

Scheduling with Machine-Learning-Based Flow Detection for Packet-Switched Optical Data Center Networks

Lin Wang, Xinbo Wang, Massimo Tornatore, Kwang Joon Kim, Sun Me Kim, Dae-Ub Kim, Kyeong-Eun Han, and Biswanath Mukherjee

Abstract—A scalable, low-latency, high-speed, and energy-efficient data center network is a key element in the deployment of future large-scale data centers, and photonic switching has recently been recognized as a promising solution to fulfill these goals. In this study, we present a packet-switched optical network (PSON) architecture with centralized control for intra-data-center connectivity. For efficient PSON operation, intelligent yet low-complexity bandwidth-scheduling algorithms are critical. To align with realistic traffic flows in a data center, we consider “mice flow,” which occurs frequently but carries a small number of bytes, and “elephant flow,” which occurs occasionally but has a huge number of bytes. To classify traffic flows with different characteristics, we investigate various machine-learning (classification) techniques, such as C4.5 and Naïve Bayes Discretization, and compare their performance in terms of accuracy and classification speed. We also develop a priority-aware scheduling algorithm for packet switching, which is optimized for PSON, and is adaptive to flow classification under a dynamic traffic scenario. Numerical simulations show that our proposed scheduling algorithm assisted by flow-classification techniques can outperform a benchmark algorithm in terms of average delay and packet-loss ratio.

Index Terms—Data center; Machine learning; Packet-switched optical network; Scheduling; Traffic classification.

I. INTRODUCTION

The global data center traffic has been increasing at an annual rate of 31% and will reach 3.3 ZB per year, or 278 EB per month, by 2021, and 76% of this traffic resides within the data center [1]. As current intra-data-center networks still rely heavily on electrical switches, data center operators are facing challenges related to high power consumption, limited scalability, and high latency inside their data centers. For example, electrical data center switches are a major contributor of data center

Manuscript received July 31, 2017; revised December 26, 2017; accepted January 19, 2018; published March 16, 2018 (Doc. ID 303343).

L. Wang (e-mail: amlwang@ucdavis.edu), X. Wang, M. Tornatore, and B. Mukherjee are with the University of California, Davis, California 95616, USA.

M. Tornatore is with Politecnico di Milano, Milano, Italy.

K. J. Kim, S. M. Kim, D.-U. Kim, and K.-E. Han are with the Electronics and Telecommunications Research Institute, Daejeon, South Korea.

<https://doi.org/10.1364/JOCN.10.000365>

power consumption. Considering that current electrical switches consume about 2.5 W/Gbps [2] and that a typical data center needs to support bandwidth on the order of terabytes per second (Tbps) today, the overall energy consumption could reach 2500 W. Moreover, low latency is becoming more crucial for data center networks because a huge volume of data must be exchanged for real-time tasks with rigid latency bounds, e.g., big-data analytics, online machine-learning-model training, etc. Hence, the current data center network needs to evolve towards a high-speed, scalable, low-latency, and energy-efficient network platform.

Optical networks have played an important role in large-scale backbone networks for a long time. The authors in Ref. [3] showed the flexibility of optical networks to provide dynamic routing in large-scale multilayer and multi-domain networks. Reference [4] demonstrated that optical interconnection is promising to meet the high-burstiness and high-bandwidth requirements of services between data centers. Reference [5] demonstrated the ability of optical networks to improve the efficiency of network infrastructure via network virtualization. Besides inter-data-center connection, optical networks can represent a valuable alternative for electrical intra-data-center networks, thanks to their combined usage of advanced technologies, such as tunable transmitters, passive optical components, etc. For example, arrayed waveguide grating routers (AWGR) plus fast-tuning lasers can significantly reduce the power consumption in data center networks while ensuring low latency, suggesting that photonic-switch-based architectures are promising candidates for future intra-data-center networks. Also, WDM techniques can be used to avoid head-of-line blocking for photonic switches, which can result in better performance in terms of latency and packet-loss rate.

Several optical data center architectures have been reported [6–10]. They employ different interconnection structures (rearrangeable, non-blocking, multistage Benes, Banyan, etc.), and typically rely on centralized control. To change their interconnection patterns, they need a certain reconfiguration time, during which the switch is unable to handle data, thus resulting in packet loss. The reconfiguration time complexity increases proportionally to $N \log 2N$, where N is the number of input and output ports [7]. Our previous work [11] introduced an optical data

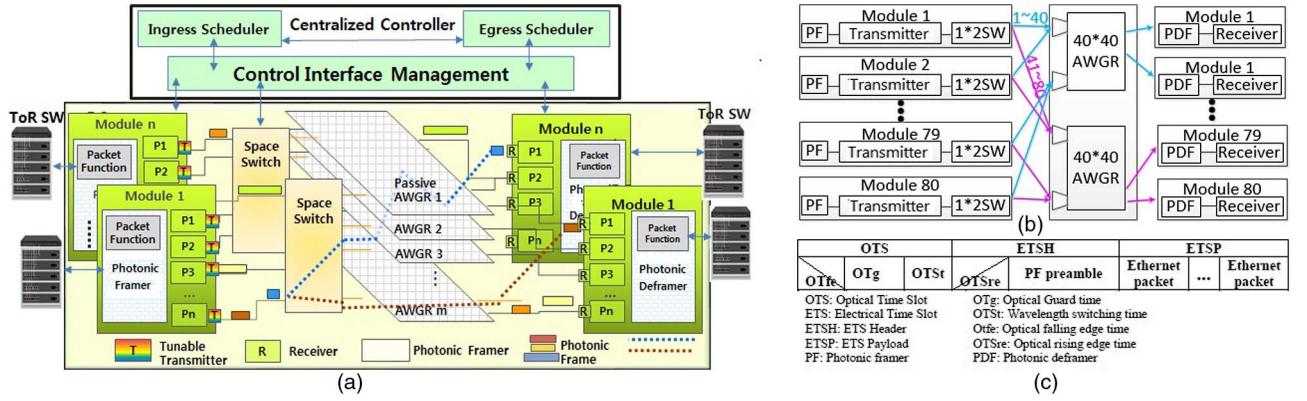


Fig. 1. (a) PSON architecture. (b) PSON data plane (with optical switch fabric). (c) Photonic frame format.

center network architecture, called a packet-switched optical network (PSON), which has only one stage of switching as shown in Fig. 1(a) (more details will be provided in Section II). Also, the data plane and control plane are decoupled to simplify scheduling and enhance scalability.

To achieve high throughput and low latency in PSON, an intelligent yet low-complexity scheduling algorithm is critical. The round robin (RR) scheduling algorithm [12–14] is widely used in resource schedulers because it is simple and fast enough to match the speed of photonic switches. But the RR algorithm fails to consider the traffic-flow characteristics in a data center [15,16], resulting in high packet-loss rate (PLR) and high latency. Other modifications to RR, which are intended to work under a distributed control [17], need a lot of signaling exchange and result in unnecessary latency.

