Blind, Adaptive and Robust Flow Segmentation in Datacenters

Francesco De Pellegrini[◇], Lorenzo Maggi[⋆], Antonio Massaro[◇],
Damien Saucez[†], Jérémie Leguay[⋆], and Eitan Altman[†].

***Abstract*—To optimize routing of flows in datacenters, SDN controllers receive a packet-in message whenever a new flow appears in the network. Unfortunately, flow arrival rates can peak to millions per second, impairing the ability of controllers to treat them on time. Flow scheduling copes with such sheer numbers by segmenting the traffic between elephant and mice flows and by treating elephant flows in priority, as they disrupt short lived TCP flows and create bottlenecks.**

We propose a learning algorithm called `SOFIA` and able to perform optimal online flow segmentation. Our solution, based on stochastic approximation techniques, is implemented at the switch level and updated by the controller, with minimal signaling over the control channel. `SOFIA` is blind, i.e., it is oblivious to the flow size distribution. It is also adaptive, since it can track traffic variations over time. We prove its convergence properties and its message complexity. Moreover, we specialize our solution to be robust to traffic classification errors. Extensive numerical experiments characterize the performance of our approach *in vitro*. Finally, results of the implementation in a real OpenFlow controller demonstrate the viability of `SOFIA` as a solution in production environments.

***Index Terms*—software defined networks, flow segmentation, stochastic approximation, adaptive algorithms, traffic classifiers**

I. Introduction

In the last decade, SDN controllers have become a de-facto production tool for routing traffic in datacenters [1], [2]. A data center fabric typically counts tens to thousands of switching units and hundreds of thousands of servers, and the rate of proactive control events – *packet-in* messages issued in OpenFlow at each flow arrival – can peak to several millions per second [3]. This causes a significant flow setup latency that can amount to up to 10% of the average flow duration [3], [4].

To this respect, customary Fat-Tree topologies [5] enable the heavy use of equal-cost multi-path (ECMP) routing [6], with flow-based hashing, as the default routing procedure. The main benefit of ECMP is that flows are immediately routed and no control message is issued toward the controller.

However, ECMP is suitable when there are several small (or *mice*) flows but no large (or *elephant*) flows [7], [8]. Indeed, the presence of elephant flows impairs the performance of ECMP for two reasons. First, ECMP does not differentiate between latency-sensitive mice flows in interactive applications and bulky transfers in data-intensive computing frameworks, e.g., Map-Reduce [9] or Spark [10]. Hence, mice flows may be queued behind elephants, which is to be avoided. Second, ECMP cannot utilize the available bandwidth effectively, as hash collisions between elephant flows create bottlenecks. This is a crucial issue in datacenters, where even though less than 10% of all flows classify as elephant flows, they carry more than 80% of the entire traffic [11].

A popular approach to overcome balancing issues with ECMP is called *flow scheduling*. Once elephant flows are detected, custom non-conflicting paths are allocated to them, thus avoiding long term collisions [7]. Hedera [7] has shown that managing elephant flows separately can yield as much as 113% higher aggregate throughput compared to ECMP. Server-side methods to detect elephant flows and hash them away from ECMP routing have been proposed and are already in production [8], [12]. In this case, the number of packet-in messages is drastically reduced, thus saving control channel capacity and latency budget for custom paths selection.

In all approaches where ECMP is used in conjunction with flow scheduling, a static threshold on the size of flows is used to discriminate elephants from mice flows. This form of *flow segmentation control* is our main focus. In fact, this segmentation threshold is typically difficult to determine: ideally, all active flows should be routed on custom optimized paths. Yet, the rate at which the controller can dispatch packet-in events is limited. Moreover, performing custom route installation for each and every flow consumes forwarding rules in switches: since they use power hungry and expensive TCAM memories, they are typically limited to a few thousands entries [13]. These limitations foster the need of continuously optimizing the threshold to follow evolving traffic conditions.

In this work we propose an adaptive and lightweight technique to combine flow scheduling decisions and flow segmentation control. First, we formulate an optimization problem on the size of flows whose route is optimized by the controller, also called "admitted" flows. We thus devise an optimal flow segmentation control policy, which is of threshold-type in the size of flows. The resulting scheme is semi-decentralized: the segmentation policy is implemented on board of each switch, thus reducing drastically packet-in events, but the policy itself is computed by the controller. The controller measures periodically the aggregated portion of optimized flows, with the aid of all switches. Then it updates the control policy and assigns at runtime to all switches the same flow segmentation threshold. The proposed algorithm, called `SOFIA`, is rooted in stochastic approximation. This allows to exploit the inherent threshold structure of the optimal segmentation policy and, remarkably, does not require to explicitly estimate the flow size distribution. Our algorithm works in the dark, i.e., irrespective of flow

[◇]Fondazione Bruno Kessler, Trento (Italy), [⋆]Huawei Technologies, France Research Center, [†]INRIA, Université Cote d'Azur, Sophia Antipolis (France).

978-1-5386-4128-6/18/$31.00 ©2018 IEEE

size distribution, and it can adapt when flow size distributions change over time. Finally, in production networks, the size of incoming flows may be unknown at packet-in generation time. To this respect, flow identification is typically performed via classification algorithms [14], [15], [16], [17], [18], [19], [20]. However, classification errors may severely degrade the performance of online flow scheduling, thus driving the system to inefficient operating points. A simple adaptation based on a delayed learning technique makes our scheme robust against classification errors.

Main contributions. *i) Flow segmentation under signaling constraint*: we formulate a flow segmentation problem that differentiates elephant and mice flows; the aim is to schedule a maximum amount of traffic under a constraint on the maximum rate of packet-in events;

ii) Online and asynchronous learning of the optimal segmentation policy: we design `SOFIA` algorithm which is *blind*, i.e., it ignores the flow size distribution, and *adaptive*, because it adjusts automatically to variations in the flow size distribution; also, it does *not* require any *synchronization* among switches;

iii) Flow size misclassification: we account for the case of flow-size classification errors; `SOFIA` is adapted using simple delayed learning of expected flow sizes.

To the best of the authors' knowledge, this work is the first one to provide a learning mechanism for flow segmentation able to work in the dark irrespective of flow size distribution and robust to flow classification errors.

Paper structure. In Sec. II we review the literature on flow scheduling for SDN and we outline the main contributions of this work. The system model and the flow segmentation control problem are described in Sec. III. The stochastic approximation algorithm is designed in Sec. IV. Robustness to classification errors is discussed in Sec. V. Numerical and network experiment results are presented in Sec. VI.

