Joint Optimization of Flow Table and Group Table for Default Paths in SDNs

Gongming Zhao, Hongli Xu, *Member, IEEE*, Shigang Chen, *Fellow, IEEE*, Liusheng Huang, *Member, IEEE*, and Pengzhan Wang

***Abstract*— Software-defined networking (SDN) separates the control plane from the data plane to ease network management and provide flexibility in packet routing. The control plane interacts with the data plane through an interface that configures the forwarding tables, usually including a flow table and a group table, at each switch. Due to high cost and power consumption of ternary content addressable memory, commodity switches can only support flow/group tables of limited size, which presents serious challenge for SDN to scale to large networks. One promising approach to address the scalability problem is to deploy aggregate default paths specified by wildcard forwarding rules. However, the multi-dimensional interaction among numerous system parameters and performance/scalability considerations makes the problem of setting up the flow/group tables at all switches for optimal overall layout of default paths very challenging. This paper studies the joint optimization of flow/group tables in the complex setting of large-scale SDNs. We formulate this problem as an integer linear program, and prove its NP-hardness. An efficient algorithm with bounded approximation factors is proposed to solve the problem. The properties of our algorithm are formally analyzed. We implement the proposed algorithm on an SDN test bed for experimental studies and use simulations for large-scale investigation. The experimental results and simulation results show that, under the same number of flow entries, our method can achieve better network performance than the equal cost multipath while reducing the use of group entries by about 74%. Besides, our method can reduce the link load ratio and the number of flow entries by approximately 13% and 60% compared with DevoFlow with 10% additional group entries.**

***Index Terms*— Software defined networks, default paths, load balancing, flow table, group table.**

I. Introduction

SOFTWARE defined networking (SDN) separates the control and data planes to different devices. Consisting of one or a cluster of controllers, the control plane provides logically centralized management by deciding and installing proper forwarding rules on switches. The switches, which comprise the data plane of an SDN network, perform packet forwarding based on the installed rules. Thanks to SDN's flexibility in network management and capability in rapid deployment of new functionalities, there is an increasing interest of deploying SDN in different networking environments, such as wide-area networks [2], and data centers [3].

In an SDN network, the control plane interacts with the data plane through an interface that configures the forwarding tables. As specified in the OpenFlow standard [4], each SDN switch usually has two forwarding tables for installing rules: *the flow table* and *the group table*. Each entry in the flow table (also called flow entry) specifies an action for flows that match the fields in the entry. Each entry in the group table (also called group entry) can specify more than one action. A flow entry may refer to a group entry in order to apply multiple actions to its matching flows; this mechanism can be used to support more complex operations, such as multi-path forwarding and multicasting [5].

One serious challenge faced by SDN is that the sizes of flow/group tables are very limited on today's commodity switches. Considering the process speed, price and energy consumption of the switch, most of today's commodity switches only support just 2-20K entries [6]. Moreover, the limited number of flow/group entries may have to be shared by routing/performance/measurement/security functions that are implemented on the same chip. For example, if the controller expects some flows to be processed by middleboxes, there should install extra rules for these flows on switches [7]. Hence, the memory for storing forwarding rules is often small, which is a limiting factor on the scalability of the network. Yet, large SDN networks are experiencing more and more flows. For example, in a practical data center network with 1,500 server clusters [8], the average arrival rate reaches 100k flows per second. If we perform per-flow routing, it will require a large number of flow entries at each switch. Since the switches do not have enough flow entries for so many flows, we are unable to provide service for new flows or we have to replace existing entries in the table with new forwarding rules, which causes churns and increases the controller load to repetitively deploy paths for the same flows. Therefore, *per-flow routing is impractical for large-scale networks* [9], [10].

To address the size limitation of the flow/group tables, an interesting idea is to deploy aggregate routing (or default paths) specified by wildcard rules. For example, we may perform prefix aggregate routing (instead of per-flow routing), where flows with the same address prefix will share a common path. As a result, it requires fewer flow entries for all flows. However, since all flows that match a flow entry will always

Manuscript received August 25, 2017; revised January 23, 2018 and April 11, 2018; accepted June 8, 2018; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor S. Uhlig. Date of publication August 2, 2018; date of current version August 16, 2018. This work was supported in part by the NSFC under Grant 61472383, Grant U1709217, Grant 61728207, and Grant 61472385, and in part by the Natural Science Foundation of Jiangsu Province in China under Grant BK20161257. The work of S. Chen was supported by the NSF under Grant STC-1562485 and Grant CNS-1719222. Some preliminary results of this paper were published in the Proceedings of IEEE ICNP 2017 [1]. *(Corresponding author: Hongli Xu.)*

G. Zhao, H. Xu, L. Huang, and P. Wang are with the School of Computer Science and Technology, University of Science and Technology of China, Hefei 230027, China, and also with the Suzhou Institute for Advanced Study, University of Science and Technology of China, Suzhou 215123, China (e-mail: zgm1993@mail.ustc.edu.cn; xuhongli@ustc.edu.cn; lshuang@ustc.edu.cn; pzwang@mail.ustc.edu.cn).

S. Chen is with the Department of Computer & Information Science & Engineering, University of Florida, Gainesville, FL 32611 USA (e-mail: sgchen@ufl.edu).

Digital Object Identifier 10.1109/TNET.2018.2853587

be forwarded to the same next hop, it may cause imbalance in traffic load distribution, where some paths are congested while alternative paths are left under-utilized [2]. One may say that we can combine aggregate routing and per-flow routing to avoid load imbalance. However, our simulation results show that the number of required flow entries for the combined routing scheme is still unacceptable for many commodity switches, especially with a large number of flows. To choose some large (or elephant) flows for per-flow routing [11], [12], we need to know the traffic intensity of all flows in the network. Thus, the flow statistics collection (FSC) is necessary. However, FSC is time-consuming and resource-intensive in a large-scale dynamic network [10], [13]. Thus, the aggregate routing (or combined with per-flow routing) may not work well for many practical applications.

The idea of multi-path routing holds great promise of solving the dilemma between the desire for network performance and the practical limitation in the number of forwarding rules. To support multi-path forwarding, a flow entry may refer to a group entry, in which multiple next hops are specified. The matching flows will be randomly dispatched to the multiple downstream paths [14], [15]. ECMP is a widely used multi-path routing protocol in large-scale networks as it provides load balancing over equal cost paths through group tables [16]. However, ECMP performs poorly in asymmetric topologies, which are common topologies in today's networks due to link failures and heterogeneous network components [15]. Moreover, ECMP directly installs group entries on switches when there exist some equal-cost paths to the destination. This method does not consider the group table size constraint and the weight allocation of each action bucket. Since the number of group entries is less than the number of flow entries, and the number of processing rules (i.e., action buckets specified in OpenFlow) supported by each group entry is limited, how to efficiently use these group entries is also a challenge. Therefore, *alternative solutions adapted to both asymmetric and symmetric topologies under group table size and action buckets constraints are in urgent need.*

The prior works [14]–[16] have focused on how to achieve multi-path packet forwarding operations at the switches after the controller has already decided the multi-path routing strategy of each flow (source-destination pair). This paper addresses the complementary problem at the controller on how to optimally decide the default paths for all flows and the corresponding flow/group rules at all switches. This is a fundamental and complex problem that directly affects network performance.

It is highly desired to reduce the number of entries that are used to support default paths, so that more flow entries can be reserved for supporting other policies (*e.g.*, middlebox deployment, management and security [7]) and more group entries can be used for other purposes such as multicasting [5]. This is however a difficult undertaking. To address this challenge, we study the joint optimization of flow table and group table in a large-scale network. We formulate this problem as an integer linear program, and prove its NP-hardness. A rounding-based algorithm with bounded approximation factors is proposed to solve the problem. The properties of the algorithm are formally analyzed. We implement the proposed algorithm on an SDN testbed for experimental studies and use simulations for large-scale investigation. The experimental results and simulation results show that, under the same number of flow entries, our method can achieve better network performance than ECMP while reducing the use of group entries about 74%. Besides, with 10% additional group entries, our method can reduce link load ratio about 13% compared with DevoFlow while reducing the use of flow entries about 60%.

The rest of this paper is organized as follows. Section II introduces the preliminaries and problem definition. In Section III, we propose an algorithm to deal with the DP-JFG problem, and give some discussion. The testing results and the simulation results are given in Section IV. Section V reviews the related works on the deployment of default paths. We conclude the paper in Section VI.

II. Preliminaries and Problem Formulation

A. Network Model

A software defined network typically consists of three device sets: a terminal set, $U = \{u_1, \ldots, u_m\}$, with $m = |U|$; an SDN switch set, $V = \{v_1, \ldots, v_n\}$, with $n = |V|$; and a cluster of controllers. The controllers are responsible for route selection of all flows in the network and will not participate in the packet forwarding. These switches and terminals comprise the data/forwarding plane of an SDN. Thus, on the view of the data plane, the network topology can be modeled by a directed graph $G = (U \cup V, E)$, where E is the directed link set in the network. For ease of expression, let $c(e)$ and $E(v)$ denote the capacity of link $e \in E$ and the set of links outgoing from switch $v \in V$ in graph G, respectively.

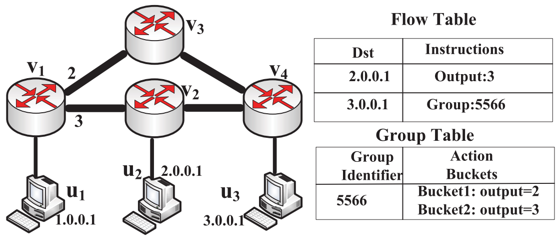
The OpenFlow specification (from the earlier version 1.1 [17] to the latest version 1.5.1 [18]) defines two main types of tables in the logical switch architecture: the *Flow Table* and *Group Table*. In an SDN network, each controller will interact with switches through the flow table and group table. More specifically, if a switch can match an incoming packet with a particular flow entry, the action specified by this flow entry will be performed. If no matched flow entry is found, the switch will report the header packet of this flow to the controller, which then determines the flow's route and installs the forwarding rules on the switches along this path. In order to support multi-path forwarding or multicast, a group table is necessary. Due to the high price of Ternary Content Addressable Memory (TCAM), the forwarding table size of an SDN switch is usually limited. Most commodity switches contain less than 4k flow entries [10], [19] and 1k group entries [20]. For example, Pica8 P-3290 switch only support about 2k flow entries [21], which is not enough for per-flow routing in most large-scale networks. For ease of reference, Table I summarizes the key notations.