Authors in Refs. [13,18] show that a large fraction of data center traffic is carried in a small fraction of flows: 90% of the flows, called mice flows, carry less than 1 MB of data, while more than 90% of bytes are transferred in flows greater than 100 MB, called “elephant flows.” Mice flows usually carry small-size packets with short duration and high time sensitivity (e.g., transactional traffic, web browsing, search queries). Elephant flows generally carry large-size packets, transmit a huge amount of bytes, and have long-lasting duration. They are usually due to bulk data transfer, data backup, virtual machine migration, etc.

In traditional data center networks, bandwidth-allocation schemes either consider number of packets or total size of packets waiting in the queue, without considering mice and elephant flows’ different requirements. This might degrade performance for the entire network. For example, if we only consider the size of all packets in a queue, we may prioritize a queue containing elephant flows due to their large volume; consequently, some time-sensitive mice flows might incur long queueing delays, and the application running over these mice flows might incur a timeout. On the other hand, if we only consider total number of packets in each queue, we might delay elephant flows for the opposite reason. Thus, an effective scheduling algorithm should treat traffic flows differently based on their type (mice or elephants) and assign bandwidth resources accordingly. To achieve this traffic-aware behavior, the scheduling

algorithm must be enhanced with the capability to classify mice and elephant flows. We study three classification methods for this purpose, viz. Mahout threshold, Naïve Bayes discretization (NBD), and the C4.5 decision tree, and we compare their performance in terms of classification accuracy and speed.

By leveraging the knowledge of traffic characteristics, we then propose a novel priority-aware (PA) scheduling algorithm that exploits information on traffic classification as well as on total length of buffered packets, number of buffered packets, and arrival time of the earliest packet in the buffer. We use different weight factors to reflect how much these three pieces of information could affect the performance of PSON. Our PA algorithm also considers the space-switch tuning time to connect a $1 \times m$ space switch to a certain AWGR (see next section). Simulation experiments show that our PA algorithm outperforms the traditional RR algorithm in terms of PLR and average delay. Furthermore, specific evaluations are conducted on PSON to study the impact on performance of buffer size and traffic-flow characteristics.

The rest of the study is organized as follows: Section II introduces the PSON architecture and photonic frame format. Section III describes the data center traffic model and the three classification methods to detect elephant flows. Section IV illustrates the PA algorithm implemented in PSON. In Section V, we study the three classification methods, investigate the performance of the PA algorithm, and compare it with RR algorithm through simulation. Section VI concludes this study.

II. PACKET-SWITCHED OPTICAL NETWORK ARCHITECTURE

We present the PSON architecture in Fig. 1(a). The core of PSON architecture is a set of AWGRs, which are passive and high-speed optical devices that can resolve contentions in the wavelength domain [8]. In PSON, multiple servers are connected to a top-of-rack (ToR) switch, and n such ToR switches are interconnected by the AWG-based core-switching network. Each ToR switch connects to an ingress module and an egress module that contain a photonic framer and de-framer, respectively. An ingress module

contains n virtual output queues (VoQs), each corresponding to a ToR switch as destination. The framer/de-framer manages the wrapping/unwrapping of Ethernet packets into photonic frames. An optical transmitter uses a fast tunable-wavelength light source to transmit photonic frames from the ingress module. Each ingress module is also equipped with a $1 \times m$ space switch to connect with the m AWGRs, and an AWGR exclusively delivers photonic frames to n/m egress modules.

We enhance the scalability of PSON by equipping each ToR with a $1 \times m$ space switch so that the number of switch ports can be easily enlarged without considerably increasing latency. Figure 1(b) shows an example of applying 1×2 space switches to increase the number of supported ToR switches from 40 to 80. PSON decouples the control plane and data plane by centralizing control logic from ToRs to a dedicated controller so that ToR is simplified as pure executor of packet switching, while the controller can efficiently utilize its computing resource to schedule photonic-frame switching. Since ToR does not mind sending signaling packets to each other anymore, sophisticated scheduling decisions can be made on a timely manner to reduce switching latency, and precious switching bandwidth can be saved for serving customers' data.

To better understand PSON architecture, we explain below how packets flow from one server to another. In the ingress side, a server generates native electrical Ethernet/IP packets, which are collected by a ToR switch and sent to an associated ingress module. An electrical packet will be placed in a VoQ that is associated to the destined egress module, waiting for photonic wrapping and switching. For each ingress module, the centralized controller must decide which VoQ can wrap the packets into a photonic frame [see format in Fig. 1(c)], because only one VoQ can transmit a packet at a time. When an ingress module is allocated with a transmission grant from control plane, it will send a photonic frame to the associated space switch in the next time slot. Meanwhile, the space switch must be configured to connect with the right AWGR that is responsible for delivering the packets to the destined egress module. For example, in Fig. 1(b), if Module 1 wants to send packets to Module 80, the space switch of Module 1 must be configured to connect with the lower AWGR, which is connected with the Module 80 by design. When packets arrive at the destined egress module, inverse operations are performed so that they can be delivered to the target server.

Packet delivery is straightforward between two servers, but it becomes intangible when hundreds of servers need to be scheduled. A critical challenge is that, given a time slot, an ingress module must decide which VoQ can transmit packets, and an egress module must decide from which ingress module it should receive the frame. Without proper scheduling, conflict can occur when multiple ingress modules want to simultaneously deliver frames to the same egress module. Due to a conflict, no transmission will happen during this time slot, which wastes bandwidth resources. To address the challenge, an efficient scheduling algorithm is needed for PSON.

III. TRAFFIC CHARACTERISTICS AND FLOW DETECTION

A. Traffic Characteristics

In traditional resource-scheduling methods, all flows are treated equally without differentiation. However, as described in the Introduction, intra-data-center traffic consists of mice flows and elephant flows, and they have very different characteristics. Studies [15,16] show that packet length of intra-data-center traffic follows a bimodal distribution at 40 bytes and 1500 bytes in real scenarios, which matches the Ethernet minimum and maximum lengths in other network environments [18]. Figure 2 shows the cumulative distribution function (CDF) and histogram of packet sizes generated during simulations [15]. Reference [19] proposes hash-based flow-forwarding techniques that work well for traffic scenarios with large numbers of mice flows and without elephant flows. Reference [20] shows that managing elephant flows effectively can yield 113% higher aggregate throughput compared to the work in Ref. [19].

An efficient scheduling algorithm should be aware of the diversity of flows and allocate resources to a flow in a timely manner. Therefore, a real-time flow-detection method is the premise of an effective resource-allocation algorithm.