II. Related works and contributions

In the SDN literature, dynamic flow allocation to overcome hot spots is a core topic, ranging from multi-commodity flow (MCF) problems [21] to switch assignment schemes [22], where switches are assigned dynamically to multiple controllers. This paper addresses adaptive flow segmentation, in the case of a single controller. Also, it naturally extends to the case of multiple controllers and can work on top of existing MCF solutions. Scalability of centralized controller architectures in data centers and related issues has been debated in literature [4], [3]. Our scheme is meant to enforce existing control channel constraints to match traffic patterns dynamically. Several works have addressed coexistence of elephant flows and mice flows in SDN-enabled datacenter networks. In [7] the standard path reservation technique – scheduling the relatively low number of elephant flows over high throughput paths – has been proposed. Our scheme adopts a similar strategy in flow segmentation, since the controller utility prioritizes flows with larger size. Packet splitting techniques for elephant flows [23], on the other hand, are subject to packet out-of-order delivery, and thus require customized solutions for data-plane packet reordering. In [24] path differentiation is obtained by partitioning high-throughput links and low-latency links; it aims at efficient trade-off between delay constraints of short-lived mice flows and throughput requirements of elephant flows. [25] proposes heuristics to optimize the allocation of paths with respect to switch memory occupation. Path differentiation techniques are out of the scope of this work. However, our flow segmentation scheme is compatible with all the aforementioned techniques. The authors of [14] proved that for frameworks such as Hadoop and Spark accurate predictions of source, destination, and flow size are possible. Also, such information can help to reduce job completion time. Similarly, the work in [26] studies how to predict the flow size of incoming flows; elephant flows are then sent on the least congested path in order to minimize the completion time. The authors of [8] propose to monitor end-host socket buffers in order to improve the detection of elephant flows. Flow classification has been performed in connection with the notion of co-flows [15], [16], namely, sets of flows representing traffic patterns of certain tasks, e.g., of Map-Reduce instances. The work in [15] performs co-flow scheduling without full prior knowledge, demonstrating remarkable performance gains. In [16], machine learning is added on top of a flow scheduling system. In our work, we do not make any specific assumption on the classifiers employed. Rather, our objective is to render our semi-decentralized flow segmentation scheme robust to classification errors.

Table I
Main notation used throughout the paper

Symbol	*Meaning*
$\mathscr{R} = \{r_j\}_{j=1}^N$	set of flow sizes, with $r_1 > r_2 > \cdots > r_N$
$\mathscr{S}$	set of switches ruled by the controller, $S = \lvert\mathscr{S}\rvert$
t^k	time instant of k-th flow arrival
λ_j	rate of arrival of flows of size r_j
λ	total flow arrival rate $\sum_j \lambda_j$
$c\lambda$ ($c \in [0;1]$)	maximum rate (portion) of flows served by the controller
$u = \{u_j\}_{j=1}^N$	threshold-type flow segmentation policy
α	flow segmentation threshold, $\alpha = \sum_j u_j$
θ	expected fraction of admitted flows, $\theta = \sum_j p_j u_j$
P_{ij}	probability that flow size r_i is classified as size r_j
T	`SOFIA` observation window size (round duration)
ϵ^n	`SOFIA` stepsize at round n
τ_i^n	`SOFIA` random backoff time of switch i
Δ^n	`SOFIA` relative error at round n
η	`SOFIA` tolerance on the relative error

III. System Model and Problem Formulation

We consider a datacenter network with assigned topology, a set $\mathscr{S}$ of leaf (origin and/or destination) switches and one controller that switches are associated to. Flows originate from racks attached to their respective origin switch.

We assume that network flows are classified with respect to their volume, which we call flow size. $\mathscr{R} = \{r_j\}_{j=1}^N$ is the set of possible flow sizes, sorted in decreasing order ($r_1 > r_2 > \cdots > r_N$). We suppose that flows with size r_j appear in the system according to a Poisson stochastic process with

intensity λ_j flows per second, and we call $\lambda = \sum_{j\in\mathscr{R}} \lambda_j$ the overall intensity of flow arrivals.

When a flow not already installed appears in the system, a flow classifier associates a flow size to the tagged flow[1]. We will first assume that the classifier is ideal and never misclassifies flows. We shall relax this assumption in Section V where we will account for classification errors.

Flows may have destination in a different rack, attached to a destination leaf switch. In such a case, they can be served either via an *optimized* route or via a *default* route. Default routes correspond to pre-installed wildcard entries in the switch flow table, enabling hash-based load balancing when multiple parallel routes exist. Installing an incoming flow on a default route does not require signaling, as no new rule has to be installed and no packet-in signal is sent to the SDN controller. However, routing all flows on default routes is not desirable for classic QoS (or routing cost) considerations, as it causes collisions among elephant flows and disruption of mice flows by elephants in customary ECMP installations [8], [12].

Instead, optimized routes are computed on-the-fly by the SDN controller with a QoS objective, e.g., to reduce congestion by avoiding collisions with the rest of the traffic.

However, the frequency at which the switches can interrogate the controller – by sending a packet-in for the new incoming flow – is limited by three factors. First, packet-in messages generate traffic on the control channel between switches and the controller. Second, the request for a new path calculation creates a computational burden for the controller. Third, routes have to be installed on all switches along the optimized route. These factors introduce additional delay in the installation of each new flow, which is to be avoided. For this reason, we assume that the controller can handle at most $c\lambda$ packet-in requests per unit of time on average, where $c \in [0;1]$. For simplicity of analysis, in this paper we will assume that c is predefined and constant. In the practice though, c should depend on the overall flow arrival rate, as well as on the congestion and computation capabilities of the controller, which vary over time.

More formally, let us assume that at time t^k ($k \in \mathbb{N}$) an origin switch, that we call $i^k \in \mathscr{S}$, detects the arrival of a new flow with destination $d^k \in \mathscr{S}$ and size $r^k \in \mathscr{R}$. We denote by $u^k = \{0,1\}$ the action taken by switch i^k at time t^k. The switch can decide whether to interrogate the SDN controller for the computation of an optimized route ($u^k = 1$) or to install the new flow on the default, pre-computed route from i^k to d^k ($u^k = 0$). In the former case we say that the packet-in request for the incoming flow has been *admitted*, in the latter it has been *rejected*. The constraint on the packet-in signaling translates into the following expression:

$$\limsup_{K\to\infty} \frac{1}{K} \sum_{k=1}^{K} \mathbb{E}\left[u^k\right] \leq c. \tag{1}$$

We describe hereafter the main assumptions underlying our network model: 1) the SDN controller can be reached from every switch in the network via a control channel, 2) packets that do not match any custom forwarding rule are forwarded on pre-installed default paths via wildcard rules, and 3) memory constraints on the flow table in the switch are less stringent than the control channel constraints. Hence, flows can always be installed in the switches by the controller; we shall study the effect of memory constraints in future works.

We study the case where the switch decision to generate a packet-in for the controller depends on the size of the incoming flow as we want to reserve specific routes to the largest flows [7]. Our objective is to maximize the overall volume of traffic that runs over the optimized routes that are computed by the SDN controller at run-time, namely

$$\max_{u} \limsup_{K\to\infty} \frac{1}{K} \sum_{k=1}^{K} \mathbb{E}\left[u^k\, r^k\right] \tag{2}$$

under the signaling budget constraint in (1). This modeling choice is backed up by the well known fact that scheduling the relatively low number of elephant flows over high throughput paths sensibly improves flows completion times [7].