B. Interaction Between Flow Table and Group Table

We introduce the operational interaction between the flow table and group table. When a flow reaches a switch, the header packet will be matched with entries in the flow table. If there is a matched flow entry, the packet will be processed based on the *Instructions field* of this flow entry, such as forwarding to a certain port or dropping. Alternatively, it may refer to a specific group entry, which mainly contains *Group Identifier field and Action Buckets field*. More specifically, the *group identifier field* uniquely identifies the group entry, and the *Action Buckets field* specifies the complex operational rules for the matched flow(s). A group table is unable to work without the help of a flow table. In other words, *a group entry will be matched and executed only if*

TABLE I
KEY NOTATIONS

Symbol	Semantics
U	a set of terminals
V	a set of SDN switches
E	a set of network links
$c(e)$	the capacity of link e
$E(v)$	the set of links outgoing from switch v
$G(v)$	the number of available group entries on switch v
Γ	a set of macroflows
Γ^u	a set of macroflows with destination u
$f(\gamma)$	the traffic demand of macroflow $\gamma \in \Gamma$
$\mathbb{P}_\gamma$	a feasible path set for macroflow $\gamma \in \Gamma$
h	the maximum number of buckets of each group entry
y_γ^p	the traffic fraction of γ forwarding by path $p \in \mathbb{P}_\gamma$
x_e^u	whether part of traffic forwarded to u through link e
g_v^u	whether $\gamma \in \Gamma^u$ needs a group entry on switch v
$I_{v,p}^u$	whether the next-hop of path p (with destination u) overlaps with the pre-deployed path on v
e_v^u	the link overlapped with pre-deployed path from switch v to destination u
$E^u(v)$	the set of outgoing links from switch v except for e_v^u



Flow Table

Dst	Instructions
2.0.0.1	Output:3
3.0.0.1	Group:5566

Group Table

Group Identifier	Action Buckets
5566	Bucket1: output=2 Bucket2: output=3

Fig. 1. Illustration of Interaction between Flow Table and Group Table. The right two plots denote the flow table and group table on switch v_1. When v_1 receives packets whose destination is u_3 with IP address 3.0.0.1, this switch will find a matched flow entry. Since its instructions field is pointed to a group entry 5566, the switch then finds the entry with group identifier 5566. Finally, these packets will be processed by the action buckets in the matched group entry.

a flow entry uses an appropriate instruction that refers to its group identifier.

We illustrate the interaction between the flow table and group table by an example. As shown in Fig. 1, assume that server u_1 needs to forward packets to server u_3. Two feasible paths $u_1 - v_1 - v_3 - v_4 - u_3$ and $u_1 - v_1 - v_2 - v_4 - u_3$ can be used. For switch v_1, there are two next-hop switches $\{v_2, v_3\}$. In order to achieve load balancing, we install one flow entry and one group entry. The flow table and group table are shown in Fig. 1. Specifically, we need a flow entry for u_3 as follows: *Match field* is $Dst =$ 3.0.0.1 and *Instructions field* is $Group:$ 5566, which is the identifier for the group entry. The action buckets in the group entry contain two buckets (output:2 and output:3), which mean that packets matching this group entry will be forwarded to port 2 or port 3 (based on given weights). If the given weights of ports 2 and 3 are 0.4 and 0.6, 40% and 60% of the matched traffic will be forwarded to port 2 and port 3, respectively. Note that a macroflow may contain many microflows, and each microflow is forwarded to a single path to avoid packet-reordering. Packets are processed as follows: when switch v_1 receives packets whose destination is u_3 with IP address 3.0.0.1, this switch will find a matched flow entry. Since its instructions field is pointed to a group entry 5566, the switch finds the entry with group identifier 5566 in the group table. Finally, these packets will be processed by the action buckets in the matched group entry.

C. Default Path by Joint Optimization of Flow Table and Group Table (DP-JFG)

This section provides a more precise description of the DP-JFG problem. Similar to [9], [10], [22], we assume that the controllers have pre-deployed aggregate paths based on destination terminals (*e.g.*, OSPF-based paths). That is, each switch has installed a flow entry for each destination. Thus, the number of occupied flow entries for default paths on each switch is nearly equal to the number of terminals in U, which is usually less than the number of flow entries. Note that our proposed algorithm and theorems are also applicable for other schemes of pre-deployed paths (*e.g.*, prefix-match paths based on rack or edge switch). It will be discussed in Section III-D.

In this paper, we mainly consider default path routing. For simplicity, all flows with the same source terminal and destination terminal will be aggregated into one macroflow. The set of all macroflows in the network is denoted by Γ. Let Γ^u denote all the macroflows with the same destination terminal $u \in U$. Due to the prior work of traffic matrix estimation on SDNs [23], it is reasonable to assume that the controller knows the traffic demand, denoted by $f(\gamma)$, of each macroflow $\gamma \in \Gamma$ through long-term observation. One may say that the traffic demand of each flow may change drastically. To solve this challenge, similar to the previous methods [10], [22], we also re-route some elephant flows for better network performance, such as load balancing and throughput maximization, which will be discussed in Section III-D. We use $\mathbb{P}_\gamma$ to represent a set of feasible paths from source to destination for each macroflow $\gamma \in \Gamma$. These paths can be pre-computed based on the network topology and dynamically updated at the controller by an OSPF-like protocol after link state information is collected from all switches. When computing the feasible paths, if there exist security or management policies, we should take these policies into consideration [7]. Hence, we assume that if there is any user-specified policy, the pre-computed paths will conform. We further discuss $\mathbb{P}_\gamma$ in Section III-A.

As described in Section II-B, if more than one default path for the same destination is deployed, we need to use group entries on some switches. When a group entry is installed, we should specify each action bucket and its weight. For simplicity, let $G(v)$ denote the number of available group entries for deploying default paths on an SDN switch v. Since some group entries should be reserved for other applications, such as multicast and broadcast, $G(v)$ is less than the maximum number of group entries on a switch v [5]. Note that, after installing the group entries, we should modified the instruction fields of some flow entries so as to refer to the corresponding group entries. For each macroflow, we will add some (or zero) feasible paths as default paths subject to following two constraints. (1) The number of required group entries on each switch should not exceed $G(v)$. (2) Due to the capacity limitation, we assume that each group entry can only support up to h buckets. For example, h is 4 for the Broadcom Trident switch [20]. Note that, with our proposed method, the number of required flow entries on any switch is nearly equal to the number of terminals (edge switches or tacks) in a

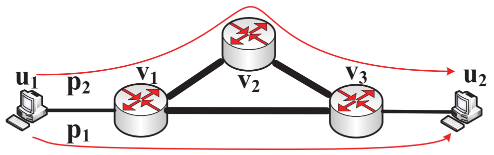


Fig. 2. Illustration of variable $I^u_{v,p}$. The pre-deployed path from switch v_1 to terminal u_2 is $v_1 \to v_3 \to u_2$. There are two feasible paths from u_1 to u_2, $p_1 = u_1 \to v_1 \to v_3 \to u_2$ and $p_2 = u_1 \to v_1 \to v_2 \to v_3 \to u_2$. The next hop of v_1 on p_2 is v_2, which does not overlap with the pre-deployed path from v_1 to u_2, so $I^{u_2}_{v_1,p_2} = 1$. On the contrary, for path p_1, $I^{u_2}_{v_1,p_1} = 0$.

network, which means the number of flow entries is sufficient to deploy default paths by our method. Thus, we do not need to consider the flow table size constraint in this problem. Our objective is to achieve load balancing in a network.

We will formulate the DP-JFG problem as an integer linear program. Let variable $y^p_\gamma \in [0,1]$ denote the traffic proportion of macroflow γ through path p. Variable $x^u_e \in \{0,1\}$ denotes whether some traffic forwarded to terminal u will pass through link $e \in E$ or not. We use variable $g^u_v \in \{0,1\}$ to denote whether the macroflow set Γ^u consumes one entry towards the group table size constraint on switch $v \in V$ or not. Let $I^u_{v,p}$ be a binary constant as follows: if the next hop of switch v on path p overlaps with the pre-deployed path from switch v to destination u, we set $I^u_{v,p} = 0$, which means that there is no need to install a group entry on switch v; otherwise $I^u_{v,p} = 1$. An example is presented in Fig. 2 to explain this variable. We assume that the pre-deployed path from switch v_1 to terminal u_2 is $v_1 \to v_3 \to u_2$. There are two feasible paths from u_1 to u_2, $p_1 = u_1 \to v_1 \to v_3 \to u_2$, and $p_2 = u_1 \to v_1 \to v_2 \to v_3 \to u_2$. For switch v_1, its next hop switch on p_2 is v_2, which does not overlap with the pre-deployed path from switch v_1 to terminal u_2, so $I^{u_2}_{v_1,p_2} = 1$. That means, when path p_2 is selected as one of default paths, a group entry should be installed on switch v_1. On the contrary, the next hop of switch v_1 on p_1 is v_3, which overlaps with the pre-deployed path from switch v_1 to terminal u_2, so $I^{u_2}_{v_1,p_1} = 0$. We formulate the DP-JFG problem as follows:

$$\min\ \lambda$$
$$S.t.\begin{cases}\sum\limits_{p\in\mathbb{P}_\gamma} y^p_\gamma = 1, & \forall\gamma\in\Gamma\\ y^p_\gamma \le x^u_e, & \forall e\in p,\ p\in\mathbb{P}_\gamma, \gamma\in\Gamma^u\\ \sum\limits_{e\in E(v)} x^u_e \le h, & \forall v\in V,\ u\in U\\ \sum\limits_{v\in p: p\in\mathbb{P}_\gamma} I^u_{v,p}\cdot y^p_\gamma \le g^u_v, & \forall\gamma\in\Gamma^u,\ u\in U,\ v\in V\\ \sum\limits_{u\in U} g^u_v \le G(v), & \forall v\in V\\ \sum\limits_{\gamma\in\Gamma}\ \sum\limits_{e\in p: p\in\mathbb{P}_\gamma} y^p_\gamma f(\gamma) & \\ \qquad\qquad \le \lambda c(e), & \forall e\in E\\ y^p_\gamma\in[0,1], & \forall p\in\mathbb{P}_\gamma,\ \gamma\in\Gamma\\ x^u_e, g^u_v\in\{0,1\}, & \forall u\in U,\ v\in V\end{cases} \tag{1}$$

The first set of equations represents that each macroflow will be forwarded through one or several feasible paths from source to destination. The second set of inequalities denotes whether fraction of traffic forwarded to terminal u will pass through link e or not. If $x^u_e = 1$, that means some traffic forwarded to terminal u will pass through link e. Combining the second and third constraints, we can guarantee that all traffic from set Γ^u can be forwarded by no more than h ports on any switch (*i.e.*, action buckets constraint). The fourth set of inequalities denotes that we need to install a group entry for macroflow $\gamma \in \Gamma^u$ on switch v if some traffic of $\gamma \in \Gamma^u$ passes through non-pre-deployed paths on switch v (*i.e.*, $I^u_{v,p} = 1$). The next two sets of inequalities indicate the group table size and link capacity constraints, respectively. Our objective is to achieve load balancing on all links, that is, $\min\ \lambda$.

Since we formalize the DP-JFG problem as a complex integer program, it is usually an NP-Hard problem.

Theorem 1: The DP-JFG problem is NP-hard.

We show that the Identical Parallel Machines Scheduling (IPMS) problem [24] is a special case of the DP-JFG problem. The detailed proof has been relegated to Appendix A.

