finally, the paper is concluded in VII.

II. BACKGROUND AND MOTIVATION

In this section, we first introduce the background of this work. Then, we present the main motivations of our work. The first example shows that mismatched thresholds reduce could reduce the performance of the information agnostic approach. The second example shows that the

performance gap between the information-agnostic approach and the information-aware approach. Through the two motivation examples, we show the key considerations of TPFS.

A. BACKGROUND

1) INFORMATION-AGNOSTIC FLOW SCHEDULING

The flow scheduling schemes need to know the flow size or deadlines to allocate rate, adjust congestion window, and schedule flows. Most of the schemes assume the flow size can be easily obtained by modifying applications. However, modifying applications is difficult and very costly [15]. Besides, in Spark, the data is dynamically generated during computation, making it is impossible to know the flow size priori [16]. Therefore, information-agnostic flow scheduling scheme PIAS [11] has been proposed, and many other schemes [12], [15], [17] are built on PIAS or leverage the key idea of PIAS.

PIAS leverages multiple priority queues available in existing commodity switches that flows in different queues are scheduled with strict priority, while packets of flows in the same queue are scheduled based on FIFO. In PIAS, a flow is gradually demoted from the highest priority queue to several lower priority queues when its sent bytes exceeds the predefined thresholds. If there are K queues, network managers need to predefine $K-1$ demotion thresholds $\alpha_1 \leq \alpha_2 \cdots \leq \alpha_{K-1}$ according to the traffic pattern. When traffic pattern changes, to obtain the best performance, the demotion thresholds need to be changed accordingly.

2) MULTI-TENANT DATA CENTERS

Multi-tenant data centers and cloud computing are emerging technologies that have been widely used by larger providers and companies. At a high level, the provided infrastructures can be accessed remotely, and the operating system can be configured and changed arbitrarily. Thus, compared to traditional data centers, multi-tenant data centers are more challenging to manage. For flow scheduling schemes, the challenges mainly come from two aspects.

- **Deployment:** Since users control the operating system of VMs, data center managers are unable to control the network stacks. Without control over the network stack, many proposals [3], [4], [9]–[12] cannot work. Besides, using different network stacks will cause unfairness between users [7], [8]. For example, some users may uses D^{2}TCP while other uses still use ECN-incapable TCP. As a result, D^{2}TCP flows will obtain more bandwidth than ECN-incapable flows. Even all users use the same protocol, unfairness may still arise when users use different parameters.
- **Mixed and Varying Traffic Pattern:** In a relatively large time scale, flows generated by different types of applications are mixed together. Karuna [12] and PIAS [11] generally set the thresholds according to the flow size distribution of all flows. However, during a small time scale, there may be only one or two types of applications that are active, which causes the mismatch between predefined thresholds and underlying traffic will happen. This issue may decrease the performance, and we will show this in our motivation examples.

3) CAUSES OF VARYING TRAFFIC PATTERN

The traffic pattern inevitably changes over time, and we summary the causes as fellows:

- **VMs Start/Stop:** Multi-tenants data centers allow users to rent VMs on demands. For example, in Amazon EC2, users can rent VMs from serval hours to couples of days. Since different users may run different applications, the traffic patten will change as the set up of new VMs or the destroy of old VMs.
- **Application Change:** Since users fully control their VMs, they may change the function of VMs. For instance, they may deploy web search at the beginning and change to data mining application letter. This will also change the traffic pattern of the network.
- **VMs Migration:** VMs migration technologies are widely used in multi-tenants data centers [18], [19]. Data center providers usually migrate VMs to the same physical machines to save energy, or migrate VMs to the same rack to reduce latency. When VMs are migrated from one rack to another rack, the underlying traffic patten may also change.

B. MOTIVATION

In this subsection, we conduct some simulations to show the problems and our motivation.

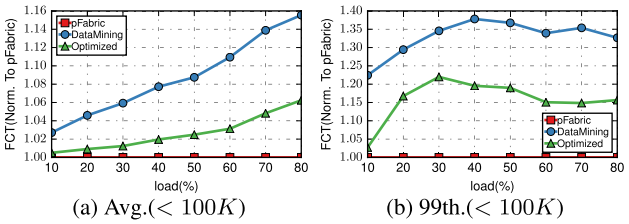
1) PROBLEM OF MISMATCHED THRESHOLDS

Although PIAS leverages ECN [1] to mitigate the threshold mismatch problem, we will show the tail latency of the network is still decreased compared to the optimal settings.

To explore the impacts of mismatched thresholds, we compare two settings in web search workload by varying the network load from 0.1 to 0.8. The Data Mining thresholds are calculated according to data mining workload, and the Optimized thresholds are obtained by running the simulation multiple times. The simulations are run at 10Gbps links using the simulation code and settings from the PIAS paper [20]. The simulation results are shown in Figure 1.

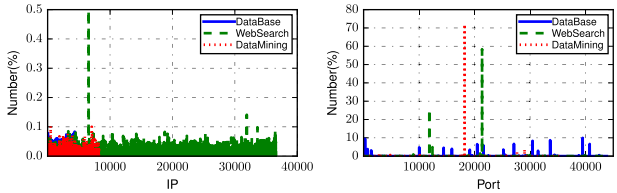
Figure 1a shows the average FCT for short flows. We can see that the performance of PIAS decreases about 10% at 0.8 load when the thresholds are changed from Optimized to Data Mining. As to the tail latency shown in Figure 1b, the performance decreases about 20% at 0.8 load. This is because that the flow size distribution of web search and data mining (shown in Figure 3) are very different that 80% of flows in data mining are smaller than 1KB while only 20% of flows in web search are smaller than 1KB. As a result, mismatched thresholds significantly reduce the performance.

Observation 1: Flow scheduling schemes should handle mixed and varying traffic.



(a) Avg.($< 100K$) (b) 99th.($< 100K$)

FIGURE 1. Motivation example: PIAS has sub-optimal performance in web search workload.



(a) IP address (b) Transport ports

FIGURE 2. The histogram of IP and transport ports for three Facebook workloads from data base, web search, data mining clusters.

2) LARGE IMPROVEMENT SPACE FOR TAIL LATENCY

The priority of flows is gradually decreased in PIAS to mimic SJF heuristic. Does this mean PIAS can mimic SJF perfectly and achieve near-optimal performance? We answer this question by comparing PIAS to pFabric [4] to find out the performance gap between PIAS (pure information-agnostic) and optimal SJF-based scheme (pure information-aware). Note that, pFabric schedules flows according to their remaining size and gives flows with the smallest remaining size the highest priority. We can see pFabric as the optimal solution to achieve SJF.