The problem (2) subject to (1) can be formulated as a constrained Markov Decision Process (MDP) where the state is simply represented by the size of the incoming flow. By MDP theory [27] we know that an optimal strategy can be found among stationary ones, that only depend on the current state. We then denote a *segmentation policy* u by the probability u_j that a switch interrogates the controller when a flow of size r_j is detected, i.e., $u_j = \mathbb{P}\left\{u^k = 1\right\}$ where k is such that $r^k = r_j$. Note that this implies that *the same strategy u is implemented by all switches*. It then follows that (2,1) can be reformulated as a standard continuous knapsack problem of the kind:

$$\max_{u\in[0,1]^N} \sum_{j\in\mathscr{R}} u_j p_j r_j \tag{3}$$

$$\text{s.t.} \sum_{j\in\mathscr{R}} u_j p_j \leq c \tag{4}$$

where $p_j = \lambda_j/\lambda$ is the probability that an incoming flow is of class j. The optimal segmentation strategy $u(\alpha^*)$ for (3,4) is provided by the classic threshold-type Dantzig solution [28]:

$$u_j(\alpha^*) = \begin{cases} 1 & j \leq \lfloor \alpha^* \rfloor \\ \alpha^* - \lfloor \alpha^* \rfloor & j = \lfloor \alpha^* \rfloor + 1 \\ 0 & j \geq \lfloor \alpha^* \rfloor + 2 \end{cases} \tag{5}$$

and the optimal segmentation threshold α^* solves the equation $\theta(.) = c$, where:

$$\theta(\alpha) := \sum_{j=1}^{\lfloor \alpha \rfloor} p_j + (\alpha - \lfloor \alpha \rfloor) \cdot p_{\lfloor \alpha \rfloor + 1}. \tag{6}$$

However, the SDN controller and the switches are *oblivious to the flow size distribution* $\{p_j\}_j$, hence they cannot compute the optimal strategy as in (5). Therefore, our paper proposes a learning algorithm that converges to the optimal threshold α^* and finally solves problem (2,1).

[1]E.g., it can be implemented by updating monitoring rules on a specific flow table [17], [18] or using dedicated flow sampling methods [19], [20]

Desiderata and possible approaches. Our aim is to design an algorithm solving problem (2,1) under the assumption that the SDN controller and the switches are oblivious to the flow size distribution $\{p_j\}_j$. A few techniques are available to tackle our problem. The simplest one prescribes to let each switch count the number of arrivals per flow class and transmit periodically the histogram to the controller, which can estimate the aggregate distribution $\widetilde{p}^n$. Then, the controller computes the corresponding approximate threshold $\widetilde{\alpha}^n$, that is sent back to the switches which apply over the next round the threshold policy $u(\widetilde{\alpha}^n)$. Although convergence to the optimal threshold α^* is guaranteed, the main drawback of this approach consists in generating high overhead traffic, being in the order of $\approx |\mathcal{S}||\mathscr{R}|$ flow counters per round. A second alternative is based on a classic Lyapunov technique, called drift-plus-penalty (DPP) [29]. Each switch simulates a virtual queue whose length is reduced by an amount c whenever a new flow appears and is increased by one unit only if the flow is accepted. In other words, if Q^k is the queue length at time t^k, $Q^{k+1} = \max(Q^k + u^k - c, 0)$. The acceptance rule is the classic DPP threshold policy: $u^k = 1$ whenever $r^k \geq Q^k/V$, where $V > 0$. Note that this policy does not suffer from the overhead issues above. However, one has to bear with the classic $O(1/V)$, $O(V)$ trade-off between optimality and constraint violation, respectively [29]. Moreover, if each switch i locally observes different flow size distributions p^i, then assigning the same constraint c to all switches is suboptimal, and each switch has to optimally tune a private constraint c_i, whose optimal value is $\sum_{j=1}^{N} u_j(\alpha^*)p_j^i$. Yet, the computation of c_i for each switch i boils down to the estimation of the local flow distribution, which pushes us back to the original problem. We will notice that this issue does not arise in our scheme, which plays directly with the segmentation threshold α.

Motivated by this, we propose a learning algorithm that generates low extra signaling traffic between switches and controller, that does not depend on locality of flow size distribution, but that is still able to converge to the optimal flow segmentation policy.

IV. Stochastic Approximation Solution

In this section we tackle the flow segmentation control problem (2) under signaling constraint (1) by solving the equation $\theta(\alpha) = c$ in an online fashion, where function $\theta(.)$ is defined as in (6). This approach requires only the iterative evaluation of the fraction of flows that have been admitted during a given observation interval.

It is important to observe that the function $\theta(\alpha)$ is unknown at runtime, because it depends on distribution $\{p_j\}_j$. Yet, we can easily come up with an unbiased estimator for $\theta(\alpha)$. In fact, this quantity is the sample average of the Bernoulli random variable $Y^k(\alpha) = \{1, 0\}$ which indicates whether the flow arriving at switch i^k at time t^k has been admitted, assuming that the policy $u(\alpha)$ is used at time t^k. Moreover, we observe that $\theta(\alpha)$ is a strictly increasing function of α. Our goal then becomes finding the root of a monotone increasing function of α, namely $(\theta(\alpha) - c)$, which we acquire through noisy observations $Y^k(\alpha)$, where the additive noise term has zero mean. For this class of problems, stochastic approximation theory provides the solution concept, which we employ with an algorithm of the Robinson-Monroe type [30].

The algorithm works in rounds with fixed time duration T, also called the observation window. During round n, i.e., during time interval $[nT, (n+1)T]$, all switches adopt a threshold policy $u(\alpha^n)$, where α^n has been broadcasted by the controller to all switches at time nT. To understand how α^{n+1} is updated at the next round, let $A_i^n(t)$ and $R_i^n(t)$ count the total number of new flows that have been admitted and rejected by switch i during the time interval $[nT, nT + t]$, respectively. More formally,

$$A_i^n(t) = \sum_{k:\, nT \leq t^k \leq nT+t} \mathbb{1}(Y^k(\alpha^n) = 1).\mathbb{1}(i^k = i) \quad (7)$$

$$R_i^n(t) = \sum_{k:\, nT \leq t^k \leq nT+t} \mathbb{1}(Y^k(\alpha^n) = 0).\mathbb{1}(i^k = i). \quad (8)$$

Each switch $i \in \mathcal{S}$ waits for a random backoff time τ_i^n and then reports to the controller the quantities $A_i^n(\tau_i^n)$ and $R_i^n(\tau_i^n)$. Note that this procedure does *not* require any sort of synchronization among switches. In order to simplify the implementation, it is convenient to assume that the random variables τ_i^n are *i.i.d.* over different switches $i \in \mathcal{S}$.

At the end of round n, i.e., at time $(n+1)T$, the controller aggregates the counters sent by switches before the deadline and then computes the total portion of accepted flows $\bar{Y}^n(\alpha^n)$:

$$\bar{Y}^n(\alpha^n) = \frac{\sum_{i\in\mathcal{S}} A_i^n(\tau_i^n).\mathbb{1}(\tau_i^n \leq T)}{\sum_{i\in\mathcal{S}} A_i^n.\mathbb{1}(\tau_i^n \leq T) + R_i^n.\mathbb{1}(\tau_i^n \leq T)}. \quad (9)$$

Under the *i.i.d.* assumption on waiting times τ_i^n, the quantity $\bar{Y}^n(\alpha^n)$ is an unbiased estimator for $\theta(\alpha^n)$, i.e.,

$$\mathbb{E}\left[\bar{Y}^n(\alpha^n)\right] = \theta(\alpha^n). \quad (10)$$

Then, the SDN controller updates the threshold α as follows:

$$\alpha^{n+1} = \Pi\left[\alpha^n + \epsilon^n\big(c - \bar{Y}^n\big)\right] \quad (11)$$

where $\Pi(.)$ is the projection $\max\{0, \min\{N, .\}\}$. The stepsize ϵ^n can be set to $\epsilon_0 \cdot n^{-\gamma}$, with $\epsilon_0 > 0$ and $1/2 < \gamma < 1$ (see Thm. 1). We call this procedure Stochastic Online Flow segmentatIon Algorithm (`SOFIA`), and one can find its compact description in Algorithm 1.