B. Traffic-Flow Detection Methods

Traditional flow-detection methods are based on in-network monitoring/sampling, because the network behavior of a flow is affected by how rapidly the end-point applications generate data, and this is not biased by congestion in the network. In modern data centers controlled by scalable

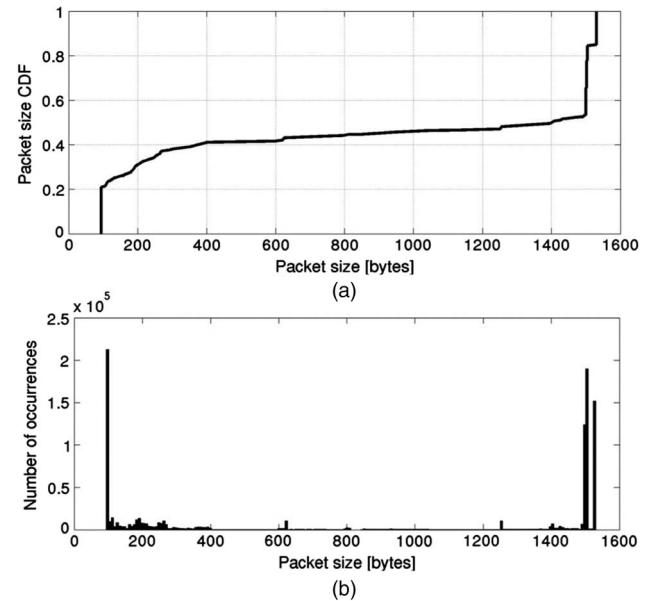


Fig. 2. Generated packet sizes. (a) CDF. (b) Histogram.

operating systems (OS) with uniform software on a single administrative domain, the end-host OS has better visibility into the behavior of applications, in contrast to in-network monitors. Reference [21] proposed an end-host-based detection method, called Mahout, which is a lightweight algorithm that can achieve better performance than in-network monitoring methods with respect to detecting elephant flow of intra-data-center traffic. Mahout monitors end-host socket buffers and uses a threshold value to detect the existence of elephant flow. But the performance of Mahout is highly dependent on the choice of threshold value, and a “one-size-fits-all” threshold does not work well under various traffic conditions in a data center. So, a flexible method is expected to better detect elephant flow in data centers.

In this study, we propose to use machine learning (ML) to detect flows in each module of PSON, for the following reasons. First, elephant flow has more noticeable features to capture in practice, e.g., large-size packets, huge amount of bytes, long-lasting duration. Second, ML has powerful techniques that uncover hidden insights through learning from useful structural patterns in data. Recently, we see increased interest in the development of ML techniques for traffic classification in computer networks. Five popular ML techniques are studied for IP traffic classification in Ref. [22]. State-of-the-art traffic classifications using ML algorithms are also well studied in Ref. [23].

Unsupervised and supervised learning are two main popular ML technique types applied in traffic classification. Unsupervised learning discovers natural clusters (groups) in the data using internalized heuristics [23]. Unsupervised learning, e.g., *K*-Means, density-based spatial clustering of applications with noise (DBSCAN), and AutoClass [23], focuses on finding patterns in the input data. It clusters instances with similar properties (defined by a specific distance-measuring approach, such as Euclidean space) into groups. The groups that are so identified may be exclusive so that any instance belongs in only one group; or they may be overlapping, when one instance may fall into several groups; or they may also be probabilistic, i.e., an instance belongs to a group with a certain probability. Unsupervised learning can model the underlying structure or distribution in the data so that the algorithm can evolve by itself to discover interesting structures in the data.

In contrast, supervised learning creates knowledge structures that support the task of classifying new instances into predefined classes. The learning machine is provided with a collection of sample instances, preclassified into classes. In effect, supervised learning focuses on modeling the input/output relationships. Its goal is to identify a mapping from input features to an output class. There are two major phases in supervised learning, i.e., training and testing. The training phase examines the provided data, called the training data set, and constructs a classification model. Testing uses the model that has been built in the training phase to classify new instances. Data sets used in training and testing phases must be independent and classified in advance, known as “labeling.”

Popular supervised learning algorithms include Naïve Bayes Tree, C4.5, Bayesian Network, Naïve-Bayes discretization (NBD), and Naïve Bayes kernel density estimation (NBK) [22,23].

In this study, we want to use an ML algorithm to identify elephant flows, which is a pre-known class. Therefore, supervised ML is more suitable for our use case.

Existing research works on traffic classification focus on improving classification accuracy of different ML algorithms. However, in a data center environment, where scalability is an issue, the classification method should not only be accurate but also have good computational performance, e.g., small building time and high speed of classification. For example, in Ref. [22], besides classification accuracy, the authors evaluate building time and classification speed of five ML techniques for IP traffic classification, i.e., Bayesian Network, C4.5 Decision Tree (C4.5), NBD, NBK, and Naïve Bayes Tree. According to Refs. [22,23], C4.5 and NBD can achieve the fastest speed and acquire good accuracy classification results. Besides, both of them only ask for low computation requirements, which is very suitable for our PSON architecture. Because PSON has a centralized controller, considering the increasing scalability requirement, we want classifiers to be fast and efficient to handle a large amount of data. So our study implements C4.5 and NBD to detect elephant flows. Besides, our ultimate goal is not to improve existing ML algorithms for traffic classification, but to evaluate how traffic classification using ML algorithms can benefit an optical data center, e.g., PSON. This is the reason why we are not testing our algorithm against other state-of-the-art ML algorithms. Instead, we want to show that traffic classification using ML is helpful in improving the performance of data center networks.

C4.5 creates a model based on a tree structure [24]. Nodes in the tree represent features, with branches representing possible values connecting features. A leaf node that terminates a series of nodes and branches represents a class. C4.5 builds decision trees from a set of training data, using the concept of information entropy, which is the average amount of uncertainty produced by data, i.e., the more possibility an event can happen, the less uncertainty this event can provide. At each node of the tree, C4.5 chooses the attribute that can effectively split the set of samples into subsets so that we can acquire the smallest uncertainty of samples. C4.5 then recursively chooses the most efficient unused attribute on new subsets. C4.5 requires data to have efficient features so that future unknown data may be identified and differentiated from these features. So, C4.5 is only suitable for data sets that are labeled with features that can be used to split the data set into multiple subsets.

NBD is based on the Bayesian theorem [25]. This classification technique analyzes the relationship between each attribute and the class for each instance to derive a conditional probability for the relationships between the attribute values and the class. Naïve Bayesian classifiers must estimate the probabilities of a feature having a certain value. Continuous features can have a large (possibly

infinite) number of values, and the probability cannot be estimated from the frequency distribution. This can be addressed by modeling features with a continuous probability distribution or by using discretization (which transforms the continuous features into discrete features, and a distribution model is not required).

Considering the importance of feature selection to apply the C4.5 decision tree and NBD, we define features as follows. When defining flow features, the “kitchen-sink” method of using as many features as possible is eschewed in favor of an economical, constraint-based approach. The main limitation in choosing features is that calculation should be realistically possible within a resource-constrained data center network device. So, potential features need to fit the following criteria:

- Packet payload independence
- Context limited to a single flow (i.e., no features spanning multiple flows)
- Simple to compute

The following features are found to match the above criteria [22] and become the base feature set for our experiments:

- Packet length (minimum, mean, maximum, and standard deviation)
- Inter-arrival time between packets (minimum, mean, maximum, and standard deviation)

For periodic real-time flow detection, it is hard to capture the start of each flow. We need an approach to allow detection to be initiated at any point in time when traffic flows are already in progress. Reference [26] proposed a classification based on only the most recent M packets of a flow, called a “classification sliding window.” Use of a small number of packets for classification ensures timeliness of classification and reduces the buffer space required to store packets’ information for the classification process. It offers a potential of monitoring traffic flow during its lifetime in a timely manner, with the constraints of physical resources. In our study, we also apply a classification sliding window to collect flow statistics for our elephant-flow detection methods.