III. Algorithm Description for the DP-JFG Problem

A. Rounding-Based Algorithm for DP-JFG

In this section, we design a rounding-based algorithm for deploying default paths, called RBDP. To solve the integer linear program in Eq. (1), the algorithm first constructs a linear program as a relaxation of the DP-JFG problem. More specifically, DP-JFG assumes that the traffic of each macroflow γ can be forwarded through at most h ports (or outgoing links) on each switch. In the relaxed version, the traffic of each macroflow γ can be arbitrarily split on any switch v and the number of required group entries is permitted to be fractional. We formulate the linear program LP_1.

$$\min\ \lambda$$
$$S.t.\begin{cases}\sum\limits_{p\in\mathbb{P}_\gamma} y^p_\gamma = 1, & \forall\gamma\in\Gamma\\ \sum\limits_{v\in p: p\in\mathbb{P}_\gamma} I^u_{v,p} y^p_\gamma \le g^u_v, & \forall\gamma\in\Gamma^u,\ u\in U,\ v\in V\\ \sum\limits_{u\in U} g^u_v \le G(v), & \forall v\in V\\ \sum\limits_{\gamma\in\Gamma}\ \sum\limits_{e\in p: p\in\mathbb{P}_\gamma} y^p_\gamma f(\gamma) & \\ \qquad\qquad \le \lambda c(e), & \forall e\in E\\ y^p_\gamma\in[0,1], & \forall p\in\mathbb{P}_\gamma,\ \gamma\in\Gamma\\ g^u_v\in[0,1], & \forall u\in U,\ v\in V\end{cases} \tag{2}$$

Since Eq. (2) is a linear program, we can solve it in polynomial time with a linear program solver. Assume that the optimal solutions for Eq. (2) are denoted by $\widetilde{y}$ and $\widetilde{g}$, and the optimal result is denoted by $\widetilde{\lambda}$. As Eq. (2) is a relaxation of the DP-JFG problem, $\widetilde{\lambda}$ is a lower-bound result for this problem. Then, we let $\widetilde{y}^u_v = \max\{\widetilde{y}^u_v, 1 - z^u_v\}$. This operation will be explained after Eq. (6).

In the second step, we determine how to install group entries on each switch for default paths. We obtain an integer solution $\widehat{g}^u_v$ using the rounding method [25], [26]. More specifically, we set $\widehat{g}^u_v = 1$, which means that a group entry will be installed on switch v for terminal u, with the probability of $\widetilde{g}^u_v$. If $\widehat{g}^u_v = 0$, this means that all macroflows with destination u will be forwarded through the link overlapped with pre-deployed path on switch v and there is no need to install a group entry.

For each $\widehat{g}_v^u = 1$, we discuss how to choose other $h-1$ forwarding ports/links at most besides the link overlapped with pre-deployed path for macroflows Γ^u on switch v and how to determine the weight of each next hop based on the solution of LP_1. For ease of expression, e_v^u denotes the outgoing link overlapped with pre-deployed path from switch v to destination u, and w_v^u denotes the weight of e_v^u in the group entry. Let $E^u(v)$ denote all the outgoing links connected with switch v except e_v^u, *i.e.*, $E^u(v) = E(v) - \{e_v^u\}$. For each destination u, each link $e \in E^u(v)$ will be assigned a weight w_e^u, initialized as 0, which denotes the weight of link in the group entry.

Based on the solution of LP_1, the total incoming traffic forwarded to destination u on switch v is:

$$\widetilde{f}(v,u) = \sum\nolimits_{\gamma\in\Gamma^u}\sum\nolimits_{v\in p:p\in\mathbb{P}_\gamma} \widetilde{y}_\gamma^p f(\gamma) \tag{3}$$

The proportion of traffic through the link e_v^u is:

$$z_v^u = \frac{\sum_{\gamma\in\Gamma^u}\sum_{e_v^u\in p:p\in\mathbb{P}_\gamma}\widetilde{y}_\gamma^p f(\gamma)}{\widetilde{f}(v,u)} \tag{4}$$

The traffic amount through other links is $(1-z_v^u)\widetilde{f}(v,u)$, and the proportion of traffic through each link $e \in E^u(v)$ is:

$$z_e^u = \frac{\sum_{\gamma\in\Gamma^u}\sum_{e\in p:p\in\mathbb{P}_\gamma}\widetilde{y}_\gamma^p f(\gamma)}{(1-z_v^u)\widetilde{f}(v,u)}. \tag{5}$$

Obviously, we know that $\sum_{e\in E^u(v)} z_e^u = 1$.

To guarantee the expected proportion of traffic through link e_v^u is z_v^u after randomized rounding, w_v^u should satisfy:

$$(1-\widetilde{g}_v^u) + \widetilde{g}_v^u \cdot w_v^u = z_v^u \Rightarrow w_v^u = \frac{z_v^u + \widetilde{g}_v^u - 1}{\widetilde{g}_v^u} \tag{6}$$

Due to $\widetilde{y}_v^u = \max\{\widetilde{y}_v^u, 1 - z_v^u\}$, we can guarantee $w_v^u \geq 0$. To choose other appropriate $h-1$ outgoing links/ports at most on switch v, the link set $E^u(v)$, with $z_e^u > 0$, is divided into two subsets, $\mathbb{P}_{v,u}^b$ and $\mathbb{P}_{v,u}^s$. First, we add each link $e \in E^u(v)$ with $z_e^u \geq \frac{1}{h-1}$ into the set $\mathbb{P}_{v,u}^b$ and the weight for link e is:

$$w_e^u = z_e^u \cdot (1 - w_v^u). \tag{7}$$

That means, we will select each link $e \in \mathbb{P}_{v,u}^b$ as one next hop and the weight for this link is w_e^u in the group entry. After that, we add each link $e \in E^u(v) - \mathbb{P}_{v,u}^b$, with $z_e^u > 0$, into the set $\mathbb{P}_{v,u}^s$. If $|\mathbb{P}_{v,u}^s| > 0$, we compute $h' = \min\{h - 1 - |\mathbb{P}_{v,u}^b|, |\mathbb{P}_{v,u}^s|\}$, which indicates that we still need to choose h' links as default paths to destination u. Then, we compute the total proportion of traffic through links $\mathbb{P}_{v,u}^s$ as follows:

$$z_{v,u}^s = \sum\nolimits_{e\in\mathbb{P}_{v,u}^s} z_e^u \tag{8}$$

For each link $e \in \mathbb{P}_{v,u}^s$, we define another variable $p(e) = \frac{h'\cdot z_e^u}{z_{v,u}^s}$, Obviously, it follows that $\sum_{e\in\mathbb{P}_{v,u}^s} p(e) = h'$.

We put all links in $\mathbb{P}_{v,u}^s$ into h' knapsacks so as to minimize the total weight of all links in each knapsack. For each knapsack j, assume that it contains a set of links, denoted by $\mathbb{P}_{v,u}^j$, and let $z_j = \sum_{e\in\mathbb{P}_{v,u}^j} p(e)$. One link $e \in \mathbb{P}_{v,u}^j$ will be chosen with probability $\frac{p(e)}{z_j}$, and the weight of this link is:

$$w_e^u = \frac{z_j \cdot z_{v,u}^s \cdot (1 - w_v^u)}{h'} \tag{9}$$

The weight for each link $e \in E(v)$ in the group entry is w_e^u. The RBDP algorithm is formally described in Alg. 1.

Algorithm 1 RBDP: Rounding-Based Algorithm for DP-JFG

```
1:  Step 1: Solving the Relaxed DP-JFG Problem
2:  Construct a linear program $LP_1$ in Eq.(2)
3:  Obtain the optimal solutions $\widetilde{y}$ and $\widetilde{g}$
4:  Compute $\widetilde{f}(v,u)$, $z_v^u$ with Eqs. (3),(4)
5:  Let $\widetilde{y}_v^u = \max\{\widetilde{y}_v^u, 1 - z_v^u\}$
6:  Step 2: Installing Entries for load balancing
7:  Derive an integer solution $\widehat{g}_v^u$ by randomized rounding
8:  for $\forall\ \widehat{g}_v^u = 1$ do
9:      Compute $w_v^u$ with Eqs. (6)
10:     for each $e \in E^u(v)$ do
11:         Compute $z_e^u$ with Eq. (5)
12:         if $z_e^u \geq \frac{1}{h-1}$ then
13:             Add $e$ into $\mathbb{P}_{v,u}^b$ and update $w_e^u$ with Eq. (7)
14:     Add link $e \in E^u(v) - \mathbb{P}_{v,u}^b$ with $z_e^u > 0$ into set $\mathbb{P}_{v,u}^s$
15:     if $|\mathbb{P}_{v,u}^s| > 0$ then
16:         Set $h' = \min\{h - 1 - |\mathbb{P}_{v,u}^b|, |\mathbb{P}_{v,u}^s|\}$.
17:         Compute $z_{v,u}^s$ with Eq. (8)
18:         for each $e \in \mathbb{P}_{v,u}^s$ do
19:             $p(e) = \frac{h'\cdot z_e^u}{z_{v,u}^s}$
20:         Put links in $h'$ knapsacks with min-max weight
21:         for for each knapsack $j$ do
22:             $z_j = \sum_{e\in\mathbb{P}_{v,u}^j} p(e)$
23:             Choose link $e \in \mathbb{P}_{v,u}^j$ with probability $\frac{p(e)}{z_j}$
24:             Update $w_e^u$ with Eq. (9) for the chosen link $e$
25:     Install a group entry on switch $v$ for terminal $u$ and the
        weight for each connected link $e \in E(v)$ is $w_e^u$.
```

Note that, the number of feasible paths connecting two terminals may be exponential and the feasible paths for different macroflows to the same destination may lead to forwarding loop. To achieve the trade-off between algorithm complexity and network performance, same as [27], we only construct some of the feasible paths for each macroflow. These feasible paths may be the shortest paths between terminals, which can be found by depth-first search. Since we consider the shortest paths for each macroflow, the forwarding loop can be avoided. For macroflow $\gamma_{u',u}$ from u' to u, if there is few (*e.g.*, only one) feasible paths, we will add other feasible paths to set $\mathbb{P}_{\gamma_{u',u}}$ as follows: We construct a directed graph $G_u = \{V, E_u\}$, where E_u is the link set in G_u and initialized as $\varnothing$. For each terminal $t \in U$, we add the feasible path set $\mathbb{P}_{\gamma_{t,u}}$ to a directed graph G_u. Given a sub-shortest path p for macroflow $\gamma_{u',u}$, after we add path p to graph G_u, there are two cases. If there is no loop in graph G_u, which means that p will not lead to forwarding loop, we add this path to $\mathbb{P}_{\gamma_{u',u}}$. Otherwise, we remove p from G_u. To decrease time complexity, the computation of feasible paths is only triggered by topology changes.

B. Performance Analysis

In this section, we will prove the correctness of our proposed algorithm and analyze its approximation performance. We first give the following lemma according to the rounding operations.

Lemma 2: Our proposed RBDP algorithm can guarantee that each macroflow γ will be forwarded through no more than h outgoing links on each switch $v \in V$.