Figure 1a shows that pFabric outperforms PIAS by up to 6% for short flows at 0.8 load, while the performance gap increases to about 20% at 0.3 load for 99th short flows. This indicates that the improvement space of information-agnostic scheme is large. By analyzing PIAS, we find that PIAS cannot quickly identify short flows and large flows. When short flows and large flows arrive at the same time, the short flows will be blocked by large flows until large flows are demoted to lower priority queues. As a result, the completion times of short flows are greatly affected by other flows. For example, we use $\alpha_1 = 46$ packets in Optimized thresholds, and about 70% of flows are larger than α_1. This means the short flows(< 46 packets) has 70% probability been blocked if it arrives with another flows at the same time.

3) PREDICTABLE TRAFFIC

Can we identify short flows and large flows quickly to improve this problem? By our investigation, if we can predict flow size and obtain some flow information, we can mitigate the problem. Fortunately, much existing work shows that the type of flows is predictable and applications have their unique characters [13], [14], [21], [22].

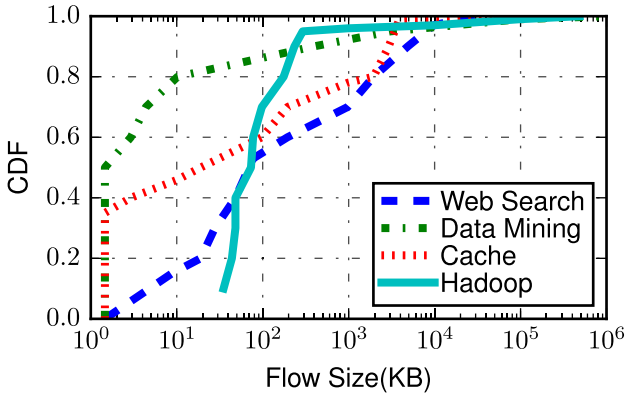


FIGURE 3. The flow size distribution of web search, data mining, cache and hadoop workloads.

To show this, we count the IP address and transport ports from three realistic workloads collected at Facebook's data base, web search and data mining cluster [21], and we plot the histogram in Figure 2. Note that since the IPs and ports are encrypted, the values are not real IPs and ports used by Facebook. We can see that in each workload, each machine send similar numbers of flows and there are no hot machines. In contrast, the majority of flows are sent from a small number of ports as shown in Figure 2b. Note that, in data base workload, although there are no obvious hot-ports, 16 ports generate more than 92% the number of flows. Thus, the transport ports contain the information of the type of flows. On the other hand, we plot the web search workload [1], the data mining workload [23], the cache workload [21], and the hadoop workload [21] in Figure 3. We can find that all the four applications have their unique flow size distributions. Hence, if we can predict the type of a flow, we can know the range of its flow size and further predict its priority.

Observation 2: Pure information-agnostic schemes have large improvement space

III. TPFS OVERVIEW

This section outlines the design and architecture of TPFS. TPFS makes a trade-off between pure information-agnostic and pure information-aware flow scheduling mechanisms. The goal of TPFS is to design an effective and practical prediction-based flow scheduling that leverages the potential information to predict flow size and further reduces the tail latency of short flows.

Figure 4 shows the main architecture of TPFS. To adapt to the variation of the underlying traffic, we need a centralized architecture to adjust the settings of distributed end-hosts accordingly. Thus, TPFS uses a loosely-coordinated architecture [17]. This architecture mainly contains, distributed end-hosts, a centralized master node, and hardware switches.

A. END-HOSTS

To avoid controlling over the VMs or requiring changes the network stack, motivated by AC ⚡DC [7], the main functions

of TPFS are implemented in the virtual switch (vSwitch) running on distributed end-hosts. vSwitch-based implementation is very light-weight and scalable to achieve good generality. More specifically, in vSwitch, TPFS records the flow size of all finished flows and associated information including MAC address, IP address and transport ports. Besides, during the transmission of a flow, TPFS adjusts its priority according to the priority policy made by the master. Similar with PIAS, the TOS field of packets are tagged at end-hosts by vSwitches to notify switches the priority of flows.

B. MASTER NODE

Distributed end-hosts locally collect flow information and send the information to the master. Then, the master analyzes global flow information using prediction algorithms to make priority policies. The flow information and priority policies are coordinated loosely at $O(10)$ minutes interval. Besides, we note that existing commercial platforms have already adopted the centralized architecture to manage vSwitches and the network, such as VMware NSX [24] and OpenStack Neutron [25]. Thus, the master can be easily implemented as a plugin in these platforms.

C. HARDWARE SWITCHES

TPFS leverages the existing priority queues of hardware switches. Specifically, packets are classified into different priority queues based on the TOS tag in the packet header. Then, packets are dequeued from the highest priority queue to the lowest priority queue when the link is idle.

D. KEY IDEA

Based on observation 2, we can find that pure information-agnostic approaches are not optimal. To achieve the best generality and simplicity, it assumes the information is completely unknown. However, by our investigation, the flow size is predictable in data centers. If we can predict flow size, we can increase the performance. Thus, the key design of TPFS is leveraging potential information to predict flow size while still achieves good transparency to applications.

At a high level, TPFS mainly have two steps to predict the priority:

- **Predicting Flow Size Distribution:** Given a set of flow size from all finished flows, TPFS selects key features that aggregate flows with similar size and split flows into different types via clustering algorithm.
- **Predicting Flow Priority:** Then, TPFS predicts the flow priority based on the flow size distribution of each type of flows. Finally, the flows are scheduled according to the predicted priority at hardware switches.

IV. TPFS DETAILED DESIGN

In this section, we describe the detailed design of TPFS. We begin by describing our two-stage prediction algorithm and then show how to determine the scheduling thresholds.

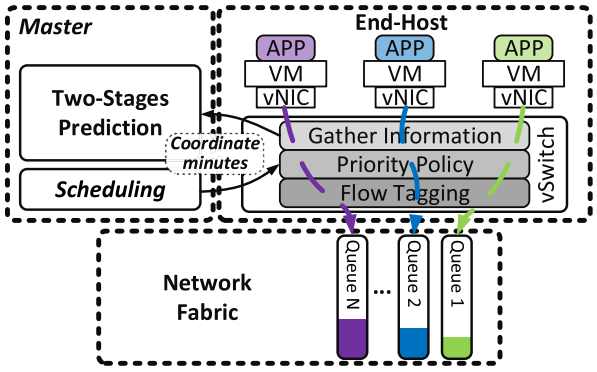


FIGURE 4. The architecture of TPFS .

TABLE 1. Potential useful features for classification.