A common way to characterize a classifier’s accuracy is through the metrics True Positives, True Negatives, False Positives, and False Negatives, which are defined as follows:

- True Positives: Percentage of members of class X correctly classified as belonging to class X .
- True Negatives: Percentage of members of other classes correctly classified as not belonging to class X .
- False Negatives: Percentage of members of class X incorrectly classified as not belonging to class X .
- False Positives: Percentage of members of other classes incorrectly classified as belonging to class X .

Our study uses True Positives metric, which is also called “recall,” to evaluate elephant-flow detection methods.

Note that it is less meaningful to evaluate the recall of mice-flow detection, because the percentage of mice flow overall (i.e., 90%) is so high that, even if flow detection does nothing but detect every flow as mice flow, there is still 90% probability that the result is correct. Since there are much fewer elephant flows, it is highly dependent on the performance of the detection method that a flow is correctly detected as elephant flow.

IV. RESOURCE SCHEDULING ALGORITHMS

A. Round-Robin Scheduling Algorithm

The classical RR algorithm [14,15] is widely used in resource schedulers due to its simplicity. In RR, an ingress module will check all non-empty VoQs and request transmission grants from their corresponding egress modules. An egress module collects requests from the ingress modules and takes turns granting transmission (one ingress module in each time slot). Each ingress module also collects grants from egress modules and takes turns accepting grants. However, classical RR ensures absolute fairness among all VoQs, so it may not be suitable for today’s data centers due to the following reasons.

First, in today’s data center application, latency is critical, because a resultant timeout can result in failure of a computing application, especially for time-sensitive mice flows, e.g., transactional traffic, web browsing, search queries, etc., and classical RR does not consider latency needs of applications. Hence, prioritizing transmission of VoQs with time-sensitive packets can improve the performance for data centers. Second, each elephant flow carries a large number of big packets. If we always prefer to transmit mice flows, transmission for elephant flows will be degraded, VoQ buffer will be clogged, and PLR can increase. Thus, a trade-off must be conducted dynamically while scheduling mice flow and elephant flow, based on which we propose an intelligent scheduling algorithm, which is aware of priority of packet transmissions according to flow detection and real-time network condition.

B. Priority-Aware Scheduling Algorithm

We propose a PA scheduling algorithm, which assigns priorities to VoQs based on four possible queueing strategies: longest queue fist (LQF), largest number of packets first (LNPF), oldest packet first (OPF), and less space switch first (LSSF); see Fig. 3 for its flowchart.

First, each ingress module maintains status information and gets priority values for the n VoQs based on their status information. Priority value is calculated as follows:

$$W_{ij} = l_{ij} * w_l + p_{ij} * w_p + d_{ij} * w_d + s_{ij} * w_s, \quad (1)$$

where W_{ij} is priority of VoQ_{ij} (virtual queue at ingress module i associated with egress module j), l_{ij} represents length of VoQ_{ij}, p_{ij} represents number of packets in VoQ_{ij}, d_{ij} represents delay of the earliest packet in VoQ_{ij}, and w_l , w_p ,

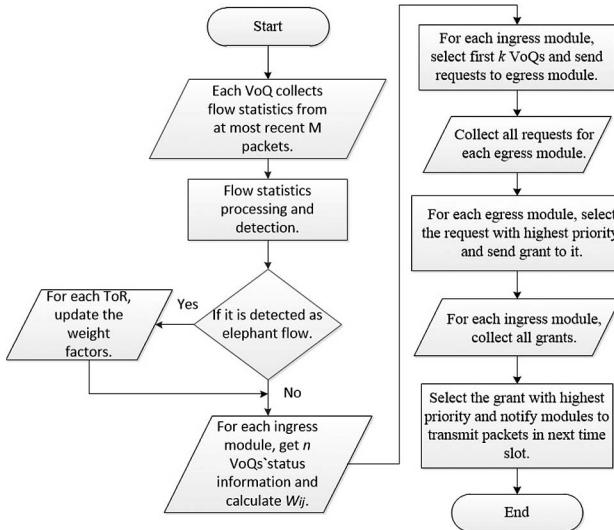


Fig. 3. Flowchart of priority-aware algorithm.

and w_d are weighting factors for them. s_{ij} is a boolean value (taking value 0 if sending a frame of VoQ_{ij} needs tuning of the space switch; 1, otherwise). Note that a space switch can only connect to a particular AWGR i each time slot. For example, in Fig. 1(b), if Module 1 wants to send packets to Module 80, but its space switch is connected with the upper AWGR, then it must be tuned to connect with the lower AWGR. Since tuning time of a space switch is at sub-microsecond speed, we should avoid frequently tuning the space switch and use the LSSF strategy introduced above so that extra s latency can be minimized. In consecutive time slots, the PA algorithm prefers to send packets to egress modules that are connected to the same AWGR.

Second, an ingress module selects the first k non-empty VoQs with highest priorities and sends requests (along with their priority values) to their associated egress modules, instead of sending requests for all VoQs as in the RR algorithm. Then, an egress module collects requests from different ingress modules and selects the request with the highest priority and grants transmission to the ingress scheduler. Finally, the ingress module collects at most k grants from all egress modules. If there are multiple grants for it, the grant for the VoQ with highest priority is accepted. Note that this entire scheduling process of the PA algorithm is executed in a separate centralized controller instead of in each module. Considering the computation and process time for scheduling, it is not practical to execute it for every time slot; instead, we do this schedule periodically (e.g., 10 time slots). Each module only reports the n VoQs' status statistics to the centralized controller, gets the grants (if possible), and prepares to transmit packets in the next period.

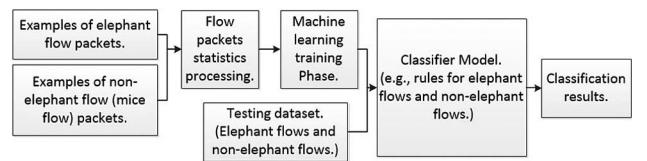
As mentioned before, we should treat elephant flows and mice flows differently. So we should adjust weight factors based on a flow's class. For example, in VoQs where there are mainly mice flows, weight factor w_p should be more important than w_l because the growing number of packets (p_{ij}) in a VoQ does not increase the total packet size (l_{ij})

a lot. If w_p is small and w_l is large, calculated priority of VoQ according to Eq. (1) will be small and result in VoQ having less chance to be transmitted. This causes a longer delay for time-sensitive packets waiting in this VoQ and leads to increased average latency for the network. Vice versa for VoQs that store elephant flows. Thus, we need the algorithm to periodically and dynamically update weight factors for each VoQ in modules to reflect the traffic-flow features.