We remark that if the SDN controller can roughly estimate the flow distribution p, e.g., via historical data, then it is sensible to initialize the value of α_0 as the optimal value of the program (3,4) with respect to the estimated distribution.

We now prove that `SOFIA` converges to the optimal threshold α^*, hence solving our original problem (2) subject to (1).

Theorem 1. *Let the sequence $\{\epsilon^n\}$ be such that $\epsilon^n \geq 0\ \forall\, n$, $\sum_{n=0}^{+\infty} \epsilon^n = +\infty$ and $\sum_{n=0}^{+\infty} (\epsilon^n)^2 < +\infty$. Then, the policy $u(\alpha^n)$ converges to the optimal policy $u(\alpha^*)$ with probability 1.*

Dynamics, message complexity and convergence time. Hereafter we analyse the performance of `SOFIA`.

Algorithm 1: SOFIA

1: **input:** T, $\{\epsilon^n = \epsilon_0 n^{-\gamma}\}_n$, and $\gamma \in (1/2; 1]$
2: **initialize:** $\alpha_0 \in [0; N]$, e.g., by solving (3,4) w.r.t an estimated distribution p
3: **for** rounds $n = 0, 1, \dots$ **do**
4: At time nT the controller broadcasts new threshold α^n to all switches.
Each switch will adopt segmentation policy $u(\alpha^n)$ during time interval $[nT, (n+1)T)$
5: Each switch $i \in \mathcal{S}$ waits for a random time τ_i^n and then sends to the controller the quantities $A_i^n(\tau_i^n)$ and $R_i^n(\tau_i^n)$, being the total number of flows accepted and rejected within the interval $[nT; \tau_i^n]$, respectively, see (7),(8).
6: At time $(n+1)T$ the controller computes the portion of admitted flows $\bar{Y}^n(\alpha^n)$ during round n, as in Eq. (9)
7: Controller computes $\alpha^{n+1} \leftarrow \Pi\Big(\alpha^n + \epsilon^n(c - \bar{Y}^n)\Big)$
8: **end for**

1.a) Dynamics. We recall the convergence properties of the stochastic approximation algorithms of the Robinson-Monroe type [30]. The convergence argument proving Thm. 1 makes use of the Ordinary Differential Equation (ODE) method for stochastic approximations, based on the dynamics $\dot{\alpha} = c - \theta(\alpha)$. The output $\{\alpha^n(w)\}_{n\in\mathbb{N}}$ generated by the algorithm is a random process, where w belongs to its natural filtration. Actually, the deterministic ODE describes the temporal evolution of the mean value of the process, namely $\mathbb{E}[\alpha^n]$. The ODE converges to the unique restpoint which is asymptotically stable. The convergence result implies that the sample paths of α^n converge a.s. to the deterministic solution of the ODE. In practice, with probability one, the sample-paths $\{\alpha^n(w)\}$ follow the solution of the ODE closely for a time that increases to infinity as the number of steps increases (see also Fig. 2.c)). As a consequence, the estimates of SOFIA are confined in a small neighborhood of the optimal segmentation threshold α^* from which they can escape at most a finite number of times.

1.b) Convergence time. The following result provides a characterization of the algorithm convergence with respect to the estimate error Δ^n.

Theorem 2. *Let $\Delta^n := \frac{|\theta(\alpha^n) - c|}{c}$ be the relative error of SOFIA at the n-th round of the algorithm. Let $\epsilon^n = n^{-\gamma}$, where $1/2 < \gamma \leq 1$ and $\eta > 0$. Then, there exist $\beta, \nu > 0$ such that*

$$\mathbb{P}\{\Delta^n > \zeta\} \leq \frac{\beta}{\eta} \exp\Big(-\nu\, n^{1-\gamma}\Big).$$

Note that the bound presented in Thm. 2 is provided with respect to the number of rounds n, and does not depend on the window size T. Yet, intuitively, there should exist a finite value of T at which convergence speed is maximized; in fact, reducing T also degrades the estimate of $\bar{Y}^n$, which translates into a higher number of rounds to reach the same optimality gap. We show this numerically in Section VI, Fig. 2.b).

1.c) Message complexity. SOFIA is a decentralized algorithm and works by exchanging messages between switches and controller on a per round basis. As detailed in Alg. 1, switches transmit the quantities $A_i^n(\tau_i^n), R_i^n(\tau_i^n)$ to the controller after a tunable backoff time τ_i^n. Assume that each message may not be received successfully with probability $p_f > 0$. In order to account for possible retransmissions of control messages, we define message complexity of SOFIA as the number of control messages to be exchanged between the switches and the controller until a certain precision is attained. It is immediate to see that the message complexity is linear in the number of switches $|\mathcal{S}|$. In fact, at each round, the controller broadcasts the newly computed policy once to all switches. The switches reply sending $|\mathcal{S}|$ messages with the number of accepted and rejected flows during the previous round. The total number of messages to be exchanged per round is hence $(1+|\mathcal{S}|)/(1-p_f)$, plus the number of broadcast message retransmissions, which is $O(1+p_f/(1-p_f)\log(|\mathcal{S}|))$ [31]. By combining such considerations with Thm. 2 we obtain the next result.

Corollary 1. *Let η be the tolerance as in Thm. 2, and fix $0 < P_\eta < 1$. In order to attain $\mathbb{P}\{\Delta^n > \eta\} < P_\eta$, SOFIA generates $O\Big(\frac{|\mathcal{S}|}{(1-p_f)\nu} \log\big(\frac{\beta}{\zeta P_\eta}\big)\Big)$ message transmissions on the control channel.*

Adaptive solution. The decreasing stepsize formulation of SOFIA (e.g., $\epsilon^n = \epsilon_0 n^{-\gamma}$) does not react quickly to changes in the flow size distribution. However, it is possible to obtain a version of the algorithm which can react to changes in the traffic pattern. In fact, we can adopt the constant stepsize approach for stochastic approximation. It allows to converge faster, while partially sacrificing some of the noise-rejection properties of the original decreasing step size formulation. We need to introduce to this purpose a simple variant in the pseudocode of SOFIA, where the step of approximations has small but constant size, i.e., by assuming $\epsilon^n = \epsilon$ for any round n. In this case, line 7 of Alg. 1, i.e., the update step, writes

$$\alpha^{n+1} = \Pi\Big(\alpha^n + \epsilon(c - \bar{Y}^n)\Big). \tag{12}$$

Since the approximation stepsize does not change over time, the algorithm continues to adapt to changes in the flow size distribution. This seamless change in the formulation of the threshold update requires anyhow to prove the convergence properties ensured by the algorithm.