Proof: We consider an arbitrary macroflow $\gamma \in \Gamma^u$ on switch v. If we deploy a group entry on switch v for destination u, then this macroflow γ will be forwarded based on the corresponding group entry. Under this situation, by the algorithm description, we divide the link set $E^u(v)$, with $z_e^u > 0$, into two subsets, $\mathbb{P}_{v,u}^b$ and $\mathbb{P}_{v,u}^s$. On one hand, all links in set $\mathbb{P}_{v,u}^b$ will be chosen as default paths, thus the total number of selected outgoing links from set $\mathbb{P}_{v,u}^b$ is $|\mathbb{P}_{v,u}^b|$. On the other hand, the algorithm chooses at most h' links from set $\mathbb{P}_{v,u}^s$, which means the total number of selected outgoing links from this set is no more than h'. Moreover, the link e_v^u will be included. Overall, the total number of selected output ports/links on switch $v \in V$ for Γ^u is no more than $|\mathbb{P}_{v,u}^b| + h' + 1 \leq h$. □

Lemma 3: The total weight of all the chosen links, *i.e.*, all the specified action buckets, configured on each group entry is 1.

The proof of lemma 3 has been relegated to Appendix B.

Lemma 4: The RBDP algorithm can guarantee that the expected traffic load on each link e is same as the solution $\widetilde{f}(e)$ of the linear program LP_1.

The proof of lemma 4 has been relegated to Appendix C.

In the following, we analyze the approximation performance of the proposed algorithm. Assume that the minimum capacity of all links is denoted by c_{min}^e. We define two constant values as follows:

$$\alpha = \min\{\frac{\widetilde{\lambda} c_{\min}^e}{f(\gamma)}, \gamma \in \Gamma\}, \quad \alpha' = \min\{G(v), v \in V\} \quad (10)$$

Under many practical application scenarios, the macroflow intensity is usually much less than the link capacity, because the macroflow intensity is not more than the corresponding terminal-switch link capacity [11], [27]. Besides, $G(v)$ is much larger than 1 [20]. Thus, it is reasonable to assume that $\alpha \gg 1$ and $\alpha' \gg 1$. We give the approximation performance of our algorithm.

Theorem 5: The proposed RBDP algorithm guarantees that the total traffic on any link $e \in E$ will not exceed the traffic of the fractional solution by a factor of at most $\frac{4 \log n}{\alpha} + 3$.

Due to space limit, we omit the proof of theorem 5. The reader can refer to [12] for the performance analysis of the randomized rounding method.

Theorem 6: After the rounding process, in most situation, the number of required group entries on any switch v will not exceed the number of available group entries $G(v)$ by a factor of $\frac{3 \log n}{\alpha'} + 3$.

The proof of theorem 6 has been relegated to Appendix D.

Approximation Factors: Following from our analysis, by forwarding all the flows on chosen paths, the capacity of links will hardly be violated by a factor of at most $\frac{4 \log n}{\alpha} + 3$, and the group table size constraint will not be violated by a factor of at most $\frac{3 \log n}{\alpha'} + 3$. It means that the algorithm can achieve the optimal solution, violating the link capacity constraint by at most a factor $\frac{4 \log n}{\alpha} + 3$ and the group table size constraint by at most a factor $\frac{3 \log n}{\alpha'} + 3$, which is also called as bi-criteria approximation. By scaling down the flow on each chosen path by a factor of $\frac{4 \log n}{\alpha} + 3$, link capacities are only violated with negligible probability. This can be implemented through traffic shaping on the hosts or ingress switches.

Moreover, we want to point out that the RBDP algorithm can reach almost the constant bi-criteria approximation in most situations. For example, let $\widetilde{\lambda}$ and n be 0.4 and 1000, respectively. We assume that we have a high-definition video conference (the flow intensity may reach 4Mbps) and the capacity of each switch-switch link is 100Mbps. Under this case, $\frac{c_{\min}^e}{f(\gamma)}$ will be 25. The approximation factor for the link capacity constraint is 4.2. Since $G(v)$ is usually at least 10^2 [20], the approximation factor for the group table constraint is 3.09. In other words, our RBDP algorithm can achieve almost the constant bi-criteria approximation for the DP-JFG problem in many practical network situations.

C. Complete RBDP Algorithm Description

According to theorem 6, we know that some switches may violate the group table size constraint after the rounding process. Thus, in practice, we need to remove some group entries so as to satisfy the group table size constraints on all switches. Below we give the complete RBDP algorithm so as to satisfy this constraint on all switches. The complete RBDP algorithm consists of three main steps. The former two steps are the same as those in Alg. 1. By theorem 6, some switches may violate the group table size constraint. Thus, the third step will remove some redundant group entries so as to satisfy the group table size constraints on all switches. Note that though the group entries for some macroflows are removed, they will still be forwarded through links overlapped with pre-deployed paths on these switches. Let V' denote the set of switches that violate the group table size constraint. We choose a switch v, which requires the maximum number of group entries in V' by the second step. The set of terminals, for which switch v has installed group entries, is denoted by U_v. The algorithm ranks all the terminals in U_v by the increasing order of $\widetilde{g}_v^u$. We remove the group entry for terminal $u \in U_v$ one by one, until the group table size constraint is satisfied on switch v. Then we remove switch v from V'. The iteration is terminated until all switches satisfy the group table size constraint. The complete RBDP algorithm is formally described in Alg. 2.

Algorithm 2 Complete RBDP Algorithm Description

```
1: **Step 1: Same as step 1 in RBDP**
2: **Step 2: Same as step 2 in RBDP**
3: **Step 3: Removing Some Group Entries**
4: Put all switches that violate the group table size constraint
   in set $V'$
5: **while** $V' \neq \phi$ **do**
6:     Select a switch $v \in V'$ with maximum number of
       required group entries
7:     The terminals that have installed group entries on switch
       $v$ is denoted by $U_v$
8:     Rank terminals $u \in U_v$ with the increasing order of $g_v^u$
9:     **for** each terminal $u \in U_v$ **do**
10:        Remove the group entry for terminal $u$, until the group
           table size constraint on $v$ is satisfied.
11:    $V' = V' - \{v\}$
```

D. Discussion

- In this paper, we assume that the SDN has pre-deployed terminal-based paths for simplicity. However, our proposed algorithm is also applicable for other pre-deployed path schemes. For example, assume that the network has pre-deployed prefix-match paths based on egress switches. To deal with this case, we only need to change the variables related to destination u (*e.g.*, g_v^u and x_e^u) to variables related to egress switch v_e (*e.g.*, $g_v^{v_e}$ and $x_e^{v_e}$). Specifically, variable $g_v^{v_e}$ denotes whether switch v needs to install a group entry for egress switch v_e or not, which is very similar to variable g_v^u. So, our proposed algorithm can be extended to several types of default routing, e.g. routing on prefixes. We should note that different pre-deployed path schemes may require different number of required flow entries and routing performance. The pre-defined path scheme may be determined by the application's requirement.

- Due to traffic dynamics in a network, if we deploy default paths statically, the network performance may become worse. Thus we should update the default paths to avoid sub-optimal flow routes that may cause network congestion. However, if we update default paths frequently, since each default path may match many flows, it may affect the corresponding flows and decrease network performance. Thus, to achieve trade-off between network performance and update frequency, we will re-run the RBDP algorithm in the following situations. (1) The topology changes, which will trigger the update of default paths. (2) We compute the optimal load balancing factor by LP_1 at a suitable interval (*e.g.*, 5 minutes), and compare with the current load balancing factor. If the ratio between the optimal value and the current value is less than a threshold, we should trigger the RBDP algorithm and update the deployment of default paths in an SDN network.
- During each update interval of default paths, the traffic of each individual flow also changes with time. To obtain better network performance, we adopt the combination scheme of default paths and per-flow paths [10], [22]. Specifically, we divide each (long) update interval of default paths (*e.g.*, 5 minutes) into some (short) intervals (*e.g.*, 1 minute). During each (short) interval, we first obtain the information of some elephant flows by sampling traffic [10]. Then the controller computes the least congested path for each of these elephant flows, and re-routes these flows by per-flow routing. In this case, we can achieve better network performance, which will be assessed in our simulations.

IV. Performance Evaluation

A. Performance Metrics and Benchmarks

We adopt seven main metrics for performance evaluation of our proposed algorithm. Since this paper studies how to deploy efficient default paths by joint optimization of flow/group tables on each switch, we care for the use of flow/group entries and routing performance, respectively. The first four metrics are the maximum/average number of required flow/group entries on all the switches in an SDN network. After executing these algorithms, we measure the number of installed flow/group entries on each switch, and compute the maximum/average number of installed flow/group entries in an SDN network. Moreover, we adopt link load ratio (LLR) and network throughput factor (NTF) as two metrics of the routing performance. At run-time, we measure the traffic load $f(e)$ of each connected link e, and the *link load ratio* is defined as: $LLR = \max\{f(e)/c(e), e \in E\}$. A smaller LLR means better load balancing. When there is congestion on some links, we can only forward fractional traffic to the destination. For each macroflow γ, the traffic of $\delta \cdot f(\gamma)$ at most can be forwarded from source to destination without link congestion, where δ is the *network throughput factor*, with $0 < \delta \leq 1$. The last metric is the cumulative distribution function (CDF) of the link load ratio on all links in a network.

To evaluate how well our proposed algorithm performs, we compare with other five benchmarks. The first benchmark is the most widely used OSPF method [28]. Each switch will construct the shortest path to each destination, that is, each switch will install a flow entry for each terminal. Thus, the number of required flow entries does not exceed the number of terminals in a network. We will compare our algorithm with this benchmark for routing performance while using the same number of flow entries. The second one is ECMP [14], which is widely applied for performance optimization in data center networks. ECMP needs to install group entries on switches when there exist some equal-cost paths to the destination. Otherwise, flows will be forwarded through the OSPF paths without the help of group entries. Since the number of group entries is limited, the group table size constraint may likely be violated by ECMP. To be practical, we only install group entry for destination u on switch v when there exist some equal-cost paths and have spare group entries on switch v. Otherwise, we forward flows with destination u through single path by flow entry. We denote this modified method as ECMP-G for distinguishing with ECMP. The fourth one is DevoFlow [10]. It combines pre-installed wildcard rules and dynamically-established exact-match rules for high scalability. We first deploy OSPF-based wildcard rules, then sample some elephant flows to re-route through exact-match entry under flow table size constraint so as to achieve load balancing. The last one is the optimal result for the linear program LP_1 in Eq. (2). Since LP_1 is the relaxed version of the DP-JFG problem, we denoted this result as OPT-R. OPT-R is a lower-bound for DP-JFG.

For a fair comparison, RBDP, OSPF, ECMP and ECMP-G all adopt the destination-based prefix-match scheme for default paths in our simulations, so that these methods require the same number of flow entries. Moreover, as OSPF and DevoFlow do not support the multi-path forwarding in our simulations, they need no group entry accordingly. So, we just compare RBDP with DevoFlow for the number of required flow entries and compare RBDP with ECMP for the number of group entries.

B. System Implementation on SDN Platform

1) Implementation on the Platform: We implement the OSPF, DevoFlow, ECMP, ECMP-G and RBDP algorithms on a small-scale testbed. Our SDN platform is comprised of three parts: a controller, 7 SDN-enabled physical/virtual switches and 6 virtual machines (acting as terminals). To expand the testing topology and collect testing data conveniently, we rely on virtualization technology for system implementation. Each open virtual switch (OVS, version 2.4.0) [29] and the connected Kernel-based Virtual Machines (KVMs) are deployed on a server with a core i5-3470 processor and 8GB of RAM. The topology of our SDN testbed is illustrated in Fig. 7. More specifically, $\{v_1,u_1\}$,$\{v_5,u_2,u_3,u_4\}$ and $\{v_3,u_5,u_6\}$ are

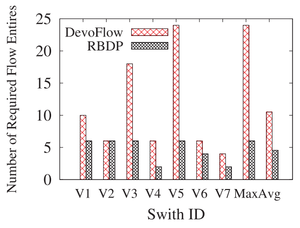


Fig. 3. Number of required flow entries on each switch.