As the tuning time of space switch is constant and the earliest packet arrival time is random, w_s and w_d do not need to be updated periodically. Instead, total length of packets and number of packets in buffer are closely related to the traffic flow in a data center; thus w_p and w_l are the two major factors to decide the efficiency of the PA algorithm. With this knowledge, the PA algorithm only needs to update w_p and w_l for each VoQ.

In this work, we apply three methods, described in Section III, to detect elephant flows and update w_p and w_l . The first is the Mahout threshold method [21], which is a benchmark for our study. For our case, we modify it by redefining its threshold as a value l_{ij}/p_{ij} , referred to as "PA-threshold", to represent average packet size in VoQ_{ij} . If average packet size is larger than the PA-threshold, i.e., most of the traffic data belongs to elephant flow, we detect it as elephant flow; otherwise, we consider it as mice flow. The other two methods we implement are the C4.5 decision tree and NBD methods; we use a sliding window with size M to collect traffic-flow statistics from M most-recent packets in each VoQ and put processed statistics into the classification model to get identification output as shown in Fig. 4. If a VoQ is determined to contain elephant flows, then we increase the value of w_l and deduct the value of w_p ; vice versa for mice flows (i.e., w_p should increase and w_l descends). When calculating the priority for each VoQ, w_p , w_l , w_d , and w_s will be normalized such that they sum up to 1 before being used as weight factors.

As both the C4.5 decision tree and NBD are able to classify data into multiple types [23,24], we can improve the scalability of the PA algorithm to handle multiple types of traffic. For example, in this work, we provide two weight factors w_l (length of VoQ) and w_p (number of packets in VoQ) to reflect features of two traffic flows: elephant and mice. When we detect elephant flows using the ML algorithm, we increase the value of w_l and decrease the value of w_p , and vice versa when detecting mice flows. Given more types of traffic in the future, e.g., if there is a third traffic type that is time sensitive, we can add time-sensitive weight factor to qualify the third traffic type's time requirement.

Fig. 4. Flowchart of updating w_p and w_l .

V. ILLUSTRATIVE NUMERICAL EXAMPLES

We conduct simulations to 1) compare two flow-detection methods, i.e., the C4.5 decision tree and NBD, benchmarked by modified end-host-based detection method, in terms of recall and classification speed; 2) evaluate the performance of the PA algorithm assisted by three different flow-detection methods, average delay and packet-loss ratio; and 3) study the influence of weight factors w_p and w_l on PSON and impact of electrical buffer dimensioning. We simulate 80 ToR switches communicating with each other, supported by two 40×40 AWGRs, and two 1×2 space switches. Each ToR is connected with 36 servers. The capacity of a wavelength is 10 Gbps. Each module has a buffer shared by all VoQs. The ingress module of a ToR switch creates packets with a size following bimodal distribution at 40 bytes and 1500 bytes (see Section III.A). These packets are injected into the system with an inter-arrival time following Pareto distribution. Specifically, the periods of packet generation are modeled by matching ON/OFF periods, i.e., with/without packets transmission, which simulates traffic behavior found in real data centers [18]. The lengths of these ON/OFF periods are characterized by heavy-tailed random distributions as shown in Fig. 5.

Simulations are conducted with different normalized offered traffic loads, defined as the ratio of packets generated by all servers to the total capacity of PSON. The optical-link length between ToR and PSON architecture is set to 60 m. As frames traveling through the switch have to cover this distance at least twice [see Fig. 1(a)], propagation delay will be 600 ns. Longer or shorter links do not impact the PSON performance, but they can be seen as a larger or smaller latency offset to the latency introduced

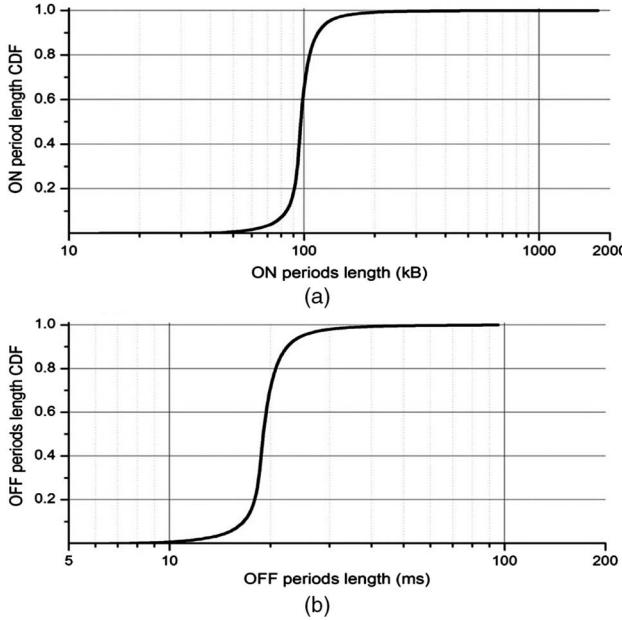


Fig. 5. Transmission periods of a simulated server. (a) CDF of ON transmission periods. (b) CDF of OFF transmission periods.

by PSON. We iterate 50 independent instances so that all plotted values have a 95% confidence level. Simulation experiments are conducted on an 8-core x86 – 64 bit Intel(R) Core(TM) i7-4770 CPU at 3.40 GHz. C4.5 has a time complexity of $O(mn^2)$, where m is the size of the training data, and n is the number of attributes. The time complexity is related with the height of the tree. So, some post-processing, like the pruning tree, can decrease the height of the tree, making the decision tree more balanced and reducing the runtime. For NBD, determining the most likely class for an instance consists of calculating the product of $n + 1$ factors K times, where n is the number of features and K is the number of classes. So, the runtime complexity of NBD classification is $O(nK)$. This is very computationally efficient and gives naive Bayes high scalability since the runtime scales linearly in the number of features n and number of classes K .

We use the above traffic-generation method to generate a data set of prelabeled mice and elephant flows. Note that elephant flow and mice flow are biased distributed in network traffic, i.e., elephant flow occupies only 10% and mice flow occupies 90% of total flow. To deal with such biased data sets, we do over-sampling for under-represented elephant flows by adding samples of instances, while doing under-sampling for over-represented mice flows by deleting samples of instances. Also, we do rescaling to standardize the range of independent features of data in [0,1]. We apply a “cross-validation” technique [27] to train and test our proposed ML-based flow detection methods (C4.5 decision tree and NBD) on the data set. Specifically, the data set is first split into 10 approximately equal partitions, then the first partition is used for testing, while the remaining 9 are used for training. The procedure is repeated 10 times, so that every instance is used exactly once for testing. The overall recall is calculated from the average of the recalls measured over 10 tests. We set the PA-threshold as 100 MB for the modified end-host-based detection method [13].