Theorem 3. *For any $\delta > 0$, define by $B_\delta(\alpha^*) = \{x \in \mathbb{R} : |x - \alpha^*| < \delta\}$. As $\epsilon \to 0$, the succession $\{\alpha^n\}_n$ computed as in (12) converges in distribution to elements in $B_\delta(\alpha^*)$. Moreover, the fraction of time spent by the process in $B_\delta(\alpha^*)$ during $[0, t]$ goes to 1 as t diverges.*

Thus, as time goes by, the fraction of time that sample paths generated in (12) spend in a (small) neighborhood of the ODE restpoint tends to one. The above result captures the trade off between the step size ϵ and adaptation capabilities of the algorithm. In fact, the larger the stepsize, the faster the algorithm convergence. However, the segmentation threshold α^n computed by the algorithm shall be confined in a larger neighborhood of the optimal threshold α^*.

V. Robustness to misclassification

So far we have assumed that the flow classifier is ideal. In other words, the switch is always able to successfully estimate the size class that the new incoming flow belongs to. In this final section we investigate the effects of flow misclassification, and provide a robust version of our stochastic approximation policy for flow segmentation.

Optimal policy under misclassification. We first describe the performance of a flow classifier through its confusion matrix P, defined for all $i, j \in \mathscr{R}$ as:

$$P_{ij} = \mathbb{P}\left\{\text{flow size is classified as } r_j \mid \text{its actual size is } r_i\right\}.$$

Similarly to the ideal case $P = I$ described in (3,4), the linear program that solves the optimal flow segmentation policy in the presence of classification errors writes:

$$W^*(P) = \max_{a\in[0;1]^N} \sum_{j=1}^{N} u_j \widehat{p}_j \widehat{r}_j \qquad (13)$$
$$\text{s.t.} \sum_{j=1}^{N} u_j \widehat{p}_j \leq c$$

where $u_j = 1$ whenever a flow that has been *classified* as having size r_j is accepted, $\widehat{p}_j$ is the probability that the incoming flow is classified as r_j and $\widehat{r}_j$ is the expected *actual* flow size when a flow has been detected as r_j, i.e.,

$$\widehat{p}_j = \sum_{i=1}^{N} p_i P_{ij}, \qquad \widehat{r}_j = \sum_{i=1}^{N} r_i \bar{P}_{ji}. \qquad (14)$$

Here $\bar{P}_{ji} = \frac{P_{ij} p_i}{\sum_{n=1}^{N} p_n P_{nj}}$ denotes, by Bayes' rule, the probability that the actual size of a flow is r_i, given that it has been classified as r_j. We observe that the optimal policy is still a threshold one, but the flow indexes are sorted according to the values of $\widehat{r}_j$, being in general different from r_j due to misclassification. More formally, let us define σ as the permutation of flows $1, \ldots, N$ such that $\widehat{r}_{\sigma(i)} > \widehat{r}_{\sigma(j)}$ whenever $\sigma(i) < \sigma(j)$. Then, the optimal policy for the problem in (13) is

$$u_i(\widehat{\alpha}, \sigma) = \begin{cases} 1 & \text{if } \sigma(i) \leq \lfloor \widehat{\alpha} \rfloor \\ \widehat{\alpha} - \lfloor \widehat{\alpha} \rfloor & \text{if } \sigma(i) = \lfloor \widehat{\alpha} \rfloor + 1 \\ 0 & \sigma(i) \geq \lfloor \widehat{\alpha} \rfloor + 2 \end{cases} \qquad (15)$$

where $\widehat{\alpha}$ solves the equation:

$$\widehat{\theta}(., \sigma) = c \qquad (16)$$

and $\widehat{\theta}(\alpha, \sigma) := \sum_{j=1}^{\lfloor \alpha \rfloor} \widehat{p}_{\sigma^{-1}(j)} + (\alpha - \lfloor \alpha \rfloor) \widehat{p}_{\sigma^{-1}(\lfloor \alpha \rfloor + 1)}$.

Clearly, the presence of classification errors decreases the achievable admitted traffic volume, namely

$$W^*(P) \leq W^*(I) := W^*.$$

To see this, it suffices to observe that the optimal strategy $u(\widehat{\alpha}, \sigma)$ for the problem (13) produces a feasible (but not necessarily optimal) solution $\bar{u}$ for the original problem (3,4) defined as $\bar{u}_i = \sum_j u_j(\widehat{\alpha}, \sigma) p_j \bar{P}_{ji}$.

The plain version of SOFIA applied to the scenario with classification errors solves an equation in the variable α which is possibly different from (16). In particular it writes:

$$\widehat{\theta}(., \sigma^I) = c \qquad (17)$$

where σ^I denotes the trivial permutation $\sigma^I(i) = i$ for all i and moreover $\widehat{\theta}(., \sigma^I)$ is defined as

$$\widehat{\theta}(\alpha, \sigma^I) = \sum_{j=1}^{\lfloor \alpha \rfloor} \widehat{p}_j + (\alpha - \lfloor \alpha \rfloor) \widehat{p}_{\lfloor \alpha \rfloor + 1}.$$

To this respect, we say that a classifier with confusion matrix P is *order-preserving* whenever $\sigma = \sigma^I$, i.e., the misclassification does not modify the order of flow sizes. We observe from the expression of $\widehat{r}_j$ in (14) that the order-preserving property not only depends on the properties of the classifier, but also on the flow size distribution p. The following fact immediately stems from the comparison of expressions (16) and (17).

Fact 1. *Under misclassification,* SOFIA *is able to solve* (13) *optimally if the classifier is order-preserving.*

In the very special case where only two flow size classes are considered ($N = 2$, e.g., elephant and mice flows) we can provide a positive result on the original version of SOFIA.

Lemma 1. *Let* $N = 2$ *and let* $P = \left(\begin{smallmatrix} 1-\varepsilon_1 & \varepsilon_1 \\ \varepsilon_2 & 1-\varepsilon_2 \end{smallmatrix}\right)$ *be the confusion matrix of the classifier. Then,* P *is order-preserving if and only if* $\varepsilon_1 + \varepsilon_2 \leq 1$.

As a consequence of Lemma 1 and Fact 1, when $N = 2$ SOFIA is able to attain the optimal value $W^*(P)$ for any flow size distribution if and only if $\varepsilon_1 + \varepsilon_2 \leq 1$. On the other hand, for $N \geq 3$ we have the following negative result.

Theorem 4. *For* $N \geq 3$*, there exists no classifier that is order-preserving for all flow size distributions.*

Next we overcome the negative result in Thm. 4 by proposing a robust version of SOFIA with delayed feedback on the actual size of flows.

Robust SOFIA with delayed feedback. We now propose a variant of SOFIA that converges to the optimal value $W^*(P)$ under misclassification errors. We exploit the fact that, although the decision on whether or not interrogate the controller has to be taken as soon as the classifier detects the arrival of a new flow, the SDN controller can still monitor the tagged flow later on. Thus, the controller can learn the value of the misclassified flow sizes $\widehat{r}$ via monitoring, and such information can be used to improve future decisions.