Level	Features
Flow Level	Sent bytes, Flow duration, Packet length , Packet inter arrival time
Community Level	The position of machines, The service group
Application Level	IP addresses, Transport port
OS Level	Process ID and names of applications

A. PREDICTING FLOW SIZE DISTRIBUTION

Given periodic flow information, TPFS should predict flow size and the priority of flows according to the information. However, as shown in Figure 5, the flow size range of all flows may be too large that we cannot accurately predict flow size. Thus, we need some key features to further split the flows into different types that flows have similar size in each type. For example, in Figure 5, we may find features to split flows into two types and each type has small flow size range.

TPFS needs to select some features and uses features to split flows into different types. As shown in Table 1, the potential useful features can be classified into four levels [13], [15].

- **Flow-level** includes flow start time, mean packet size and average packet interval. These features are widely used by flow classification algorithms [13], [14].
- **Community-Level** includes machine id, the position of machines, the service group. Since machines in data centers are organized at racks and separated into service groups where flows in the same machine, racks, or service groups may have similar size.
- **Application-Level** includes IP addresses, transport port. Applications usually use their reserve port to send data or specific machines only run one type of application. For example, HTTP traffic uses port 80, FTP uses port 21 and shuffle process of Hadoop uses 13652 by default [15]. Thus, IPs and port may indicate application types.
- **OS-level** includes process ID and names of applications.

We further analyze this features and find that community-level and OS-level features are challenging to collect in multi-tenant data centers, because the VMs may be migrated to new

positions and we cannot control the stacks of VMs. On the other hand, we have shown that the transport ports have the strong correlation with the type of applications. Thus, we mainly leverage the transport ports, IP addresses and sent bytes.

Since we don't know which feature is important, the key idea of our algorithm is to search all possible feature combinations and find the most useful features. For example, if we aggregate flows according to their IP addresses and find the range of flow size distribution is too large, then IP addresses is not a good feature for the underlying traffic. In contrast, if we change to use the transport ports, we can get flow size distribution varying in a small range. This idea is also used by CS2P [26] that designed for predict throughput of flows.

The flow clustering algorithm of TPFS works in four steps:

1) Choose feature combinations from all the features as selected feature set $\mathbf{Feature_s}$. For example, if there are n features, we will try 2^n combinations. In our work, we mainly consider the IP address and transport ports.

2) After selecting the features, the feature combination has many values, we statistic the flow size of each value and get a flow size distribution. For example, if we select the port as the feature, and we can get the flow size distribution of $port = 80$ is F_1, the flow size distribution of $port = 22$ is F_2. Since there are a lot of flow size distributions, we need to cluster similar flow size distributions to reduce the number of types. Before clustering the similar flow size distribution, we first show how to measure the similarity and distance between two flow size distributions. Assuming the flow size distribution for all flows ranges from x_{min} to x_{max}. We define $x_i = x_{min} + i \times \frac{x_{max} - x_{min}}{M+1}$, and for a flow size distribution F(x), we can describe it with $\mathbf{X_i} = F(x_1^i), F(x_2^i), \ldots, F(x_M^i)$ where M determines the accuracy of the description. Then, the distance between two flow size distribution is:

$$d(F_i, F_j) = ||\mathbf{X_i} - \mathbf{X_j}|| = \sqrt{\sum_{l=1}^{M} (x_l^i - x_l^j)^2} \tag{1}$$

3) Using $\mathbf{X_i}$, each flow size distribution can be mapped to a point in the M-dimension space. Now, we can use DBSCAN [27] to automatically learn the number of types N and classify flows into type T_i.

4) Assuming we get N types of flow size distributions with F_i where δ_i is the standard deviation of F_i. Our goal is to find the best combination features $\mathbf{Feature_s^*}$ that minimize the sum of standard deviation:

$$\mathbf{Feature_s^*} = \min_{\mathbf{Features_s}} \sum_{i=1}^{n} \delta_i \tag{2}$$

The performance and scalability of this algorithm are mainly determined by the DBSCAN. Since we only use flow history in the past $O(1)$ hours, the total data set will not too large. Besides, the flow size clustering is run at $O(10)$ minutes interval, TPFS can tolerate minutes level running time.

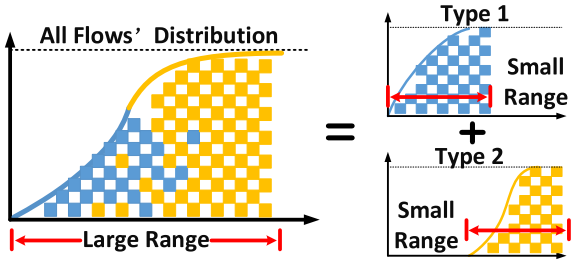


FIGURE 5. The motivation of flow size distribution prediction.

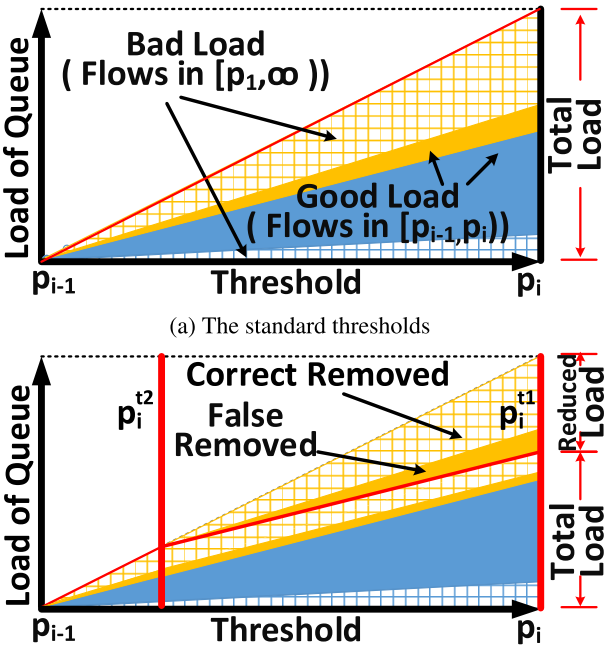


FIGURE 6. The motivation of priority prediction: Types T_1 (dark/blue) and T_2 (light/yellow) generate different percentage of bad load (with grid) in the queue P_i.

B. PREDICTING FLOW PRIORITY

Assuming the flows have been clustered into N classes T_i, $1 \le i \le N$, and there are K priority queues P_i, $1 \le i \le K$ where P_1 has the highest priority. Given a flow $f \in T_i$, TPFS predicts which priority the flows belong to.