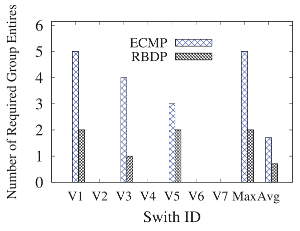


Fig. 4. Number of required group entries on each switch.

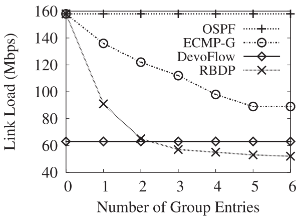


Fig. 5. Link load vs. Number of group entries.

run on 3 servers, respectively. These servers (acting as one virtual switch and several terminals) are connected with 4 H3C switches (v_2,v_4,v_6,v_7).

2) Testing Results: We run three sets of testings on the SDN platform. In each set, we generate 300 flows by default in the network, and the expected traffic intensity for each flow is 1Mbps. Moreover, we simulate a distribution with 20% of elephant flows and 80% of mice flows as observed in [10]. We first observe the number of required flow/group entries on all switches. The testing results are presented in Figs. 3-4. Fig. 3 shows that DevoFlow and RBDP need 24 and 6 flow entries at most, respectively. That's because DevoFlow combines aggregate routing and per-flow routing while RBDP only adopts aggregate routing. Fig. 4 shows that RBDP and ECMP need 5 and 2 group entries at most, respectively. That's because our RBDP algorithm takes the group table size constraint into consideration. The second set of tests observes the link load by changing the number of available group entries on each switch. Fig. 5 shows that RBDP reduces the link load by about 42% compared with ECMP-G while using the same number of flow/group entries. Moreover, with the increase of the number of group entries, the link load of RBDP is lower than that of DevoFlow. The last group of testing shows the link load by changing the number of flows. The testing results in Fig. 6 indicate that RBDP reduces the link load by about 28%, 45% and 60% compared with ECMP, ECMP-G and OSPF, respectively, and achieve similar routing performance as DevoFlow.

C. Simulation Evaluation

1) Simulation Setting: In the simulations, we selected two practical topologies, one for ISP networks and one for campus

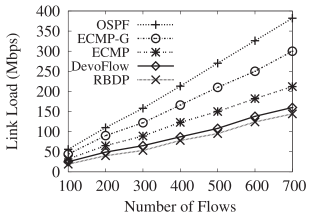


Fig. 6. Ink load vs. Number of flows.

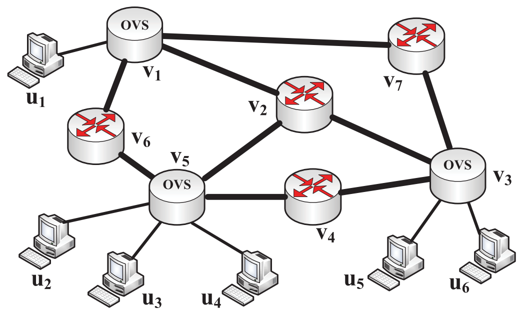


Fig. 7. Topology of the SDN testbed. Our SDN testbed consists of 7 physical/virtual switches (*i.e.*, 4 physical SDN-enabled H3C switches: v_2, v_4, v_6, v_7 and 3 Open vSwitches: v_1, v_3, v_5) and 6 virtual machines (u_1,u_2,u_3,u_4,u_5,u_6). Moreover, each vSwitch and its connected virtual machines are run on a server with a core i5-3470 processor and 8GB of RAM. For example, vSwitch v_5 and three virtual machines $\{u_2, u_3, u_4\}$ are run on the same server.

networks. The first topology is an ISP backbone network, denoted by (a), which contains 87 switches and 174 servers from Rocketfuel project [30]. The second one is a campus topology [31], denoted by (b), which contains 100 switches and 200 servers from Monash University [32]. For both topologies, each server runs 10 virtual machines (VMs), and each link has a uniform capacity, 10Gbps. We execute each simulation 100 times and average the numerical results. We conduct our simulations using realistic workloads based on empirically observed traffic patterns in real networks. Similar to [33], we consider three synthetic workloads from (1) web search services [34], (2) enterprise networks [35], and (3) data mining services [14]. The authors of [10] have shown that less than 20% of the top-ranked flows may be responsible for more than 80% of the total traffic. Thus, we allocate the size for all workloads according to this 2-8 distribution and the expected traffic demand of each flow is 0.5Mbps. We use iperf2 (Version 2.0.5) server-client tool [36] to simulate diverse kinds of flows, such as different packet size and traffic duration. First, we deploy iperf servers at some hosts, the other hosts act as clients. Then, we generate TCP/UDP flows with different MSS (Maximum Segment Size) or Packet-Size to simulate different volume of each flow. We start several clients on one host to simulate multiple applications.

The flow table size is set as 5k for the following reasons. On one hand, due to the high price and energy-consuming of TCAM, SDN switch usually contains less than 5k flow entries (*e.g.*, Broadcom Trident has 4k flow entries [20]). On the other hand, even if some commodity switches have larger rules, these rules may have to be shared by various functions (*e.g.*, security, management and flow statistics collection [7]). The number of available group entries is set as 300 on each switch by default for the same reason. Main code is published at https://github.com/sdntest/default_path. Note that, in real

TABLE II
COMPARISON ON NUMBER OF FLOW ENTRIES

No. of Flow Entries	Topology (a)		Topology (b)	
	Max.	Avg.	Max.	Avg.
DevoFlow	5k	4.3k	5k	4.4k
RBDP	2k	1.9k	2k	1.9k
ECMP	2k	1.9k	2k	1.9k
OSPF	2k	1.9k	2k	1.9k

TABLE III
COMPARISON ON NUMBER OF GROUP ENTRIES

No. of Group Entries	Topology (a)		Topology (b)	
	Max.	Avg.	Max.	Avg.
ECMP	1.4k	0.9k	1.4k	0.9k
ECMP-G	0.3k	0.23k	0.3k	0.23k
RBDP	0.3k	0.23k	0.3k	0.23k

networks, there may encounter link failures and sudden traffic changes, especially in data center networks. We have not considered these unpredictable situations in our evaluation. Thus, we can say that the evaluation is a preliminary analysis and that consistent amount of future work is still needed to cast RBDP to data center networks.

2) Simulation Results: We run four sets of simulations on two different topologies to check the effectiveness of our proposed algorithm. The first set of simulations shows the required flow/group entries by different algorithms in a network which contains 300k flows (about 150 flows per VM). We execute four algorithms on two different topologies, and the simulation results are shown in Tables II and III. From Table II, we can see that our proposed algorithm reduces the flow entries about 60% compared with DevoFlow on average. For example, for topology (a), RBDP only needs no more than 2k flow entries while DevoFlow needs about 5k flow entries at most and 4.3k flow entries on average. Note that, the number of required entries for DevoFlow will increase with the number of flows and there may contain millions of flows in some large networks. Thus, it is highly meaningful to reduce the number of flow entries that are used to support routing, so that more flow entries can be reserved for supporting other policies [7]. Since only ECMP, ECMP-G and RBDP need to install group entries on switches, we compare the use of group entries of ECMP, ECMP-G and RBDP on both two topologies. Table III shows that the number of required group entries of RBDP is same as that of ECMP-G and fewer than that of ECMP. For example, when there are 300k flows in topology (b), ECMP needs about 1.4k group entries at most and about 0.9k group entries on average while RBDP only needs about 0.3k group entries at most and 0.23k group entries on average. In other words, our proposed algorithm can reduce group entries about 74% compared with ECMP.

The second set of simulations mainly shows how the number of flows affects the routing performance on two topologies. We change the number of flows from 100k to 900k, and the simulation results are shown in Figs. 8-13. Figs. 8 and 9 show that the link load ratio increases with the number of flows for all algorithms. Our RBDP algorithm has decent link load ratio performance on both two topologies. For example, when there are 300k flows, the proposed RBDP algorithm reduces

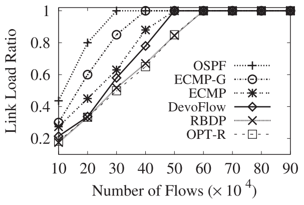


Fig. 8. LLR vs. Number of flows for topology (a).

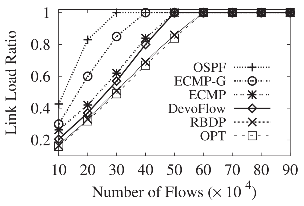


Fig. 9. LLR vs. Number of flows for topology (b).

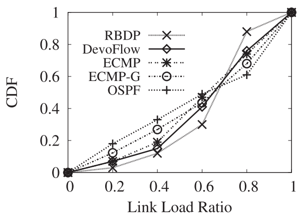


Fig. 10. CDF of LLR for topology (a).

link load ratio about 39% and 42% compared with the OSPF method in topologies (a) and (b), respectively. Meanwhile, RBDP can reduce link load ratio by about 25% compared with ECMP-G using the same number of flow/group entries, and achieve better link load ratio performance compared with ECMP, which requires more group entries than ours illustrated by Table III. Moreover, our proposed algorithm can achieve similar performance as OPT-R, which means efficiency of the approximation algorithm to solve the NP-hard problem. Even compared with DevoFlow, which increases the number of required flow entries about 60%, our proposed algorithm also can reduce link load ratio about 13%. We also give the cumulative distribution function (CDF) of the link load ratio under a fixed number (*e.g.*,50×10^4) of flows, shown in Figs. 10-11. The results indicate that our proposed RBDP algorithm reduces the variance in the LLR of all links compared with other algorithms. For example, in topology (b), over 80% of links have a link load ratio between 0.2-0.8 by our solution, while only 37% of links have a link load ratio between 0.2-0.8 and over 30% of links encounter congestion by OSPF. Figs. 12-13 indicate that the throughput factor decreases with the number of flows in both two topologies. For example, when there are 600k flows in topology (a), RBDP improves the network throughput factor about 115% compared with the OSPF method, and achieves better throughput factor compared with DevoFlow and ECMP.

The third set of simulations shows how the number of available group entries affects the routing performance on two topologies. The results are shown in Figs. 14-17. By default, there are 400k flows in Figs. 14-15. These two figures show that, the link load ratio of our RBDP algorithm is better

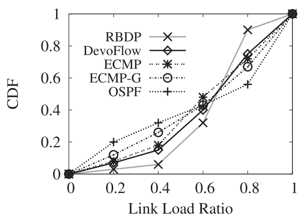


Fig. 11. CDF of LLR for topology (b).

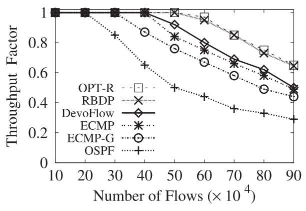


Fig. 12. NTF vs. Number of flows for topology (a).