More precisely, we call $\widehat{r}_j^n$ the average actual size of flows that are classified as belonging to class j up to time nT. Similarly, σ^n is defined as the corresponding permutation of flow indexes, i.e., $\widehat{r}^n_{\sigma(i)^n} > \widehat{r}^n_{\sigma(j)^n}$ whenever $\sigma(i)^n < \sigma(j)^n$. Thus, the policy that is adopted during round n is $u(\alpha^n, \sigma^n)$, see (15). We name this variant Robust SOFIA (R-SOFIA) and we resume its steps in Algorithm 2. By the strong law of large numbers we can prove that R-SOFIA reaches asymptotically the optimal performance in terms of maximum average traffic volume $W^*(P)$ handled by the controller.

Algorithm 2: R−SOFIA

1: **input:** T, $\{\epsilon^n = \epsilon_0 n^{-\gamma}\}_n$, $\epsilon_0 > 0$, $\gamma \in (1/2; 1]$
2: **initialize:** $\alpha_0 \leftarrow 1$, $\sigma^1 \leftarrow \sigma^I$
3: **for** rounds $n = 0, 1, \ldots$ **do**
4: At time nT the controller broadcasts new threshold α^n and flow permutation σ^n to all switches.
Each switch will adopt segmentation policy $u(\alpha^n, \sigma^n)$ during time interval $[nT, (n+1)T)$
5: Each switch $i \in \mathcal{S}$ waits for a random time τ_i^n and then sends to the controller the quantities $A_i^n(\tau_i^n)$ and $R_i^n(\tau_i^n)$, see Eq. (7,8)
6: At time $(n+1)T$ the controller computes the portion of admitted flows $\bar{Y}^n(\alpha^n)$ during round n, see Eq. (9)
7: Controller computes $\alpha^{n+1} \leftarrow \Pi\Big(\alpha^n + \epsilon^n(c - \bar{Y}^n)\Big)$
8: Controller monitors flows in the network and produces the estimate $\widehat{r}_j^{n+1}$ as the average size of flows that are classified as belonging to class j up to time $(n+1)T$. It then computes the new class permutation σ^n that sorts $\widehat{r}^{n+1}$ in decreasing order
9: **end for**

Lemma 2. *The flow segmentation policy $u(\alpha^n, \sigma^n)$ of Algorithm 2 converges to the optimal policy $u(\widehat{\alpha}, \sigma)$ for problem* (13) *with probability 1.*

We remark that the price incurred in making SOFIA robust against misclassification errors is *i)* the overhead to communicate also the updated permutation σ^n in the step 1 of R−SOFIA, and *ii)* the flow size monitoring performed by the controller in step 8. However, such flow size monitoring can be performed at much slower timescale than R−SOFIA execution, and does not need to be applied to each and every flow, as long as the estimate $\widehat{r}_j$ is allowed to converge to the real value r_j for each class j.

VI. Numerical results

In this section we characterize the performance of SOFIA both from an algorithmic and a networking standpoint.

In-vitro experiments. We first describe numerical experiments on the performance of SOFIA. Fig. 1.a) reports two runs of the algorithm for the case $P = I$ (no classification errors). Flows arrivals follow a Poisson process with intensity $\lambda = 10^5$ flows/s. The probability distribution of flow size over $N = 4$ classes is $p = (1/6, 1/3, 1/12, 5/12)$. In the two runs of the algorithm the observation window size is set to $T = 1$ ms and $T = 10$ ms, respectively. Qualitatively, the time to converge appears several tenth milliseconds slower for $T = 10$ ms; SOFIA's output appears more noisy for $T = 1$ ms.

Fig. 1.b) quantifies the dependence of the convergence time on the window size T. The convergence time of the algorithm is measured by the largest time such that relative error $\Delta^n \geq \eta$. In our experiments, tolerance is set to $\eta = 0.05$; we report confidence intervals at 95% over 300 samples. We observe that if T is small, then increasing T allows to speed up the convergence: the larger T, the larger the number of samples and the better the estimates of the fraction of optimized flows $\theta(\alpha^n)$. Yet, setting T too big is detrimental for convergence time: in fact, the advantage of having good estimates of $\theta(\alpha^n)$ is canceled by the drawback of updating the threshold α^n too infrequently. As observed in Fig. 1.b), there exists a finite optimal value of T minimizing convergence time, the analysis of which we leave as part of future work.

Fig. 1.c) describes the trajectory of the ODE associated to SOFIA : $\dot{\alpha}(t) = c - \theta(\alpha(t))$. Such trajectory has been superimposed to the envelope of the sample paths generated by the algorithm for same initial condition $\alpha(0) = N/2$ (the time scale is the one of the stochastic approximation, namely $z_n = \sum_{k=1}^{n} \varepsilon_k$, see [30]). The solution of the ODE appears as the meanfield approximation of the sample paths, which are concentrated in a narrow neighborhood of the ODE dynamics.

Fig. 2.a) describes the adaptation of the algorithm with constant step size. A time-varying flow size distribution has been assumed for $N = 3$, where the flow size distribution switches between $p_1 = (2/3, 1/4, 1/12)$ and $p_2 = (1/4, 2/3, 1/12)$. As seen in the figure, the algorithm maintains the expected number of admitted flows (upper figure for $\theta(\alpha^n)$), while adjusting the optimal policy (lower figure for α^n); dashed lines denote the optimal solution.

Fig. 2.b) shows the effect of a non ideal classifier. We consider $P_{ii} = 1 - \varepsilon$ and $P_{ij} = \varepsilon$ for $i \neq j$ in the confusion matrix, for $0 \leq \varepsilon \leq 1$. For $\varepsilon = 0$, the optimal policy is a water-filling solution, where flows of size r_1 are admitted deterministically, flows of size r_2 undergo randomized admission and flows of size r_3 use default routes. In presence of classification errors ($\varepsilon = 0.15$) the threshold policy determined by SOFIA– optimal in this setting – is equivalent to a segmentation policy being randomized on all three flow sizes.

In Fig. 2.c) we compare the value of the maximum segmented traffic volume optimized by R−SOFIA in presence of classification errors against the value attained by SOFIA. The sample values are derived for 100 sample paths with 95% confidence intervals. As it can be observed, the optimal value decreases up to the critical value $\varepsilon = 2/3$. Beyond that value, the errors of the classifier are large, the order of flows solving optimally problem (13) becomes $\{2, 1, 3\}$, and the optimal policy attainable by R−SOFIA compensates for the errors by admitting larger number of flows classified as low size. Conversely, for $\varepsilon > 2/3$, SOFIA – which does not perform flow reordering – is suboptimal. The results suggests that flow rate reordering, as performed in R−SOFIA, is mandatory in the event of large classification errors.