However, the main challenge of this problem is that although the size range of each type of class is small, it may also coverage several queues, and it is difficult to predict the correct priority immediately. More specifically, to mimic SJF, the optimal priority policy is giving flows with small size the highest priority. Thus, when the flow size is known priori, pure information-aware scheme [4] can determine a series of thresholds that classify flows into priority queues based on their size. For instance, for a queue P_i and associated threshold $p_1, p_1 \le p_2 \cdots \le p_k$, it should serve flows in $[p_{i-1}, p_i)$. However, in TPFS, we don't know the flow size, and we only know the flow size distribution. If the flow size of a class varies in $[p_i, p_{i+3}]$, and 90% percentage of flows

is in $[p_{i+2}, p_{i+3})$. In this situation, if we use the average size as the predicted flow size and classify all flows into P_{i+3}, then the FCT of 10% flows that belong to $[p_i, p_{i+2})$ will increase. Especially, if these 10% flows are short flows, the tail-latency will significantly increase. Apparently, this heuristic will violate the SJF policy.

1) PROBLEM OF PIAS

Pure information-agnostic scheme PIAS [11] solves this problem by using the sent bytes of a flow, and queue P_i serves flows whose sent bytes in $[p_{i-1}, p_i)$. Thus, a flow is gradually demoted from the highest priority to lowest priority when it sends more data. However, this scheme also has the drawback that cannot remove unnecessary load generated by large flows from high priority queues, and we use a simple example shown in Figure 6 to show this. The load ρ_i of a queue P_i mainly contains two parts. The first part of the load is the traffic generated by flows whose size in $[p_{i-1}, p_i)$(good load in the Figure). The second part of the load is the traffic generated by flows within $[p_i, \infty)$(bad load in the Figure). Obviously, the second part of the load should be removed to reduce the queueing delay since it belongs to large and low priority flows. However, for pure information agnostic schemes, it is hard to identify this part of flows.

2) KEY IDEA OF PREDICTING PRIORITY

TPFS leverages the pure information-agnostic scheme and further improves its performance by removing the bad load. As shown in Figure 6b, TPFS has learned a number of classes, and each class may have different flow size distribution. For example, 80% flows of class T_1 is in P_i while 80% flows of class T_2 is in the lower priority queues than P_i. Thus, we can directly remove most of the flows of T_2 to reduce almost half of the load of P_i. However, if we remove all flows of T_2 from P_i to P_{i+1}, 20% of high priority flows are also removed, and the FCT of those flows are increased. We call those flows are false removed flows. Therefore, we have to make a tradeoff between the percentage of removed bad load and the percentage of false removed flows. For example, as shown in Figure 6b, we made the tradeoff by finding a proper threshold p_i^{t2} that removes 70% bad load with only 5% false removed flows.

3) PROBLEM FORMULATION

We assume ρ_i is the load of the queue P_i, and P_i serves two parts of flows. The first part is good flows in $[T_{i-1}, T_i)$ with load ρ_i^G and the second part is bad flows in $[T_i, \infty)$ with load ρ_i^B. Thus, $\rho_i = \rho_i^G + \rho_i^B$, and the serving time for good flows is $W_i^G = \sum_{k=1}^{i-1} W_k^G + \frac{R_i^G}{1-\sum_{l=1}^{i}\rho_l}$ where R_i^G is the average remaining size of good flows [11]. If we sacrifice α percentage of good flows for removing β percentage of bad flows, we can reduce the load of P_i to $\widehat{\rho_i} = (1-\alpha)\rho_i^G + (1-\beta)\rho_i^B$. Then, we have the following optimization problem, we choose an optimal α to minimize the objective: the average serving time of good flows in P_i:

$$\begin{aligned}\min_{\alpha}\ \widehat{W_i^G} &= (1-\alpha)[\sum_{k=1}^{i-1} W_k^G + \frac{R_i^G}{1-\sum_{l=1}^{i-1}\rho_l + \widehat{\rho_i}}] \\ &+\alpha[\sum_{k=1}^{i-1} W_k^G + \frac{R_i^G}{1-\sum_{l=1}^{i-1}\rho_l + \widehat{\rho_i}} + \frac{R_i^G}{1-\sum_{l=1}^{i+1}\rho_l}] \\ \textit{subject to}\quad & 0 \le \alpha \le 1 \end{aligned} \tag{3}$$

where the first part is the serving time of G_i in queue P_i and the second part is the serving time of sacrificed G_i in queue P_{i+1}.

4) ANALYSIS

Recall that we cluster flows into N types, and each type T_i has a unique flow size distribution $F_i(x)$. Thus, the good load ρ_i^G and bad load ρ_i^B are generated by the N types, and $\rho_i^G = \sum_{l=1}^{N}\rho_{i,T_l}^G$ and $\rho_i^B = \sum_{l=1}^{N}\rho_{i,T_l}^B$ where ρ_{i,T_l} is the load of T_l in queue P_i. If we gives T_l a threshold $p_i^{T_l}$ and demote the type T_l flows into P_{i+1} when their sent bytes exceed $p_i^{T_l}$, we will remove $\alpha_{i,T_l} = F_l(P_i) - F_l(P_i^{T_l})$ percentage good flows and $\beta_{i,T_l} = \frac{P_i^{T_l} - P_{i-1}}{P_i - P_{i-1}}$ percentage bad flows (note that a bad flow will send $T_i - T_{i-1}$ bytes in P_i, and it sends $P_i^{T_l} - P_{i-1}$ bytes when the threshold is $P_i^{T_l}$). Then, $\alpha_i = \sum_{l=1}^{N}\alpha_{i,T_l}$ and $\beta_i = \sum_{l=1}^{N}\beta_{i,T_l}$. Finally, the above optimization problem becomes that we choose an optimal set of thresholds $\{p_i^{T_l}\}$ for each queue P_i to minimize the equation 3.

5) PROBLEM SOLUTION

The above problem is a Sum-of-Linear-Ratios (SoLR) problem that is NP hard [11], making we can not solve the problem. To solve the problem, we note that the optimal solution should remove maximum bad load while removing minimum good load. Thus, we consider the following optimization problem, we choose a set of thresholds $\{p_i^{T_l}\}$ to maximize the removed bad load:

$$\begin{aligned}\max_{\{p_i^{T_l}\}}\ \beta\rho_i^B &= \sum_{l=1}^{N}\frac{P_i^{T_l} - P_{i-1}}{P_i - P_{i-1}}\rho_{i,T_l}^B \\ \textit{subject to}\ \alpha\rho_i^G &= \sum_{l=1}^{N}(F_l(P_i) - F_l(P_i^{T_l}))\rho_{i,T_l}^G \le \delta \end{aligned} \tag{4}$$

Note that p_{i-1} is predefined thresholds, and ρ_{i,T_l}^B and ρ_{i,T_l}^G can be know from the flow size distribution. Therefore, when given the δ that the maximum good load can be removed and F_i is concave, the equation 4 is a concave optimization problem that has only N variables. The scale of the problem is small and can be solved in a timely manner.