Figure 6 demonstrates the recall for applying NBD and the C4.5 decision tree on elephant-flow detection with increasing sliding window size. With a small window size

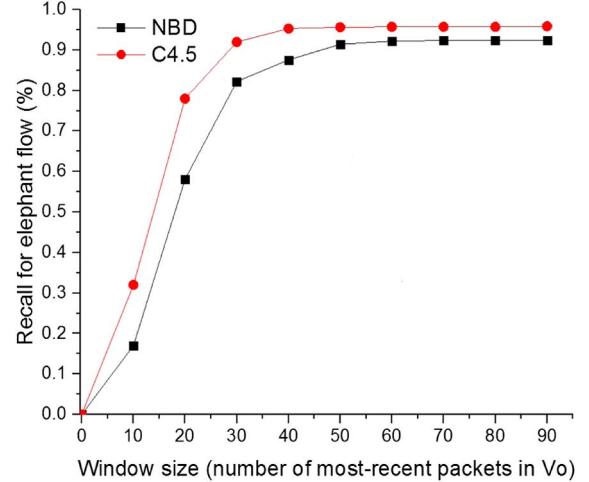


Fig. 6. Recall for NBD and C4.5 with different window size.

(less than 30), both C4.5 and NBD cannot achieve high recall, i.e., both methods are not able to detect elephant flows accurately. But the C4.5 decision tree can obtain recall value higher than 95% with window size larger than 30, while NBD can get recall value higher than 90% with window size larger than 50.

Figure 7 shows the normalized classification speed for the C4.5 decision tree and NBD detection methods when tested with different window sizes. A value of 1 represents the fastest classification speed (60,500 classifications per second with 10 window size). We note that larger window size leads to larger classification time but improves recall. Thus, in practice, a trade-off must be made between accuracy and processing speed when choosing the flow-detection method. Note that we do not compare the modified end-host-based detection method in Figs. 6 and 7, because these metrics are only for ML-based flow-detection methods.

In Fig. 8, we compare the average delay of PA and RR algorithms. We apply the Mahout threshold, C4.5 decision tree, and NBD methods separately to detect elephant flows periodically for the PA algorithm. A packet's latency has four parts: ingress buffering time in VoQs, tuning time of space switch, tuning time of tunable-wavelength light source, and the optical links. At low loads (i.e., smaller than 0.3), PA and RR incur similar delays because high-priority VoQs are rarely occupied, and most VoQs have similar priority values. But as load increases, PA outperforms RR, and delay of PA increases at a much slower rate than RR. This happens because, when load is large, temporal and spatial traffic dynamics are drastically different among servers (due to bursty traffic), which leads to various priority values for different VoQs in different time slots. The PA algorithm promotes the transmission of packets in VoQ with highest priority, reducing the average delay, while in RR, packets of bursty traffic are buffered in VoQs, and they have to wait until their turn of transmission. Note that PA with the C4.5 decision tree performs best for all loads.

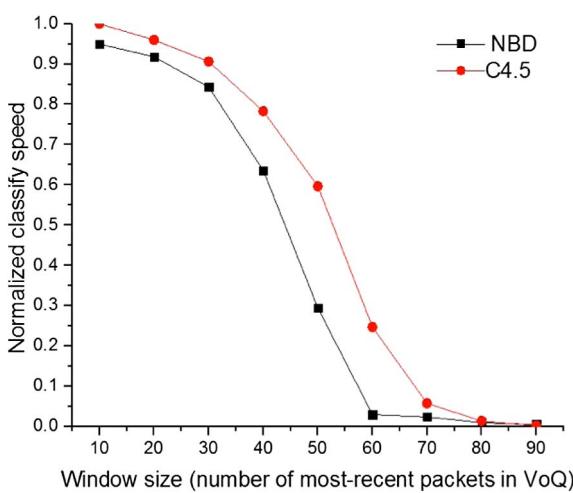


Fig. 7. Normalized classification speed for NBD and C4.5 with different window size.

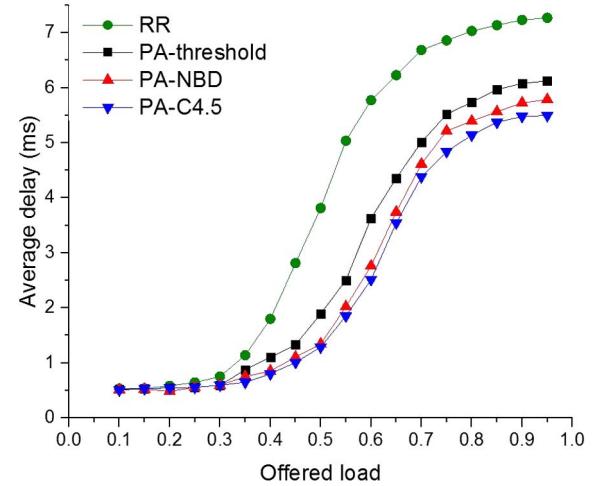


Fig. 8. Average delay of PA and RR algorithms.

This is because C4.5 provides higher accuracy for elephant-flow detection than the other methods. PA with the Mahout threshold does not work well compared with C4.5 and NBD. We explain this using the following example. Imagine a sliding window containing a few large-size packets belonging to elephant flows at the beginning of the M samples, while the rest of the samples are mice-flow packets. Now, only a few elephant packets might make the average packet length bigger than the PA-threshold and be detected as elephant flows by mistake. Generally, it is hard to find a perfect threshold to handle all scenarios, so threshold-based approaches tend to perform worse than ML-based approaches.

Figure 9 shows PLR of the PA and RR algorithms as a function of offered load. PA and RR achieve similar PLR under low load (<0.3), confirming the results in Fig. 8, because contention rarely happens and almost all of the newly generated packets can be transmitted immediately, so latency only depends on tuning time and propagation time. But, as load increases (>0.3), PA achieves much less

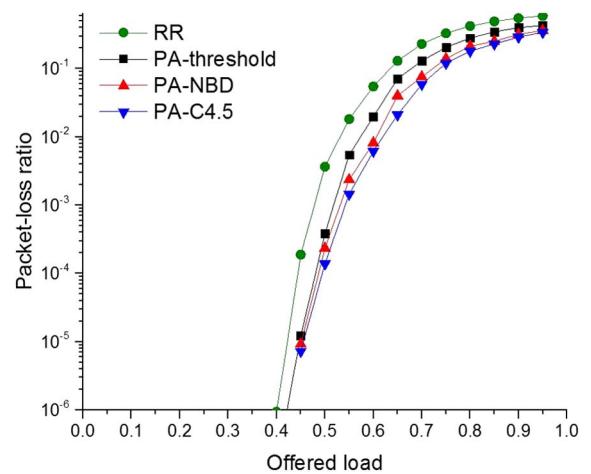


Fig. 9. PLR of PA and RR algorithms.

PLR than RR. This is because, in each period, an ingress module can only choose one VoQ to transmit its packets so that which VoQ being chosen is very important to decide the performance of the total PSON. PA always selects the VoQ with highest priority (either packets with larger sizes or a larger number of packets), while RR may choose a VoQ with small priority due to its randomness, which leads to longer latency. Hence, PA can achieve smaller PLR, improves bandwidth utilization, and enhances the PSON capacity. PLRs of both PA and RR tend to saturate, because at high loads, the buffer is not large enough to store all packets, and only some partially generated packets can be transmitted. Also note that the C4.5 detection method can achieve the smallest PLR, confirming that an accurate elephant-flow detection method is very helpful to improve the performance of PSON datacenter.