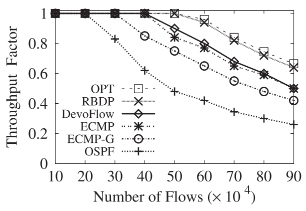


Fig. 13. NTF vs. Number of flows for topology (b).

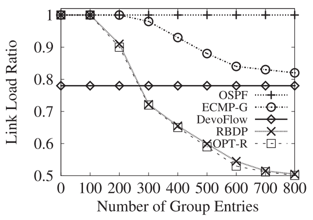


Fig. 14. LLR vs. Number of group entries for topology (a).

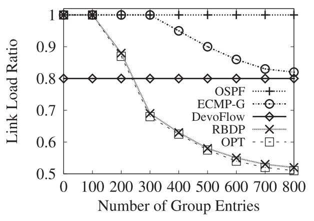


Fig. 15. LLR vs. Number of group entries for topology (b).

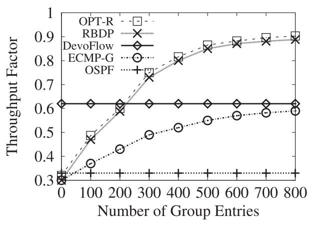


Fig. 16. NTF vs. Number of group entries for topology (a).

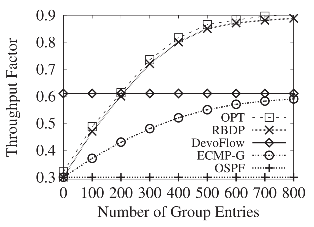


Fig. 17. NTF vs. Number of group entries for topology (b).

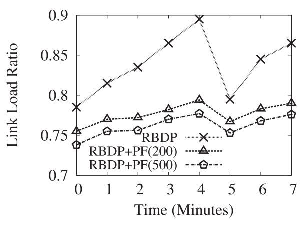


Fig. 18. Time vs. LLR for Topology (a).

than that of DevoFlow when the number of available group entries is more than 250. Meanwhile, RBDP can achieve lower link load ratio compared with ECMP-G while using the same number of flow/group entries. For example, when the group table size constraint is 600 in topology (a), RBDP reduces link load ratio by about 45% and 35% compared with OSPF and ECMP-G, respectively. By default, there are 800k flows in Figs. 16 and 17. These two figures show the throughput factor performance by changing the number of available group entries on two different topologies. When each switch contains 300 available group entries in topology (a), our proposed algorithm can increase throughput factor about 110% and 48% compared with OSPF and ECMP-G, respectively.

The last set of simulations illustrates the effectiveness of the combination scheme of default paths and per-flow paths, which has been discussed in Section III-D. To avoid sub-optimal performance caused by traffic dynamics, we re-run our RBDP algorithm every 5 minutes and select some elephant flows to re-route through per-flow routing every minute. The number of available flow entries is set to 200 and 500, respectively, to shown the effects of the number of occupied flow entries for per-flow routing. As shown in Figs. 18-19, after we deploy default paths, the network performance will get worse over time due to traffic dynamics. However, if we re-run the RBDP algorithm during a suitable interval (*e.g.*, 5 min), we can relieve the impacts to a certain extent. Since only a very small part of flows will be affected if we update per-flow paths, we can update the per-flow paths in a short interval to achieve better network performance. For example, as shown in Fig. 18, the update of default paths helps to reduce the LLR from 0.89 to 0.8, and the combination scheme can reduce the LLR from 0.89 to 0.75 with a small number of flow entries. Moreover, the combination scheme helps to keep a nice performance before the update of default paths (*e.g.*, the time from 1 minute to 5 minutes).

From these simulation results, we can draw the following conclusions. First, from Table II, RBDP reduces the number of required flow entries about 60% compared with DevoFlow. Second, RBDP reduces the number of required group entries about 74% compared with ECMP from Table III. Third, from Figs. 8-13, our algorithm improves routing performance about

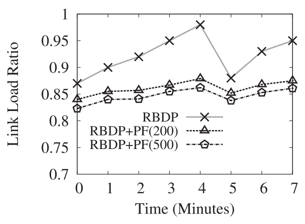


Fig. 19. Time vs. LLR for topology (b).

35% on average compared with OSPF while using 10% additional group entries, and improves routing performance about 13% on average compared with DevoFlow while reducing the flow entries about 60% and using 10% additional group entries. Moreover, RBDP can achieve similar performance compared with ECMP while reducing the number of required group entries by about 74%. Fourth, RBDP can increase routing performance by about 35% compared with ECMP-G using the same number of flow/group entries by Figs. 14-17. Fifth, the RBDP algorithm can achieve similar network performance compared with OPT-R. Last but not least, the combination scheme of default paths and per-flow paths can greatly relieve the impacts of traffic dynamics.

V. Related Works

With its advantages, SDN is becoming an innovative technology behind many traffic engineering solutions for both campus networks and data center networks. Since routing is a critical issue to achieve better network performance in an SDN network, there are many related works to handle the routing problem. To provide the fine-grained flow control, a natural way is to install one individual forwarding rule for each flow. M. Al-Fares *et al.* [9] designed a dynamic flow scheduling for data center networks that set up a rule for every new flow in the network.

As the networks are experiencing more and more flows while the commodity switches only contain a few thousand TCAM entries [10], [16], [37], these works can not be applied directly to scenarios with a massive number of flows in some practical applications. This challenge can be solved by dropping some flows or using wildcard routing. The authors of [27] considered the throughput maximization problem under the forwarding table size constraint. They formulated an offline routing optimization problem with a flow table size constraint and presented an approximate solution for the NP-hard problem. They dropped some flows so that the controller could install per-flow rules for other flows with flow-table size constraint. M. Huang *et al.* [38] studied dynamic unicast and multicast routing in SDNs under both link/switch capacities and request bandwidth demands constraints. They rejected the request if one of the constraint could not be met. However, since the dropped flows can not be served, it reduces the user experience.

To serve all flows in the network and accommodate the flow-table size constraint, default path [10], [39]–[43] is an efficient solution for SDNs. Many works based on wildcard routing have been proposed to minimize the rule space consumption on switches. Devoflow [10] combined pre-installed wildcard rules and dynamically-established exact rules. DomainFlow [44] divided the network into two parts, one part using wildcard rules and another part using exactly matching rules. However, these works did not mention how to deploy default paths.

As in [10], [45], OSPF was the widely-used method for default paths. However, these works suffered from worse network performance, for many flows would be forwarded through one single path. To improve the network performance, multipath forwarding has been proposed. Equal Cost Multipath (ECMP) [14], [46] is the most popular technique for spreading traffic among available paths, which has been applied in [2], [3], [47]. ECMP splits a set of flows uniformly over a group of next hops to achieve load balancing. However, it splits the flow space equally, rather than based on the network status. This weakness will cause network imbalance, especially when handling elephant flows or running on the network of unequal link capacities. Besides, the group table size constraint is not considered in the ECMP protocol. To handle the weakness of ECMP, WCMP [15] focused on how to establish weighted multipath to divide traffic based on a given ratio. Niagara [16] handled weighted splits through prefix and suffix. This problem was completely different from ours, which takes advantages of group table and flow table into account for deploying default paths. We should note that, to apply the default paths, some works first identified elephant flows [23], [48]–[50]. Then, they forwarded elephant flows at per-flow level, and the rest (mice) flows were forwarded through default paths. However, the specific methods for default path deployment are not mentioned or emphasized in these works.

All the above works can not achieve trade-off optimization between the occupied flow/group entries and network performance. The per-flow routing methods can achieve better network performance while the required flow entries are much more than flow table size constraint. Moreover, the previous default path methods, such as OSPF and ECMP, may lead to worse network performance or can not satisfy the group table size and action buckets constraints. In contrast, this paper focuses on deploying default paths to achieve better trade-off performance by joint optimization of flow table and group table in an SDN network.

VI. Conclusion

In this paper, we have studied efficient deployment of default paths by joint optimization of flow table and group table for an SDN. We have designed a rounding-based algorithm for the DP-JFG problem. The testing results on an SDN platform and the extensive simulation results show the high efficiency of our algorithm for default paths deployment. In the future, we will study how to deal with some unpredictable situations, including link failures and traffic abrupt changes.

Appendix A
Proof of Theorem 1

To show the NP-hardness, we first give the following definition.

Definition 7 (Identical Parallel Machines Scheduling (IPMS) Problem [24]): Given m parallel machines and q independent jobs, each job is to be assigned to one of the machines. All the parallel machines are identical in terms of their processing speed. Thus, every job will take the same amount of processing time on each machine. The objective is to find a schedule that minimizes the makespan.

Proof: In the following, we prove the NP-hardness by showing that the identical parallel machines scheduling problem [24] is a special case of the DP-JFG problem. We consider an arbitrary IPMS instant $\mathcal{S}$. There are a set of m machines

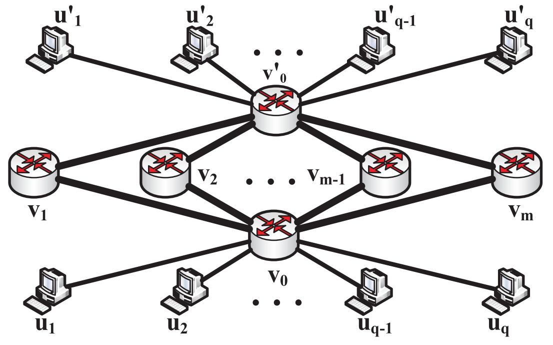


Fig. 20. A special example of the DP-JFG problem.

and a set of q jobs. Moreover, the processing time of job i on each machine is denoted as p_i.