Now, we show how to choose δ. Because we want to reduce the average serving time for good flows, thus $\widehat{W_i^G}$ should be smaller than W_i^G:

$$W_i^G - \widehat{W_i^G} > 0 \tag{5}$$

Substituting equation 2 into it, we can finally get:

$$\alpha < \frac{\frac{1}{1-\sum_{l=1}^{i}\rho_l} - \frac{1}{1-\sum_{l=1}^{i-1}\rho_l+\widehat{\rho_i}}}{\frac{1}{1-\sum_{l=1}^{i+1}\rho_l}} < \frac{\frac{1}{1-\sum_{l=1}^{i}\rho_l} - \frac{1}{1-\sum_{l=1}^{i-1}\rho_l}}{\frac{1}{1-\sum_{l=1}^{i+1}\rho_l}} \tag{6}$$

Therefore, to guarantee we can optimize the performance of good flows, we should choose:

$$\delta = \alpha\rho_i^G < \frac{\frac{1}{1-\sum_{l=1}^{i}\rho_l} - \frac{1}{1-\sum_{l=1}^{i-1}\rho_l}}{\frac{1}{1-\sum_{l=1}^{i+1}\rho_l}}\rho_i^G \tag{7}$$

Therefore, when given the δ, pre-defined thresholds p_i for each queue, and N flow size distributions T_l, we can determine a set of thresholds $p_i^{T_l}$ to adjust the priority of each type of flow. We assume the cumulative density function of all flows is $F(x)$, and $F(x)$ is combined by N types T_i where $F_{T_i}(x)$ is the flow distribution of T_i.

Once we know the cumulative density function of flow size distribution F(x), we can determine a series of thresholds that classify flows into priority queues based on their sent bytes x. For a queue P_i and associated threshold p_i, it should serve flows $p_{i-1} \leq x < p_i$.

V. TESTBED EVALUATION

In this section, we evaluate the performance of TPFS on our small-scale testbed through three experiments. First, we evaluate how the performance of TPFS is influenced by mixing two types of applications. Second, we evaluate the performance of TPFS by mixing four types applications. Finally, we show the CPU overhead of TPFS.

A. EVALUATION SETTINGS

1) TESTBED

We built a small-scale testbed that consists of 8 hosts connected to a Pronto 3290 48-port Gigabit Ethernet switch. Eight hosts running TPFS are Dell OPTIPLEX with a 2-core Intel I3-3220 3.3GHz CPU, 4G memory, and a Broadcom BCM5719 NIC. The default OS is Ubuntu 16.04 64 bit version with Linux 4.4.15 kernel. The base round-trip-time in our testbed is around 200μs. Note that, the default ECN marking scheme is the per-queue marking in PIAS. However, since our hardware switch only supports the per-port ECN marking, we set the ECN marking threshold of each queue to 4KB (a half of the queue size) and use 8 queues.

Scheme compared: We evaluate the following schemes:

- **pFabric:** pFabric perfectly mimics SJF and can achieve close to theoretically minimal latency over a variety of workloads. Through comparing TPFS with pFabric, we can inspect how information-agnostic affects the performance and the performance gap between our schemes and the optimal. Because pFabric needs special hardware to support, we only compare it in simulations.
- **PIAS:** We use the PIAS software implemented in NETFILTER [28] and use the settings in the software. Besides, we choose the DCTCP as our transport protocol as PIAS does.
- **TPFS:** TPFS is implemented according to the design described above in OpenVSwitch 2.6.0 [29]. However, we find that OpenVSwitch will add additional 20μs RTT to our testbed. To fairly compare with PIAS, we also implement TPFS built on the PIAS software in NETFILTER. For convenient, we set $\alpha_i < \frac{\delta}{\rho_i^G} < 10\%$ for all loads.

2) WORKLOAD

We use four realistic workloads: the web search workload [1], the data mining workload [23], the cache workload [21], and the Hadoop workload [21] as shown in Figure 3. To generate flows, 7 hosts run web search, data mining, cache, Hadoop applications on different transport ports, then client running on one host randomly and evenly sets up TCP connections to the applications running on the other hosts and requests flows. The flows are requested using a Poisson arrival process whose parameter is chosen depending on desired link load. Note that we only emulate some features of flows due to lack the datasets. We found that most of the datasets used by the work of flow classification are not publicly released while public datasets lack the flow-level and detail information [13], [14], [21]. As a result, we can only emulate the difference of flows in transport ports and flow size.

3) METRIC

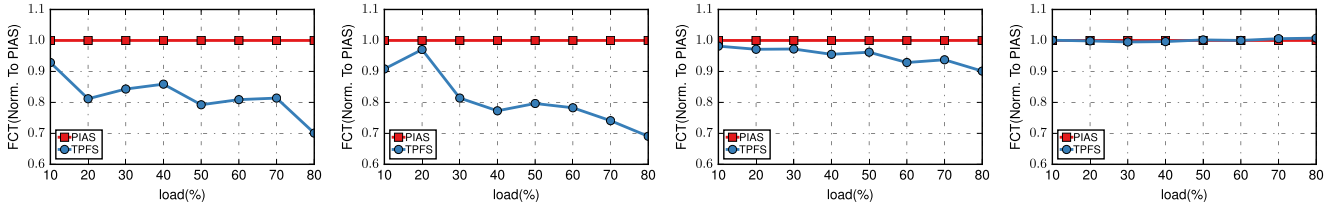
Our main performance metric is the FCT normalized to the PIAS. Besides, we break down the FCT stats across small (0,100KB] and large (10MB,∞) flows, and medium (100KB, 10MB] flows. In order to show tail latency, we also compare the 99th FCT of short flows under different mechanisms.

B. EVALUATION RESULTS

1) MIXING TWO APPLICATIONS

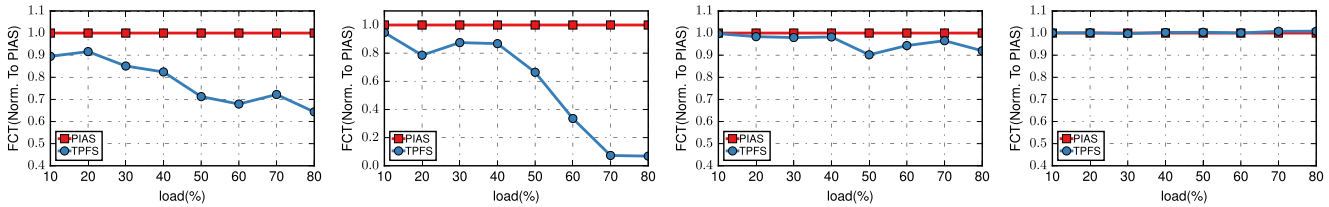
In this evaluation, we let the client randomly requests flows from web search and data mining workload, and evaluate the performance of TPFS and PIAS by varying the average traffic load from 0.1 to 0.8.