Network experiments. After assessing the performance of SOFIA, we aim at understanding its behavior in a realistic environment. To that aim we have emulated the SDN-based cluster depicted in Fig. 2.c). The cluster is composed of four MapReduce servers running Hadoop [9] and a home-made traffic generator. The network is managed by one OpenFlow Ryu controller (https://osrg.github.io/ryu/) that implements SOFIA to dynamically configure two OpenvSwitch switches (http://openvswitch.org). We have loaded the Hadoop cluster with the `terasort` benchmark sorting a 1 GB file generated with *teragen*. This benchmark is available in the standard Hadoop MapReduce framework (http://hadoop.apache.org). In addition, we loaded the network with synthetic background

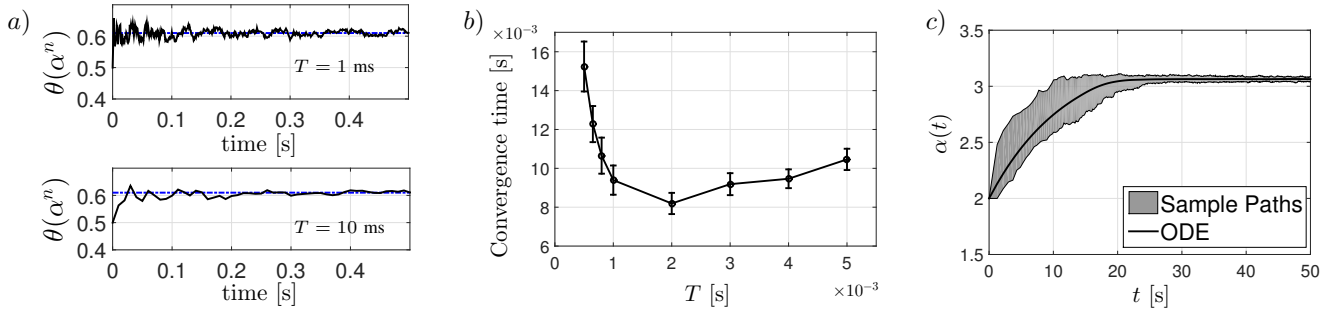

Figure 1. a) Sample paths of SOFIA for decreasing step size for $T = 1$ ms and $T = 10$ ms; $c = 0.61$ marked by the dashed blue line. b) Convergence time for $\eta = 0.05$ c) Sample paths versus ODE dynamics; $p = (1/6, 1/3, 1/12, 5/12)$, $c = 0.61$, $\lambda = 10^5$ flows/s.

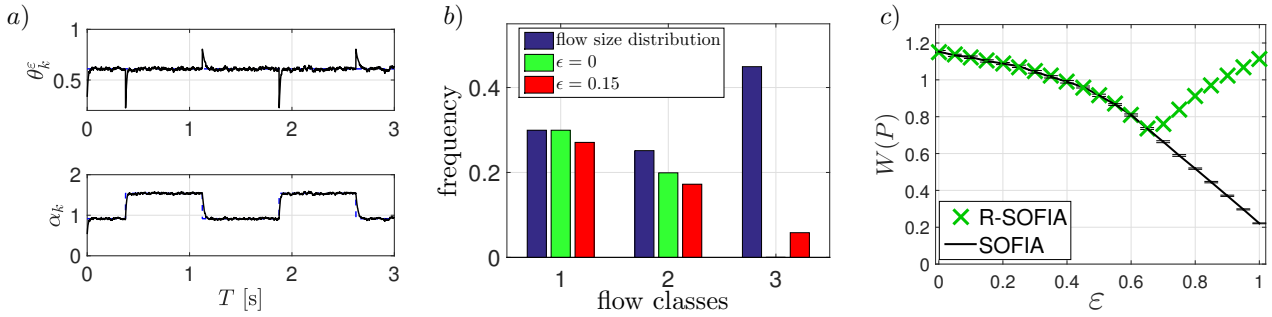

Figure 2. Sample paths of SOFIA for $N = 3$ and constant step $\varepsilon = N/10$, $c = 0.61$. b) Effect of classification errors confusion matrix P on the optimal policy; $c = 0.5$, and $p = (0.3, 0.25, 0.45)$ c) Performance loss for increasing values of classification error ε: comparison between SOFIA and R-SOFIA; $p = (0.01, 0.1, 0.89)$, $r_1 = 100$, $r_2 = 1$ and $r_3 = 0.1$ Mbyte, $c = 0.6$.

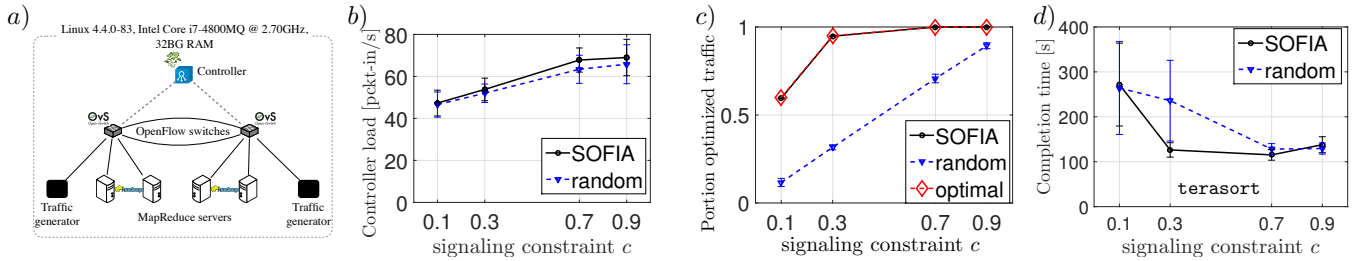

Figure 3. a) Cluster used for the network emulation; b) Maximum controller load; c) Average portion of optimized background volume; d) completion time under terasort vs. signaling constraint c. Confidence intervals at 95%.

traffic generated with our home-made TCP traffic generator. Background traffic flows are produced with an average rate of 20 new flows/s, according to a Poisson law and picked randomly from $N = 1000$ classes distributed according to a Zipf law of exponent 0.8 [32]. The size of every flow in a class i is $i^2 \cdot 1024$ bytes, in order to produce frequent mice and rare but large elephant flows.

As our objective is to optimize the control channel usage and not the data-plane, we have implemented a routing policy that segments background traffic by sending large background traffic flows to a non-conflicting path (similar to Hedera [7]). The remaining traffic, i.e., MapReduce and small background flows, is always routed on default ECMP paths without triggering control messages, i.e., packet-in messages; in fact, switches identify MapReduce via their TCP port numbers.

SOFIA is a sampling method that automatically adapts to keep a predefined signaling load c on the controller and still maximizes the amount of optimized traffic. That being said, we can first compare it with a naive random sampler that randomly filters signaling messages with a probability c of sending the packet-in messages to the controller. Fig. 3.b) reports the controller load, i.e., the fraction of packet-in messages received. The comparison with random confirms the correct computation of the threshold by SOFIA as it is essentially equivalent to random sampling, meaning that SOFIA conserves the good properties of usual flow sampling. Yet, Fig. 3.c) reveals the real advantages of SOFIA. It depicts the fraction of background traffic volume optimized by the controller and compares it to two radically different approaches: (i) the theoretical optimal with full a-priori

traffic knowledge and (ii) the random approach with no knowledge at all. In Fig. 3.c) we observe that the proportion of optimized traffic with SOFIA coincides with the expected optimal, which empirically confirms the convergence to the optimal solution, as proven in Thm. 1.