Figure 10 demonstrates how weight factors w_l and w_p impact the PA algorithm in terms of average delay. In this figure, PA algorithm follows the updating process described in Section III.B to change w_l and w_p values, and the plot marked as w_l represents a special version of the PA algorithm that does not have a weight-updating process and assigns weight factors set to zero except w_l (similarly for the plot marked as w_p). As we can see, PA with a weight-updating process can achieve a smaller average delay than the other two cases. This is because different servers may generate and exchange different flow types, as stated in Section III.A. For example, if we decide the VoQ from which we transmit to be based on packet length only, then some VoQs containing many small-size packets will have little opportunity to be selected and result in longer delay. Thus, a PA algorithm with a weight-updating process that considers both factors (w_s, w_d) can achieve the best performance.

In Fig. 11, we study the influence of weight factors w_l and w_p in PSON in terms of PLR. When load is small, contention rarely happens, so there is almost no packet loss. With increasing load (>0.4), the PLR of the PA algorithm grows more slowly than the other two, which confirms the results in Fig. 9. We also see that the plot of w_l , which only

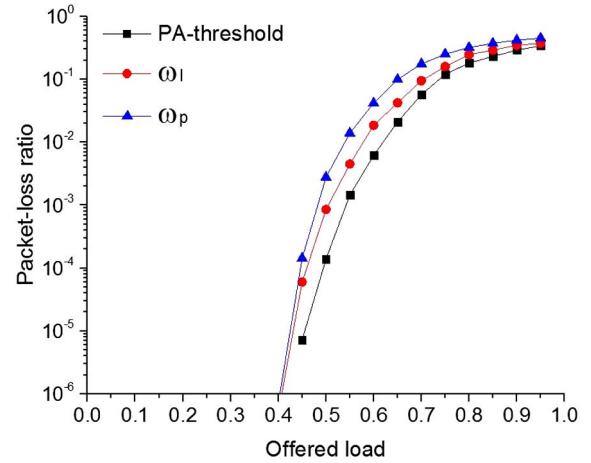


Fig. 11. PLR of PA with different weight factor.

considers the total packets' length in each VoQ, gives lower PLR than the plot of w_p , which only considers the number of packets in each VoQ. Assuming that w_p decides the priority and w_l is set as zero, then a VoQ containing a small number of packets will have a low priority and less chance to be transmitted. But this VoQ may buffer elephant flows, and this leads to dropping of new incoming packets due to limited buffer size. This also confirms the result in Fig. 10 because only considering w_p will result in more packets being dropped, and it can reduce average delay on the other hand.

Figure 12 shows the impact of different buffer sizes on average delay in PSON. At low load (<0.3), buffer size does not have an effect on latency. Since there is low contention, packets can be transmitted immediately without queueing in the buffer. As load increases, we see that small buffer size, such as 8 KB, gives the lowest average delay. But it also leads to high PLR (see Fig. 13), so it is not practical. More buffer capacity leads to larger average delay, because of more packets are queueing in the buffer and waiting to be transmitted.

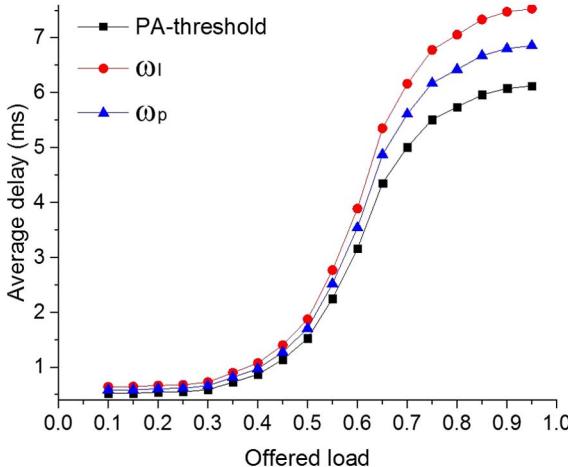


Fig. 10. Average delay of PA with different weight factor.

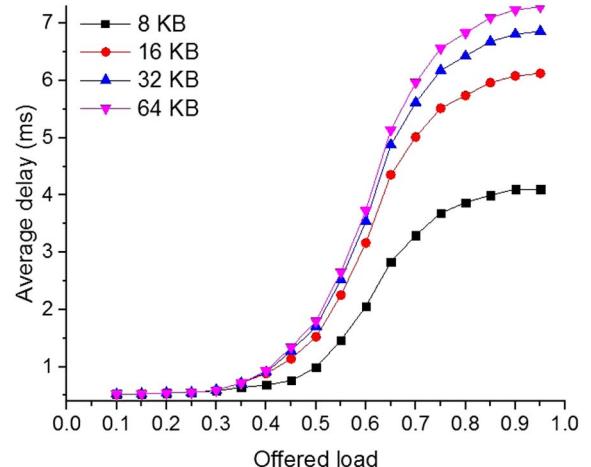


Fig. 12. Average delay of PA with different buffer size.

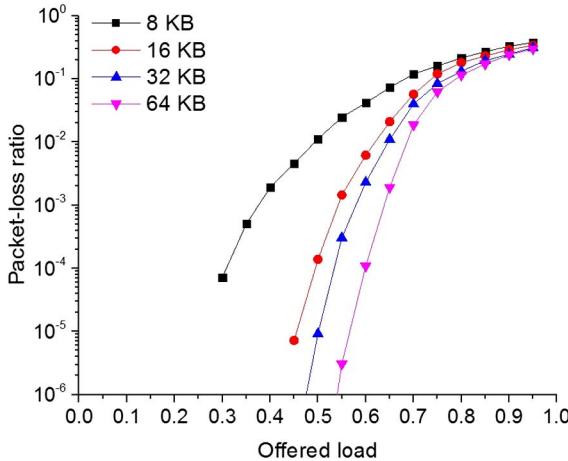


Fig. 13. PLR of PSON with different buffer size.

Figure 13 shows the effect of different buffer sizes in terms of PLR in PSON. We can see how, as buffer size increases, the system can manage heavier loads without packet loss. Smaller buffers, such as 8 KB, are insufficient for data center traffic. Accordingly, buffer of 16 KB can be enough to achieve a PLR below 10^{-6} for data center traffic loads smaller than 0.4 and achieve sub-microsecond latency (900 ns). Increasing buffer size beyond 16 KB provides better performance in terms of packet loss. As expected, larger buffers can store incoming bursty traffic, and hence they drop fewer packets. But larger buffers also lead to increasing average delay and are much more expensive. In practice, an optimal size should be chosen depending on the application requirements, latency limitation, and other factors.