From the evaluation results show in Figure 7, we can see that TPFS achieves the best performance, reduces the FCT for short flows by up to 30% and reduces the FCT for 99th short flows by up to 32%. The above performance improvements are expected, since TPFS improves the thresholds scheme of PIAS and uses more proper and accurate thresholds. The above results also indicate that predicting flow size by leveraging the flow information is needed to improve the performance. Besides, TPFS also improves the performance for medium flows and achieves up to 10% lower FCT. The performance of TPFS for large flows is a little worse than PIAS that is too small to see in the figure. This is because that TPFS removes bad load from higher priority queues to lower



(a) Avg.($< 100K$) (b) 99th.($< 100K$) (c) Avg.($100K \sim 10M$) (d) Avg.($> 10M$)

FIGURE 7. [Testbed] The impact of mixing web search and data mining traffic.



(a) Avg.($< 100K$) (b) 99th.($< 100K$) (c) Avg.($100K \sim 10M$) (d) Avg.($> 10M$)

FIGURE 8. [Testbed] The impact of mixing web search, data mining, cache, hadoop traffic.

priority queues, thus increasing the latency of the large flows. Although the latency of large flows is increased, it will not affect the revenue of providers since latency-sensitive short flows are more important.

Besides the FCT of flows, we also found that our clustering algorithm successfully selects the port as the feature and clusters flows into two types according to their ports. Since the scenario is very simple and there are only two clusters, we don't plot the clustering results.

2) MIXING FOUR APPLICATIONS

In this evaluation, we let the client randomly requests flows from all the four types of workloads.

The evaluation results are shown in Figure 8. Compared with DCTCP, the average and 99th FCT of small flows can be decreased by up to 50% and 90% respectively. We can find that compared to mixing two applications, the performance improvement space increases. This is because that as the number of application types increases, if the range of their flow distributions have few overlaps, we will have more accurate information to predict the flow priority. Thus, this indicates that TPFS may have better performance in the real data center with a large number of applications.

3) OVERHEAD

To show the scalability of TPFS, we measure the CPU overhead of TPFS by setting up multiple simultaneous TCP flows between two hosts running TPFS. The total CPU utilization is measured on the sender using Python *psutil* ad 1 second interval for 30 seconds.

As shown in Figure 10, the CPU overhead of TPFS are compared to the CPU overhead of OpenVSwitch(OVS) without TPFS module. The evaluation results show that during the 30 seconds, the difference of average CPU overhead between TPFS and OVS is within 3% as the increasing of concurrent flows. Since the concurrent flows in data centers on one machine are small, TPFS will have a good scalability in real deployment, and the CPU overhead is negligible.

VI. SIMULATION

In this section, we evaluate the performance of TPFS through ns-2 simulator. First, we evaluate the performance of TPFS in large-scale scenarios. Second, we show the impact of the threshold mismatch in TPFS and further show it is necessary to identify the type of traffic to further improve the performance.

A. SIMULATION SETTINGS

1) SCHEME COMPARED

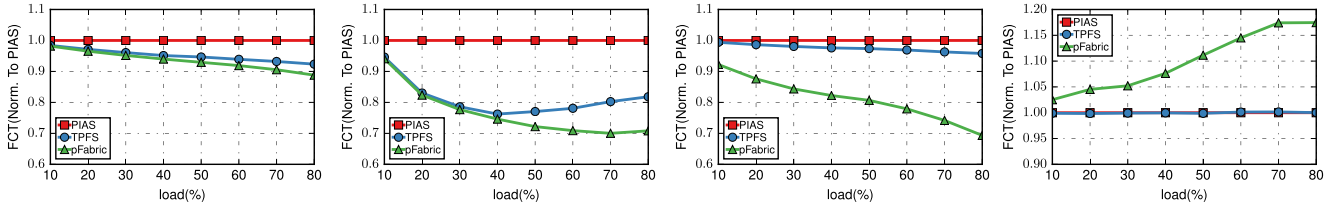
We compare TPFS with pFabric and PIAS. We use per-queue ECN marking schemes and set the queue size to be 240 packets and ECN marking threshold to be 65 packets used by PIAS.

2) TOPOLOGY

We use leaf-spine topology to evaluate the performance of TPFS. The network has 144 hosts, 9 leaf switches, and 4 spine switches. All hosts are connected by 10Gbps links with $20.2\mu s$ delays, and leaf switches are connected to spine switches by 40Gbps links with $0.2\mu s$ delays. Therefore, the oversubscription of the network is 1:1 and the end-to-end round-trip latency across spines is about $85.2\mu s$ [11].

3) WORKLOAD

To generate flows, each server sets up TCP connections to the other servers and requests flows whose flow sizes are randomly selected from the four workloads. The flows are



(a) Avg.($< 100K$) (b) 99th.($< 100K$) (c) Avg.($100K \sim 10M$) (d) Avg.($> 10M$)

FIGURE 9. [Simulation] The impact of mixing web search, data mining, cache, hadoop traffic.

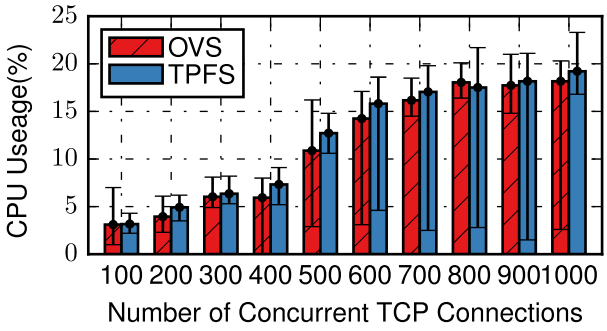


FIGURE 10. The CPU usage for OVS and TPFS on one machine.

requested using a Poisson arrival process whose parameter is chosen depending on desired link load.