Next, we consider a special example of the DP-JFG problem. As shown in Fig. 20, there are q macroflows in the network, in which each macroflow is from u_i to u'_i, with $1 \leq i \leq q$. The capacities of links v_0v_i and $v_iv'_0$ are set as c_0 and ∞, respectively, where c_0 is constant, *e.g.*, 1Gbps or 10Gbps. Assume that there are q available group entries on each switch, and each group entry can support 2 action buckets. Thus, for each macroflow $\gamma \in \Gamma$, we can only choose another feasible path except the pre-deployed (*e.g.*, OSPF) path as default paths. To simplify the problem, for each macroflow γ, we assume that the traffic amount through its pre-deployed path has been determined. We will choose one feasible path for each macroflow γ_i to forward its remaining traffic, $p_i \cdot c_0$, so as to minimize the maximum traffic load ratio in a network. Thus, we regard each macroflow from u_j to u'_j and each link v_0v_i as a job j and a machine i. Moreover, the processing time for each job j is $\frac{p_i \cdot c_0}{c_0} = p_i$. This is just the IPMS instant $\mathcal{S}$. As a result, each IPMS instant is a special instant of DP-JFG, which shows the NP-hardness of the DP-JFG problem. □

Appendix B
Proof of Lemma 3

Proof: By the algorithm description, for terminal u and switch v, we divide all links $E^u(v)$ with $z_e^u > 0$ into two subsets, $\mathbb{P}_{v,u}^b$ and $\mathbb{P}_{v,u}^s$. On the one hand, according to Eq. (7), the total weight of all links in $\mathbb{P}_{v,u}^b$ is:

$$w_{v,u}^b = \sum\nolimits_{e \in \mathbb{P}_{v,u}^b} z_e^u \cdot (1 - w_v^u) \tag{11}$$

On the other hand, by Eq. (9), the total weight of all links in $\mathbb{P}_{v,u}^s$ is:

$$w_{v,u}^s = \sum\nolimits_{j=1}^{k'} \frac{z_j \cdot z_{v,u}^s \cdot (1 - w_v^u)}{k'} = z_{v,u}^s \cdot (1 - w_v^u) \tag{12}$$

Combining Eqs. (8), (11) and (12), we have

$$\begin{aligned} &w_{v,u}^b + w_{v,u}^s \\ &\quad = \sum\nolimits_{e \in \mathbb{P}_{v,u}^b} z_e^u \cdot (1 - w_v^u) + \sum\nolimits_{e \in \mathbb{P}_{v,u}^s} z_e^u \cdot (1 - w_v^u) \\ &\quad = \sum\nolimits_{e \in E^u(v)} z_e^u \cdot (1 - w_v^u) = 1 - w_v^u \end{aligned} \tag{13}$$

Thus, the total weight of all the chosen links on each group entry is $w_v^u + w_{v,u}^b + w_{v,u}^s = 1$ □

Appendix C
Proof of Lemma 4

Proof: Let $\widetilde{f}(e,u)$ and $\widetilde{f}(v,u)$ denote the traffic load on link e and the traffic amount on switch v from macroflows Γ^u by the linear program LP_1, respectively. By the algorithm, the controller will derive integer solution $\widehat{g}_v^u$ one by one. We consider the situation that the controller processes the first variable g_v^u in the network. After the rounding operation, if $\widehat{g}_v^u = 1$, we install a group entry on switch v for terminal u. Otherwise, we forward all macroflows $\gamma \in \Gamma^u$ through the link overlapped with pre-deployed path from switch v to terminal u. After this operation on variable g_v^u, the traffic on switch v from macroflows Γ^u is denoted by $\widehat{f}_t(v,u)$. This operation will not affect the incoming traffic on switch v, *i.e.*,

$$\widehat{f}_t(v,u) = \widetilde{f}(v,u). \tag{14}$$

Then, we consider the traffic load on the outgoing links from switch v. The traffic load on link $e \in E(v)$ from macroflows Γ^u is denoted by $\widehat{f}_t(e,u)$ after the rounding process. We prove that, for each link $e \in E(v)$, $\mathbb{E}\left[\widehat{f}_t(e,u)\right] = \widetilde{f}(e,u)$, so that this rounding operation will not affect the expected traffic load of link $e \in E$. There are three cases of link $e \in E(v)$.

1) If link $e \in \mathbb{P}_{v,u}^b$, combining Eqs. (5), (6), (7) and (14), we know:

$$\begin{aligned} \mathbb{E}\left[\widehat{f}_t(e,u)\right] &= w_e^u \cdot \widehat{f}_t(v,u) \cdot \widehat{g}_v^u \\ &= z_e^u \cdot (1 - w_v^u) \cdot \widetilde{f}(v,u) \cdot \widetilde{g}_v^u \\ &= z_e^u \cdot \frac{1 - z_v^u}{\widetilde{g}_v^u} \cdot \widetilde{f}(v,u) \cdot \widetilde{g}_v^u \\ &= \sum\nolimits_{\gamma \in \Gamma^u} \sum\nolimits_{e \in p: p \in \mathbb{P}_\gamma} y_\gamma^p f(\gamma) = \widetilde{f}(e,u) \end{aligned} \tag{15}$$

2) If link $e \in \mathbb{P}_{v,u}^s$, combining Eqs. (5), (6), (9) and (14), we have:

$$\begin{aligned} \mathbb{E}\left[\widehat{f}_t(e,u)\right] &= \frac{p(e)}{z_j} \cdot w_e^u \cdot \widehat{f}_t(v,u) \cdot \widetilde{g}_v^u \\ &= \frac{p(e)}{z_j} \cdot \frac{z_j \cdot z_{v,u}^s \cdot (1 - w_v^u)}{h'} \cdot \widetilde{f}(v,u) \cdot \widetilde{g}_v^u \\ &= \frac{h' \cdot z_e^u}{z_{v,u}^s} \cdot \frac{z_{v,u}^s \cdot (1 - w_v^u)}{h'} \cdot \widetilde{f}(v,u) \cdot \widetilde{g}_v^u \\ &= \sum\nolimits_{\gamma \in \Gamma^u} \sum\nolimits_{e \in p: p \in \mathbb{P}_\gamma} y_\gamma^p f(\gamma) = \widetilde{f}(e,u) \end{aligned} \tag{16}$$

3) If link $e = e_v^u$, following Eqs. (4) and (6), it follows:

$$\begin{aligned} \mathbb{E}\left[\widehat{f}_t(e,u)\right] &= w_v^u \cdot \widehat{f}_t(v,u) \cdot \widetilde{g}_v^u + \widehat{f}_t(v,u) \cdot (1 - \widetilde{g}_v^u) \\ &= z_v^u \cdot \widetilde{f}(v,u) \\ &= \sum\nolimits_{\gamma \in \Gamma^u} \sum\nolimits_{e_v^u \in p: p \in \mathbb{P}_\gamma} \widetilde{y}_\gamma^p f(\gamma) = \widetilde{f}(e,u) \end{aligned} \tag{17}$$

Thus, the expected traffic load on each link e is:

$$\mathbb{E}\left[\widehat{f}_t(e)\right] = \mathbb{E}\left[\sum\nolimits_{u \in U} \widehat{f}_t(e,u)\right] = \mathbb{E}\left[\sum\nolimits_{u \in U} \widetilde{f}(e,u)\right] = \widetilde{f}(e) \tag{18}$$

In a similar way, after we install all the required group entries, the expected traffic load on each link e is same as $\widetilde{f}(e)$. □

APPENDIX D
PROOF OF THEOREM 6

We give two famous theorems for probability analysis.

Theorem 8 (Chernoff Bound): Given n independent variables: $x_1, x_2, \ldots, x_n$, where $\forall x_i \in [0,1]$. Let $\mu = \mathbb{E}[\sum_{i=1}^{n} x_i]$. Then, $\mathbf{Pr}\left[\sum_{i=1}^{n} x_i \geq (1+\epsilon)\mu\right] \leq e^{\frac{-\epsilon^2\mu}{2+\epsilon}}$, where ϵ is an arbitrarily positive value.

Theorem 9 (Union Bound): Given a countable set of n events: $A_1, A_2, \ldots, A_n$, each event A_i happens with possibility $\mathbf{Pr}(A_i)$. Then, $\mathbf{Pr}(A_1 \cup A_2 \cup \ldots \cup A_n) \leq \sum_{i=1}^{n} \mathbf{Pr}(A_i)$.

Note that, by Eq. (2), we know that $\sum_{u\in U} \widetilde{g}_v^u \leq G(v)$ before updating. Since only a few number of variables (less than 0.4%) do not obey $\widetilde{g}_v^u \geq 1 - z_v^u$, we assume that it also follows $\sum_{u\in U} \widetilde{g}_v^u \leq G(v)$ after updating in most situation.

Proof: The number of required group entry on switch v for terminal u is defined as a variable $\varphi_{v,u}$. The expected number of required group entries is:

$$\mathbb{E}\left[\sum_{u\in U} \varphi_{v,u}\right] = \sum_{u\in U} \mathbb{E}[\varphi_{v,u}] = \sum_{u\in U} \widetilde{g}_v^u \leq G(v) \tag{19}$$

Combining Eq. (19) and the definition of α' in Eq. (10), we have:

$$\begin{cases} \dfrac{\varphi_{v,u}\cdot\alpha'}{G(v)} \in [0,1] \\ \mathbb{E}\left[\sum_{u\in U} \dfrac{\varphi_{v,u}\cdot\alpha'}{G(v)}\right] \leq \alpha'. \end{cases} \tag{20}$$

Then, by applying theorem 8, assume that σ is an arbitrary positive value. It follows:

$$\mathbf{Pr}\left[\sum_{u\in U} \frac{\varphi_{v,u}\cdot\alpha'}{G(v)} \geq (1+\sigma)\alpha'\right] \leq e^{\frac{-\sigma^2\alpha'}{2+\sigma}} \tag{21}$$

Now, we assume that

$$\mathbf{Pr}\left[\sum_{u\in U} \frac{\varphi_{v,u}\cdot\alpha'}{G(v)} \geq (1+\sigma)\alpha'\right] \leq e^{\frac{-\sigma^2\alpha'}{2+\sigma}} \leq \frac{\mathcal{F}}{n} \tag{22}$$

Then, we get the result:

$$\sigma \geq \frac{\log\frac{n}{\mathcal{F}} + \sqrt{\log^2\frac{n}{\mathcal{F}} + 8\alpha'\log\frac{n}{\mathcal{F}}}}{2\alpha'}, \quad n \geq 2 \tag{23}$$

Set $\mathcal{F} = \frac{1}{n^2}$. Apparently $\mathcal{F} \to 0$ as $n \to \infty$. With respect to Eq. (23), we set

$$\begin{aligned} \sigma &= \frac{\log\frac{n}{\mathcal{F}} + \log\frac{n}{\mathcal{F}} + 4\alpha'}{2\alpha'} \\ &= \frac{6\log n + 4\alpha'}{2\alpha'} = \frac{3\log n}{\alpha'} + 2 \end{aligned} \tag{24}$$

By applying Theorem 9, following Eq. (22), we have,

$$\begin{aligned} &\mathbf{Pr}\left[\bigvee_{v\in V}\sum_{u\in U}\frac{\varphi_{v,u}}{G(v)} \geq (1+\sigma)\right] \\ &\quad \leq \sum_{v\in V}\mathbf{Pr}\left[\sum_{u\in U}\frac{\varphi_{v,u}}{G(v)} \geq (1+\sigma)\right] \\ &\quad \leq n\cdot\frac{1}{n^3} = \frac{1}{n^2}, \quad \sigma \geq \frac{3\log n}{\alpha'} + 2 \end{aligned} \tag{25}$$

Then Eq. (25) is guaranteed with $1 + \sigma = \frac{3\log n}{\alpha'} + 3$. That means, after the rounding process, the total number of required group entries on any switch v will not exceed the number of available group entries $G(v)$ by a factor of $\frac{3\log n}{\alpha'} + 3$. □

REFERENCES

[1] G. Zhao, H. Xu, S. Chen, L. Huang, and P. Wang, "Deploying default paths by joint optimization of flow table and group table in SDNs," in *Proc. IEEE ICNP*, Oct. 2017, pp. 1–10.

[2] C.-Y. Hong *et al.*, "Achieving high utilization with software-driven WAN," in *Proc. ACM SIGCOMM*, 2013, pp. 15–26.

[3] S. Jain *et al.*, "B4: Experience with a globally-deployed software defined WAN," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 43, no. 4, pp. 3–14, 2013.

[4] *OpenFlow Switch Specification Version 1.3.4*, Open Netw. Found., 2014.

[5] L.-H. Huang, H.-C. Hsu, S.-H. Shen, D.-N. Yang, and W.-T. Chen, "Multicast traffic engineering for software-defined networks," in *Proc. IEEE INFOCOM*, Apr. 2016, pp. 1–9.