VI. CONCLUSION

In this study, we present PSON with a centralized controller. We apply three classification methods to detect elephant flows, including Mahout, C4.5, and NBD. According to the PSON architecture and data center traffic-flow characteristics, we propose a PA scheduling scheme by enhancing the traditional RR algorithm. We study three classification methods in terms of recall and classification speed. C4.5 performs the best under emulated traffic in a data center network. We numerically study the performance of the PA algorithm in terms of packet-loss ratio and average delay, compared with the RR algorithm. We also explored the impact of various factors on performance of PA algorithm in PSON architecture. Finally, we studied how buffer size affects the system performance and illustrated the trade-off between average delay and packet-loss ratio.

ACKNOWLEDGMENT

This work was supported by an Institute for Information & Communications Technology Promotion (IITP) grant funded by the Korean government (MSIT) (No. 2016-0-00573, Development of Data Center Optical Networking

Core Technologies for Photonic Frame Based Packet Switching).

REFERENCES

- [1] “Cisco Global Cloud Index: Forecast and Methodology, 2016–2021,” Cisco White Paper, 2016 [Online]. Available: <https://www.cisco.com/c/en/us/solutions/collateral/service-provider/global-cloud-index-gci/white-paper-c11-738085.html>.
- [2] “Cisco Nexus 7000 series hardware installation and reference guide,” 2012 [Online]. Available: http://www.cisco.com/c/en/us/td/docs/switches/datacenter/hw/nexus7000/installation/guide/n7k_hig_book.html.
- [3] Y. Zhao, J. Zhang, M. Zhang, Y. Ji, and W. Gu, “DREAM: dual routing engine architecture in multilayer and multidomain optical networks,” *IEEE Commun. Mag.*, vol. 51, no. 5, pp. 118–127, May 2013.
- [4] H. Yang, J. Zhang, Y. Zhao, J. Han, Y. Lin, and Y. Lee, “SUDO: software defined networking for ubiquitous data center optical interconnection,” *IEEE Commun. Mag.*, vol. 54, no. 2, pp. 86–95, Feb. 2016.
- [5] J. Zhang, Y. Ji, M. Song, H. Li, R. Gu, Y. Zhao, and J. Zhang, “Dynamic virtual network embedding over multilayer optical networks,” *J. Opt. Commun. Netw.*, vol. 7, no. 9, pp. 918–927, Sept. 2015.
- [6] C. Kachris and I. Tomkos, “A survey on optical interconnects for data centers,” *Commun. Surv. Tutorials*, vol. 14, no. 4, pp. 1021–1036, 2012.
- [7] S. Di Lucente, N. Calabretta, and H. J. S. Dorren, “Low-latency photonic packet switches with large number of ports,” in *16th European Conf. Networks and Optical Communications (NOC)*, 2011, pp. 5–7.
- [8] H. Takahasi, “Transmission characteristics of arrayed waveguide $N \times N$ wavelength multiplexer,” *J. Lightwave Technol.*, vol. 13, no. 3, pp. 447–455, Mar. 1995.
- [9] K. Xi, Y. H. Kao, and H. J. Chao, “Petabit Optical Switch for Data Center Networks,” Tech. Rep., Polytechnic Institute of New York University, 2010.
- [10] O. Liboiron-Ladouceur, I. Cerutti, P. G. Raponi, N. Andriolli, and P. Castoldi, “Energy-efficient design of a scalable optical multiplane interconnection architecture,” *IEEE J. Sel. Top. Quantum Electron.*, vol. 17, no. 2, pp. 377–383, 2010.
- [11] L. Wang, X. Wang, M. Tornatore, K. Joon Kim, S. M. Kim, D. U. Kim, K. E. Han, and B. Mukherjee, “Priority-aware scheduling for packet-switched optical networks in datacenter,” in *Optical Network Design and Modeling (ONDM) Conf.*, 2017.
- [12] H. Mehrvar, Y. Wang, X. Yang, M. Kiaei, H. Ma, J. Cao, D. Geng, and D. Goodwill, “Scalable photonic packet switch test-bed for datacenters,” in *Optical Fiber Communication Conf.*, Los Angeles, California, 2016.
- [13] R. V. Rasmussen and M. A. Trick, “Round robin scheduling—a survey,” *Eur. J. Operation Res.*, vol. 188, no. 3, pp. 617–636, 2008.
- [14] E. L. Hahne, “Round-robin scheduling for max-min fairness in data networks,” *IEEE J. Sel. Areas Commun.*, vol. 9, no. 7, pp. 1024–1039, 1991.
- [15] T. Benson, A. Anand, A. Akella, and M. Zhang, “Understanding data center traffic characteristics,” *ACM SIGCOMM Comp. Commun. Rev.*, vol. 40, no. 1, pp. 92–99, 2010.
- [16] S. Kandula, S. Sengupta, A. Greenberg, P. Patel, and R. Chaiken, “The nature of datacenter traffic: measurements & analysis,” in *9th ACM SIGCOMM Internet Measurement Conf. (IMC)*, 2009, pp. 202–208.

- [17] 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 *9th ACM SIGCOMM Internet Measurement Conf. (IMC)*, 2009, pp. 95–104.
- [18] Caida (The Cooperative Association for Internet Data Analysis), "Packet size distribution comparison between Internet links in 1998 and 2008," [Online]. Available: <http://www.caida.org/research/traffic-analysis/>.
- [19] C. Hopps, "Analysis of an equal-cost multi-path algorithm," RFC 2992 (Informational), Nov. 2000.
- [20] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat, "Hedera: dynamic flow scheduling for data center networks," in *Conf. Networked Systems Design and Implementation (NSDI)*, 2010, vol. 10, p. 19.
- [21] A. R. Curtis, W. Kim, and P. Yalagandula, "Mahout: low-overhead datacenter traffic management using end-host-based elephant detection," in *IEEE INFOCOM*, Apr. 2011, pp. 1629–1637.
- [22] N. Williams, S. Zander, and G. Armitage, "A preliminary performance comparison of five machine learning algorithms for practical IP traffic flow classification," *ACM SIGGCOM Comp. Commun. Rev.*, vol. 36, pp. 5–16, 2006.
- [23] T. T. Nguyen and G. Armitage, "A survey of techniques for internet traffic classification using machine learning," *Commun. Surv. Tutorials*, vol. 10, pp. 56–76, 2008.
- [24] R. Kohavi and J. R. Quinlan, "Data mining tasks and methods: classification: decision-tree discovery," in *Handbook of Data Mining and Knowledge Discovery*, Oxford University, 2002, pp. 267–276.
- [25] G. H. John and P. Langley, "Estimating continuous distributions in Bayesian classifiers," in *11th Conf. Uncertainty in Artificial Intelligence*, 1995, pp. 338–345.
- [26] T. T. Nguyen and G. Armitage, "Training on multiple subflows to optimise the use of machine learning classifiers in real-world IP networks," in *31st Conf. Local Computer Networks*, Tampa, Florida, Nov. 2006.
- [27] I. H. Witten, E. Frank, L. E. Trigg, M. A. Hall, G. Holmes, and S. J. Cunningham, *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations*, 2nd ed., Morgan Kaufmann, 2005.