B. SIMULATION RESULTS

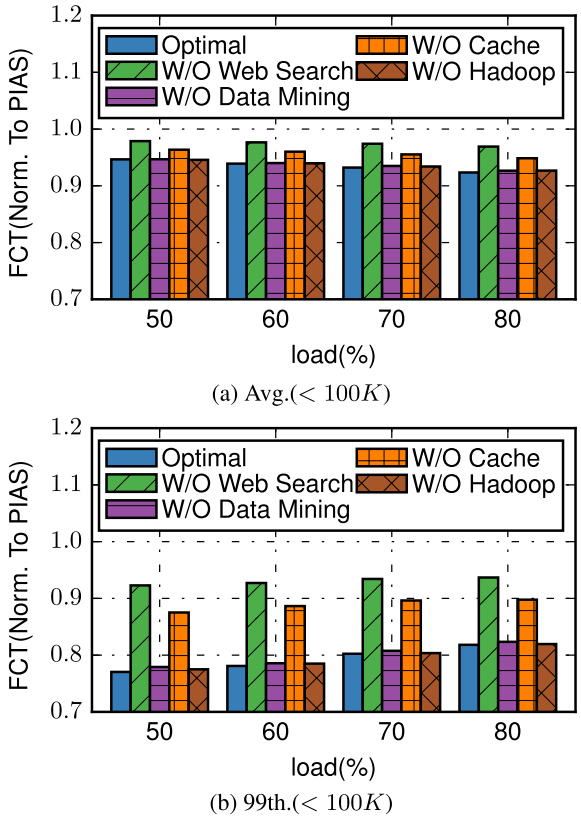
1) MIXING FOUR APPLICATIONS IN LARGE SCALE

In this simulation, we mix all the four types of workloads and evaluate the performance of TPFS in large-scale simulations.

Figure 9 shows the simulation results. Compared to PIAS, both TPFS and pFabric reduce the average FCT of short flows and greatly reduce the 99th FCT of short flows. Compared to pFabric, TPFS has a little worse performance within 3% for short flows, and TPFS has very similar performance with pFabric for 99th FCT of short flows when the load is smaller than 0.4. This indicates TPFS can achieve near-optimal performance when the load is light. When the load is heavy, we find that the performance gap between TPFS and pFabric becomes larger for 99th short flows and medium flows. This is that because pFabric sacrifices the performance of large flows while TPFS still gives some large flows that can't be identified immediately higher priority. However, while pFabric achieves the best performance for short and medium flows, it also has the worst performance for large flows.

2) IMPACTS OF THRESHOLD MISMATCH

In TPFS, threshold mismatch occurs when the features of a application suddenly change. For example, when a user changes the port for web search applications. TPFS adjusts the thresholds periodically to improve this problem. In this simulation, we evaluate how this problem impacts the performance once it happens. We repeat the previous simulation by



(a) Avg.($< 100K$)

(b) 99th.($< 100K$)

FIGURE 11. The impacts of threshold mismatch when TPFS cannot identify(W/O) one of four types of applications.

suddenly changing the transport ports of one of the four types of applications, respectively. Thus, TPFS cannot identify the type of the port-changed application and will use the default pre-defined thresholds to adjust its priority.

The simulation results are shown in Figure 11. Although TPFS has sub-optimal performance when it cannot identify some applications, it also has a better performance than PIAS. Specifically, we can see that when TPFS cannot identify web search and cache workloads, the performance reduces by up to 12% and 10% for 99th short flows, respectively. When TPFS cannot identify data mining and hadoop workloads, the performance reduces a little. This indicates that the impacts of threshold mismatch are mainly affected by the type of workloads. Since more than 80% of flows of data mining

and hadoop workloads are smaller than 100 KB, they have little impacts on the performance. In contrast, only about 50% of flows of web search and cache workloads are smaller than 100 KB, they will generate a large number of bad loads in the highest priority queue by using the default thresholds. Besides, this simulation also shows that it is necessary to identify the type of flows and remove bad loads, which are the motivation of this work.

VII. CONCLUSION

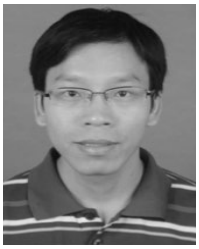
This article validated that the traffic is predictable in data centers, and there is a large improvement space for the pure information-agnostic scheduling. Based on this, we verified that the performance of information-agnostic scheme could be further improved by leveraging the unique characteristics of different traffic types. Thus, we have presented TPFS that predicts flow size to further reduce the completion time of flows. Testbed and simulation results demonstrated that TPFS could greatly reduce the completion time of small flows and outperformed the previous information-agnostic flow scheduling algorithm PIAS.