Fig. 3.d) shows the benefit of flow segmentation on the MapReduce traffic: since background elephant flows are rerouted by the controller to free bandwidth for MapReduce traffic resulting in lower completion times. The terasort benchmark traffic is composed by a small fraction of very heavy elephant flows, as it happens in the background traffic as well. Nevertheless, avoiding collision of rare, yet harmful, large elephants reduces drastically (up to 50%) the completion time. The gain of SOFIA over random is visible at intermediate values of c. In fact, the skewness of flow size distribution allows SOFIA to cope with all elephant flows even when the signaling constraint c is small. On the contrary, random cannot perform any better than basic ECMP load balancing.

VII. Conclusions

Ideally, the SDN controllers of a datacenter should compute an optimized route for every new connection request, aiming at network-wide objectives such as minimum routing cost or minimum link congestion. Yet, at the typical frequency of packet-in events occurring in such networks, SDN schemes incur excessive latency. This suggests to segment the traffic flows to be optimized: switches interrogate the controller for path computation only for flows which are "big enough", following the customary strategy by which elephant flows should be treated in priority. In this paper we presented an online learning algorithm called SOFIA, able to learn the optimal segmentation threshold without any *a priori* knowledge on the traffic characteristics. Correctness, convergence time and message complexity of the algorithm have been analyzed. SOFIA can work on top of existing solutions for route optimization. With a simple backward learning procedure it can be made robust with respect to flow classification errors. SOFIA has been implemented and tested in a MapReduce cluster on a real OpenFlow controller, hence proving that it is a promising solution for production environments.

In the future we aim at reducing the signaling overhead of SOFIA. Instead of letting switches report asynchronously to the controller at each round, we will study how to make control messages reactive to traffic changes. Also, we plan to consider the constraint on limited switch memory, which may prevent the installation of a new custom forwarding rule.

References

[1] N. McKeown, T. Anderson, H. Balakrishnan *et al.*, "OpenFlow: Enabling Innovation in Campus Networks," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 69–74, Mar. 2008.

[2] B. A. A. Nunes, M. Mendonca, X. N. Nguyen, K. Obraczka, and T. Turletti, "A survey of software-defined networking: Past, present, and future of programmable networks," *IEEE Commu. Surveys Tutorials*, vol. 16, no. 3, pp. 1617–1634, February 2014.

[3] T. Benson, A. Akella, and D. A. Maltz, "Network traffic characteristics of data centers in the wild," in *Proc. ACM IMC*, Melbourne, Australia, November 1-3, 2010.

[4] A. Tootoonchian, S. Gorbunov, Y. Ganjali, M. Casado, and R. Sherwood, "On Controller Performance in Software-defined Networks," in *Proc. USENIX Hot-ICE*, 2012.

[5] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 4, pp. 63–74, Aug. 2008.

[6] C. Hopps, "RFC2992: analysis of an equal-cost multi-path algorithm," United States, 2000.

[7] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat, "Hedera: Dynamic Flow Scheduling for Data Center Networks," in *Proc. USENIX NSDI*, 2010.

[8] A. R. Curtis, W. Kim, and P. Yalagandula, "Mahout: Low-overhead datacenter traffic management using end-host-based elephant detection," in *Proc. IEEE INFOCOM*, 2011.

[9] J. Dean and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," *Comm. of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[10] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: Cluster computing with working sets," in *Proc. USENIX HotCloud*, 2010.

[11] S. Kandula, S. Sengupta, A. Greenberg, P. Patel, and R. Chaiken, "The nature of data center traffic: measurements & analysis," in *Proc. ACM SIGCOMM*, 2009.

[12] "Introducing data center fabric, the next-generation Facebook data center network," https://code.facebook.com/posts/360346274145943/, 2014.

[13] K. Kannan and S. Banerjee, "Compact TCAM: Flow entry compaction in TCAM for power aware SDN," in *Proc. IEEE ICDCN*, 2013.

[14] H. Wang, L. Chen, K. Chen, Z. Li, Y. Zhang, H. Guan, Z. Qi, D. Li, and Y. Geng, "Flowprophet: generic and Accurate Traffic Prediction for Data-Parallel Cluster Computing," in *Proc. IEEE ICDCS*, 2015.

[15] M. Chowdhury and I. Stoica, "Efficient coflow scheduling without prior knowledge," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 45, no. 4, pp. 393–406, Aug. 2015.

[16] 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. ACM SIGCOMM*, 2016.

[17] M. Moshref, M. Yu, R. Govindan, and A. Vahdat, "DREAM: dynamic resource allocation for software-defined measurement," *ACM SIGCOMM Computer Comm. Review*, vol. 44, no. 4, pp. 419–430, 2015.

[18] M. Malboubi, L. Wang, C. N. Chuah, and P. Sharma, "Intelligent SDN based traffic (de)Aggregation and Measurement Paradigm (iSTAMP)," in *Proc. IEEE INFOCOM*, 2014.

[19] M. Yu, L. Jose, and R. Miao, "Software Defined Traffic Measurement with OpenSketch," in *Proc. USENIX NSDI*, 2013.

[20] B. Claise, "Cisco systems NetFlow services export version 9," 2004.

[21] S. Paris, A. Destounis, L. Maggi, G. S. Paschos, and J. Leguay, "Controlling flow reconfigurations in SDN," in *IEEE INFOCOM*, 2016.

[22] T. Wang, F. Liu, and H. Xu, "An Efficient Online Algorithm for Dynamic SDN Controller Assignment in Data Center Networks," *IEEE/ACM Trans. on Networking*, vol. PP, no. 99, pp. 1–14, June 2017.

[23] H. Xu and B. Li, "TinyFlow: Breaking elephants down into mice in data center networks," in *Proc. IEEE LANMAN*, 2014.

[24] W. Wang, Y. Sun, K. Salamatian, and Z. Li, "Adaptive path isolation for elephant and mice flows by exploiting path diversity in datacenters," *IEEE Trans. on Network and Service Management*, vol. 13, no. 1, pp. 5–18, January 2016.

[25] X. N. Nguyen, D. Saucez, C. Barakat, and T. Turletti, "OFFICER: A general optimization framework for OpenFlow rule allocation and endpoint policy enforcement," in *Proc. IEEE INFOCOM*, 2015.

[26] 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 *Network Protocols (ICNP), 2016 IEEE 24th International Conference on*. IEEE, 2016, pp. 1–6.

[27] E. Altman, *Constrained Markov decision processes*. CRC Press, 1999.

[28] B. Korte and J. Vygen, *Approximation Algorithms*. Springer, 2012.

[29] M. J. Neely, "Stochastic network optimization with application to communication and queueing systems," *Synthesis Lectures on Comm. Networks*, vol. 3, no. 1, pp. 1–211, 2010.

[30] H. J. Kushner and G. G. Yin", *Stochastic Approximation and Recursive Algorithms and Applications"*. Springer, 2nd Edition, 2003.

[31] B. N. Levine and J. Garcia-Luna-Aceves, "A Comparison of Reliable Multicast Protocols," *Multimedia Syst.*, no. 5, pp. 334–348, Sep. 1998.

[32] S. K. Fayazbakhsh, Y. Lin, A. Tootoonchian *et al.*, "Less Pain, Most of the Gain: Incrementally Deployable ICN," in *Proc. ACM SIGCOMM*, 2013.