[6] N. Katta, O. Alipourfard, J. Rexford, and D. Walker, "Infinite CacheFlow in software-defined networks," in *Proc. 3rd ACM Workshop Hot Topics Softw. Defined Netw.*, 2014, pp. 175–180.

[7] Z. A. Qazi *et al.*, "SIMPLE-fying middlebox policy enforcement using SDN," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 43, no. 4, pp. 27–38, 2013.

[8] S. Kandula, S. Sengupta, A. Greenberg, P. Patel, and R. Chaiken, "The nature of data center traffic: Measurements and analysis," in *Proc. ACM SIGCOMM*, 2009, pp. 202–208.

[9] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat, "Hedera: Dynamic flow scheduling for data center networks," in *Proc. NSDI*, vol. 10, 2010, p. 19.

[10] A. R. Curtis *et al.*, "DevoFlow: Scaling flow management for high-performance networks," *Comput. Commun. Rev.*, vol. 41, no. 4, pp. 254–265, Aug. 2011.

[11] H. Xu *et al.*, "Real-time update with joint optimization of route selection and update scheduling for SDNs," in *Proc. IEEE 24th Int. Conf. Netw. Protocols (ICNP)*, Nov. 2016, pp. 1–10.

[12] H. Xu *et al.*, "Joint route selection and update scheduling for low-latency update in SDNs," *IEEE/ACM Trans. Netw.*, vol. 25, no. 5, pp. 3073–3087, Oct. 2017.

[13] H. Xu, Z. Yu, C. Qian, X.-Y. Li, and Z. Liu, "Minimizing flow statistics collection cost of SDN using wildcard requests," in *Proc. IEEE INFOCOM*, May 2017, pp. 1–9.

[14] A. Greenberg *et al.*, "VL2: A scalable and flexible data center network," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 39, no. 4, pp. 51–62, 2009.

[15] J. Zhou *et al.*, "WCMP: Weighted cost multipathing for improved fairness in data centers," in *Proc. ACM 9th Eur. Conf. Comput. Syst.*, 2014, p. 5.

[16] N. Kang, M. Ghobadi, J. Reumann, A. Shraer, and J. Rexford, "Efficient traffic splitting on commodity switches," in *Proc. 11th ACM Conf. Emerg. Netw. Exp. Technol.*, 2015, pp. 1–6.

[17] *OpenFlow Switch Specification Version 1.1.0*, Operating Syst. Consortium, 2011.

[18] *OpenFlow Switch Specification Version 1.5.1*, Open Netw. Found., 2014.

[19] G. Zhao, L. Huang, Z. Yu, H. Xu, and P. Wang, "On the effect of flow table size and controller capacity on SDN network throughput," in *Proc. IEEE Int. Conf. Commun. (ICC)*, May 2017, pp. 1–6.

[20] B. Stephens, A. Cox, W. Felter, C. Dixon, and J. Carter, "PAST: Scalable Ethernet for data centers," in *Proc. ACM 8th Int. Conf. Emerg. Netw. Exp. Technol.*, 2012, pp. 49–60.

[21] H. Chen and T. Benson, "The case for making tight control plane latency guarantees in SDN switches," in *Proc. ACM Symp. SDN Res.*, 2017, pp. 150–156.

[22] H. Xu, H. Huang, S. Chen, and G. Zhao, "Scalable software-defined networking through hybrid switching," in *Proc. IEEE INFOCOM*, May 2017, pp. 1–9.

[23] Z. Hu and J. Luo, "Cracking network monitoring in DCNs with SDN," in *Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, Apr./May 2015, pp. 199–207.

[24] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness* (Series of Books in the Mathematical Sciences). San Francisco, CA, USA: Freeman, 1979, p. 340.

[25] A. Srinivasan, *Approximation Algorithms Via Randomized Rounding: A Survey* (Advanced Topics in Mathematics). Polish Scientific Publishers PWN, 1999, pp. 9–71.

[26] H. Xu *et al.*, “Incremental deployment and throughput maximization routing for a hybrid SDN,” *IEEE/ACM Trans. Netw.*, vol. 25, no. 3, pp. 1861–1875, Jun. 2017.

[27] R. Cohen, L. Lewin-Eytan, J. S. Naor, and D. Raz, “On the effect of forwarding table size on SDN network utilization,” in *Proc. IEEE INFOCOM*, Apr./May 2014, pp. 1734–1742.

[28] T. M. Thomas, II, *OSPF Network Design Solutions*, vol. 10. Indianapolis, IN, USA: Cisco Press, 2003.

[29] *Open vSwitch: Open Virtual Switch*. Accessed: Jun. 16, 2018. [Online]. Available: https://www.openvswitch.org/download/

[30] N. Spring, R. Mahajan, and D. Wetherall, “Measuring ISP topologies with Rocketfuel,” *ACM SIGCOMM Comput. Commun. Rev.*, vol. 32, no. 4, pp. 133–145, 2002.

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

[32] *The Network Topology From the Monash University*. Accessed: Jun. 16, 2018. [Online]. Available: https://ecse.monash.edu/twiki/bin/view/InFocus/LargePacket-switchingNetworkTopologies

[33] E. Vanini, R. Pan, M. Alizadeh, P. Taheri, and T. Edsall, “Let it flow: Resilient asymmetric load balancing with flowlet switching,” in *Proc. NSDI*, 2017, pp. 407–420.

[34] M. Alizadeh *et al.*, “Data center TCP (DCTCP),” *ACM SIGCOMM Comput. Commun. Rev.*, vol. 40, no. 4, pp. 63–74, Oct. 2010.

[35] M. Alizadeh *et al.*, “CONGA: Distributed congestion-aware load balancing for datacenters,” *ACM SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 4, pp. 503–514, 2014.

[36] J. Dugan, *Iperf Tutorial*. Columbus, OH, USA: JointTechs, 2010 pp. 1–4.

[37] O. Rottenstreich and J. Tapolcai, “Lossy compression of packet classifiers,” in *Proc. 11th ACM/IEEE Symp. Archit. Netw. Commun. Syst.*, May 2015, pp. 39–50.

[38] M. Huang *et al.*, “Dynamic routing for network throughput maximization in software-defined networks,” in *Proc. IEEE INFOCOM*, Apr. 2016, pp. 1–9.

[39] W. Braun and M. Menth, “Wildcard compression of inter-domain routing tables for OpenFlow-based software-defined networking,” in *Proc. 3rd IEEE Eur. Workshop Softw. Defined Netw.*, 2014, pp. 25–30.

[40] A. S. Iyer, V. Mann, and N. R. Samineni, “SwitchReduce: Reducing switch state and controller involvement in OpenFlow networks,” in *Proc. IEEE IFIP Netw. Conf.*, May 2013, pp. 1–9.

[41] M. Rifai *et al.*, “Too many SDN rules? Compress them with MINNIE,” in *Proc. IEEE GLOBECOM*, Dec. 2015, pp. 1–7.

[42] M. Rifai *et al.*, “MINNIE: AN SDN world with few compressed forwarding rules,” Ph.D. dissertation, INRIA Sophia Antipolis Méditerranée, Biot, France, 2016.

[43] B. Yan, Y. Xu, H. Xing, K. Xi, and H. J. Chao, “CAB: A reactive wildcard rule caching system for software-defined networks,” in *Proc. ACM 3rd Workshop Hot Topics Softw. Defined Netw.*, 2014, pp. 163–168.

[44] Y. Nakagawa *et al.*, “DomainFlow: Practical flow management method using multiple flow tables in commodity switches,” in *Proc. ACM CoNext*, 2013, pp. 399–404.

[45] S. Agarwal, M. Kodialam, and T. V. Lakshman, “Traffic engineering in software defined networks,” in *Proc. IEEE INFOCOM*, Apr. 2013, pp. 2211–2219.

[46] C. Hopps, *Analysis of an Equal-Cost Multi-Path Algorithm*, document RFC 2992, 2000.

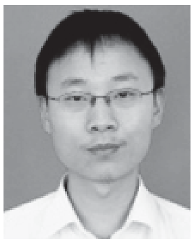
[47] P. Patel *et al.*, “Ananta: Cloud scale load balancing,” *ACM SIGCOMM Comput. Commun. Rev.*, vol. 43, no. 4, pp. 207–218, 2013.

[48] T. Benson, A. Anand, A. Akella, and M. Zhang, “MicroTE: Fine grained traffic engineering for data centers,” in *Proc. ACM 7th Conf. Emerg. Netw. Exp. Technol.*, 2011, Art. no. 8.

[49] A. R. Curtis, W. Kim, and P. Yalagandula, “Mahout: Low-overhead datacenter traffic management using end-host-based elephant detection,” in *Proc. IEEE INFOCOM*, Apr. 2011, pp. 1629–1637.

[50] J. Liu *et al.*, “SDN based load balancing mechanism for elephant flow in data center networks,” in *Proc. IEEE Int. Symp. Wireless Pers. Multimedia Commun. (WPMC)*, Sep. 2014, pp. 486–490.

Gongming Zhao is currently pursuing the Ph.D. degree in computer science with the University of Science and Technology of China. His main research interests are software-defined networks and cloud computing.



Hongli Xu (M’08) received the B.S. degree in computer science and the Ph.D. degree in computer software and theory from the University of Science and Technology of China in 2002 and 2007, respectively. He is currently an Associate Professor with the School of Computer Science and Technology, University of Science and Technology of China. He has authored or coauthored over 70 papers, and held about 30 patents. His main research interest is software-defined networks, cooperative communication, and vehicular ad hoc network.

Shigang Chen (F’16) received the B.S. degree in computer science from the University of Science and Technology of China in 1993, and the M.S. and Ph.D. degrees in computer science from the University of Illinois at Urbana–Champaign in 1996 and 1999, respectively. He served as the CTO for Chance Media Inc. from 2012 to 2014. He worked with Cisco Systems for three years before joining the University of Florida in 2002. He is currently a Professor with the Department of Computer and Information Science and Engineering, University of Florida. He has authored over 160 peer-reviewed journal/conference papers. His research interests include computer networks, Internet security, wireless communications, and distributed computing. He was a recipient of the IEEE Communications Society Best Tutorial Paper Award and the NSF CAREER Award. He holds 12 U.S. patents. He served in various chair positions or as committee members for numerous conferences. He is an ACM Distinguished Member and a Distinguished Lecturer of the IEEE Communication Society. He was an Associate Editor for the IEEE/ACM TRANSACTIONS ON NETWORKING, the IEEE TRANSACTIONS ON VEHICULAR TECHNOLOGY, and a number of other journals.

Liusheng Huang received the M.S. degree in computer science from the University of Science and Technology of China in 1988. He is currently a Senior Professor and a Ph.D. Supervisor with the School of Computer Science and Technology, University of Science and Technology of China. He has authored six books and over 300 journal/conference papers. His research interests are in the areas of Internet of Things, vehicular ad hoc network, information security, and distributed computing.

Pengzhan Wang received the B.S. degree in computer science and the Ph.D. degree in computer software and theory from the University of Science and Technology of China in 2013 and 2018, respectively. His main research interest is software-defined networks.