REFERENCES

[1] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan, "Data Center TCP (DCTCP)," in *Proc. ACM SIGCOMM*, 2011, pp. 63–74.
[2] B. Vamanan, J. Hasan, and T. N. Vijaykumar, "Deadline-aware datacenter tcp (D2TCP)," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 42, no. 4, pp. 115–126, Sep. 2012.
[3] M. Alizadeh, A. Kabbani, B. Prabhakar, A. Vahdat, and M. Yasuda, "Less is more: Trading a little bandwidth for ultra-low latency in the data center," in *Proc. USENIX NSDI*, 2012, pp. 253–266.
[4] M. Alizadeh, S. Yang, M. Sharif, S. Katti, N. McKeown, B. Prabhakar, and S. Shenker, "PFabric: Minimal near-optimal datacenter transport," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 43, no. 4, pp. 435–446, Sep. 2013.
[5] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," *Commun. ACM*, vol. 51, no. 1, pp. 107–113, Jan. 2008.
[6] R. Mittal, D. Zats, V. T. Lam, N. Dukkipati, E. Blem, H. Wassel, M. Ghobadi, A. Vahdat, Y. Wang, and D. Wetherall, "TIMELY: RTT-based congestion control for the datacenter," in *Proc. ACM Conf. Special Interest Group Data Commun. (SIGCOMM)*, 2015, pp. 537–550.
[7] K. He, E. Rozner, K. Agarwal, Y. J. Gu, W. Felter, J. Carter, and A. Akella, "AC/DC TCP: Virtual congestion control enforcement for datacenter networks," in *Proc. Conf. ACM SIGCOMM Conf. (SIGCOMM)*, 2016, pp. 244–257.
[8] B. Cronkite-Ratcliff, A. Bergman, S. Vargaftik, M. Ravi, N. McKeown, I. Abraham, and I. Keslassy, "Virtualized congestion control," in *Proc. Conf. ACM SIGCOMM Conf. (SIGCOMM)*, 2016, pp. 230–243.
[9] C. Wilson, H. Ballani, T. Karagiannis, and A. Rowtron, "Better never than late: Meeting deadlines in datacenter networks," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 41, no. 4, p. 50, Oct. 2011.
[10] C.-Y. Hong, M. Caesar, and P. B. Godfrey, "Finishing flows quickly with preemptive scheduling," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 42, no. 4, pp. 127–138, Sep. 2012.
[11] W. Bai, L. Chen, K. Chen, D. Han, C. Tian, and W. Sun, "PIAS: Practical information-agnostic flow scheduling for data center networks," in *Proc. USENIX NSDI*, 2015, pp. 1–7.
[12] L. Chen, K. Chen, W. Bai, and M. Alizadeh, "Scheduling mix-flows in commodity datacenters with Karuna," in *Proc. Conf. ACM SIGCOMM Conf. (SIGCOMM)*, 2016, pp. 174–187.
[13] T. T. T. Nguyen and G. Armitage, "A survey of techniques for Internet traffic classification using machine learning," *IEEE Commun. Surveys Tuts.*, vol. 10, no. 4, pp. 56–76, Oct. 2008.
[14] L. Bernaille, R. Teixeira, I. Akodkenou, A. Soule, and K. Salamatian, "Traffic classification on the fly," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 36, no. 2, p. 23, Apr. 2006.
[15] H. Zhang, L. Chen, B. Yi, K. Chen, M. Chowdhury, and Y. Geng, "CODA: Toward automatically identifying and scheduling coflows in the dark," in *Proc. Conf. ACM SIGCOMM Conf. (SIGCOMM)*, 2016, pp. 160–173.
[16] M. Chowdhury and I. Stoica, "Coflow: A networking abstraction for cluster applications," in *Proc. 11th ACM Workshop Hot Topics Netw (HotNets)*, 2012, pp. 31–36.
[17] M. Chowdhury and I. Stoica, "Efficient coflow scheduling without prior knowledge," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 45, no. 4, pp. 393–406, Sep. 2015.
[18] V. Shrivastava, P. Zerfos, K.-W. Lee, H. Jamjoom, Y.-H. Liu, and S. Banerjee, "Application-aware virtual machine migration in data centers," in *Proc. IEEE INFOCOM*, Apr. 2011, pp. 66–70.
[19] M. F. Bari, R. Boutaba, R. Esteves, L. Z. Granville, M. Podlesny, M. G. Rabbani, Q. Zhang, and M. F. Zhani, "Data center network virtualization: A survey," *IEEE Commun. Surveys Tuts.*, vol. 15, no. 2, pp. 909–928, 2nd Quart., 2013.
[20] *PIAS Simulation Code*. Accessed: Jun. 2, 2019. [Online]. Available: https://github.com/HKUST-SING/PIAS-NS2
[21] A. Roy, H. Zeng, J. Bagga, G. Porter, and A. C. Snoeren, "Inside the social Network's (Datacenter) network," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 45, no. 4, pp. 123–137, Sep. 2015.
[22] P. Poupart, Z. Chen, P. Jaini, F. Fung, H. Susanto, Y. Geng, L. Chen, K. Chen, and H. Jin, "Online flow size prediction for improved network routing," in *Proc. IEEE 24th Int. Conf. Netw. Protocols (ICNP)*, Nov. 2016, pp. 1–6.
[23] A. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. A. Maltz, P. Patel, and S. Sengupta, "VL2: A scalable and flexible data center network," in *Proc. ACM SIGCOMM Conf. Data Commun. (SIGCOMM)*, 2009, pp. 51–62.
[24] *VMware NSX Doc*. Accessed: Jun. 2, 2019. [Online]. Available: https://www.vmware.com/products/nsx.html
[25] *OpenStack Neutron Doc*. Accessed: Jun. 2, 2019. [Online]. Available: https://wiki.openstack.org/wiki/Neutron
[26] Y. Sun, X. Yin, J. Jiang, V. Sekar, F. Lin, N. Wang, T. Liu, and B. Sinopoli, "CS2P: Improving video bitrate selection and adaptation with data-driven throughput prediction," in *Proc. Conf. ACM SIGCOMM Conf. (SIGCOMM)*, 2016, pp. 272–285.
[27] D. Birant and A. Kut, "ST-DBSCAN: An algorithm for clustering spatial–temporal data," *Data Knowl. Eng.*, vol. 60, no. 1, pp. 208–221, Jan. 2007.
[28] *PIAS-Software*. Accessed: Jun. 2, 2019. [Online]. Available: https://github.com/HKUST-SING/PIAS-Software
[29] *OpenvSwitch*. Accessed: Jun. 2, 2019. [Online]. Available: http://openvswitch.org

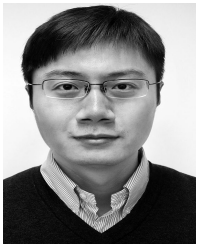
SHUO WANG received the B.S. degree in communication engineering from Zhengzhou University, China, in July 2013, and the Ph.D. degree in information and communication engineering from the Beijing University of Posts and Telecommunications, China, in July 2018. He has been a Postdoctoral Researcher with the Beijing University of Posts and Telecommunications, since 2018. His research interests include data center networking and software-defined networking.

JIAO ZHANG received the B.S. degree from the School of Computer Science and Technology, Beijing University of Posts and Telecommunications (BUPT), in July 2008, and the Ph.D. degree from the Department of Computer Science and Technology, Tsinghua University, China, in July 2014. From August 2012 to August 2013, she was a Visiting Student with the Networking Group of ICSI, UC Berkeley. She is currently an Associate Professor with the School of Information and Communication Engineering, and the State Key Laboratory of Networking and Switching Technology, BUPT. She has (co)authored more than 30 international journal and conference papers. Her research interests include traffic management in data center networks, software-defined networking, network function virtualization, the future Internet architecture, and routing in wireless sensor networks.



TAO HUANG received the B.S. degree in communication engineering from Nankai University, Tianjin, China, in 2002, and the M.S. and Ph.D. degrees in communication and information system from the Beijing University of Posts and Telecommunications, Beijing, China, in 2004 and 2007, respectively. He is currently a Professor with the Beijing University of Posts and Telecommunications. His current research interests include network architecture, routing and forwarding, and network virtualization.

TIAN PAN received the B.S. degree from Northwestern Polytechnical University, Xi'an, China, in 2009, and the Ph.D. degree from the Department of Computer Science and Technology, Tsinghua University, Beijing, China, in 2015. He has been a Postdoctoral Researcher with the Beijing University of Posts and Telecommunications, since 2015. His research interests include router architecture, network processor architecture, network power efficiency, and software-defined networking.



JIANG LIU received the B.S. degree in electronics engineering from the Beijing Institute of Technology, China, in 2005, the M.S. degree in communication and information systems from Zhengzhou University, China, in 2009, and the Ph.D. degree from the Beijing University of Posts and Telecommunications (BUPT), in 2012. He is currently an Associate Professor with BUPT. His current research interests include network architecture, network virtualization, software-defined networking, information-centric networking, and platforms for networking research and teaching.

YUNJIE LIU received the B.S. degree in technical physics from Peking University, Beijing, China, in 1968. He is currently an Academician of the China Academy of Engineering, the Chief of the Science and Technology Committee of China Unicom, and the Dean of the School of Information and Communication Engineering, Beijing University of Posts and Telecommunications. His current research interests include next generation networks, network architecture, and management.