Deploying Default Paths by Joint Optimization of Flow Table and Group Table in SDNs

Gongming Zhao^{1,3} Hongli Xu^{*1,3} Shigang Chen^{2} Liusheng Huang^{1,3} Pengzhan Wang^{1,3}
^{1}School of Computer Science and Technology, University of Science and Technology of China, China
^{2}Department of Computer & Information Science & Engineering, University of Florida, USA
^{3}Suzhou Institute for Advanced Study, University of Science and Technology of China, China

***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 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 (TCAM), 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 testbed for experimental studies and use simulations for large-scale investigation. The experimental results and simulation results demonstrate high efficiency of our proposed algorithm.**

***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 centralized management by installing proper forwarding rules on switches. The switches, which comprise the data plane of an SDN, 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 [1], and data centers [2].

In an SDN network, the control plane interacts with the data plane through the forwarding tables. As specified in the OpenFlow standard [3], 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 [4].

One serious challenge faced by SDN is that the sizes of flow/group tables are very limited on today's commodity switches. Broadcom Trident [5] has only 4k flow entries and 1k group entries. 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 [6] [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 (*i.e.*, one flow entry for each flow), it will require tens of thousands of flow entries at each switch. Since the switches do not have enough flow entries for so many flows, we have to reject some flows [9] or 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* [10] [11].

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 be forwarded to a same next hop, it may cause imbalance in traffic load distribution, where some paths are congested while alternative paths are left under-utilized [1]. 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 [12], we need to know the traffic intensity of all flows in the network. Thus, the flow statistics collection (FSC) is necessary. However, it is time-consuming and resource-intensive for FSC in a large-scale dynamic network [11]. 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 [13] [14]. 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 [15]. However, ECMP performs poorly in asymmetric topologies, which are common topologies in today's networks due to link failures and heterogeneous network components [14]. Moreover, ECMP does not consider the detailed method to install group entries for multi-path routing. 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 [13] [14] [15] have focused on multi-path packet *forwarding operations* at the switches *after the wild-card forwarding rules are installed*. 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 [6]) and more group entries can be used for other purposes such as multicasting [4]. 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 70%. Besides, with additional 10% group entries, our method can reduce link load ratio about 13% compared with DevoFlow while reducing the use of flow entries about 60%.

II. Preliminaries and Problem Formulation

A. Network Model

A software defined network typically consists of three device sets: a cluster of controllers; an SDN switch set, $V = \{v_1, ..., v_n\}$, with $n = |V|$; and a terminal set, $U = \{u_1, ..., u_m\}$, with $m = |U|$. 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 to the latest version 1.5.1) defines two main types of tables in the logical switch architecture: the *Flow Table* and *Group Table*. In an SDN, 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. For example, most commodity switches contain less than 4k flow entries [11] and 1k group entries [5]. Even the high-end Broadcom Trident2 chipset only supports 16k flow entries [9], which is not enough for per-flow routing in most large-scale networks.

B. Interaction between Flow/group Tables

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 entry, the packet will be processed based on the *Instructions field* of this flow entry, such as dropping or forwarding to a certain port. Meanwhile, it may refer to a specific group entry, which mainly contains Group Identifier and Action Buckets fields. 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. That is, *a group entry will be matched and executed only if a flow entry uses an appropriate instruction that refers to its group identifier*. Due to space limit, the reader can refer to [3] for detailed description of the group table.

C. Default Paths 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 [10] [11] [16], 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 except terminal-based OSPF (*e.g.*, prefix-match paths based on rack or edge switch). It will be discussed in Section III-D.

In this paper, we do not consider per-flow routing. For simplicity, all flows with the same source and destination 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 $u \in U$. Due to the prior work of traffic matrix estimation on SDNs [17], it is reasonable to assume that the controller knows the traffic demand, denoted by $f(\gamma)$, of each marcoflow $\gamma \in \Gamma$. 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 network policies, we should take these policies into consideration [6]. 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. *Since each action bucket is expressed by an outgoing port (i.e., an outgoing link) for packet forwarding, the weight for each action bucket is also called the link weight in the following description*. 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 [4]. After installing the group entries, some flow entries (mainly for instructions fields) should be accordingly modified. 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 operations. For example, h is 4 for the Broadcom Trident switch [5]. Note that, the number of required flow entries on switch v is related with the number of terminals (edge switches or tacks) in a network. Thus, we do not consider the flow table size constraint. Our objective is to achieve load balancing in a network.

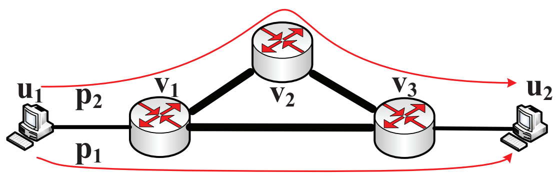

Fig. 1: Illustration of variable $I^u_{v,p}$. The OSPF path from v_1 to u_2 is $v_1 \to v_3 \to u_2$. There are two paths from u_1 to u_2, $p_1 = u_1 \to v_1 \to v_3 \to u_2$ and $p_2 = u_1 \to v_1 \to v_2 \to v_3 \to u_2$. The next hop of v_1 on p_2 is v_2, which does not overlap with the OSPF path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

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 contributes a unit 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 OSPF 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. 1 to explain this variable. We assume that the OSPF 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 OSPF 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 OSPF 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:

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

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. The third set of inequalities denotes that all traffic from set Γ^u should be forwarded by no more than h ports on switch v. 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-OSPF default 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, $\min \ \lambda$.

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

We show that the multi-commodity flow with minimum congestion problem [18] is a special case of DP-JFG. Due to space limit, we omit the detailed proof here.

III. Algorithm Description

Due to the NP-hardness, it is difficult to optimally solve the DP-JFG problem in polynomial time. This section presents an approximate algorithm, called RBDP, for DP-JFG (Section III-A). We analyze the approximation performance of the proposed algorithm (Section III-B). Then, we give the complete description of RBDP so as to satisfy the group table size constraint (Section III-C). Finally, we give some discussion on our algorithm (Section III-D).

A. Rounding-Based Algorithm

In this section, we design a rounding-based algorithm (RBDP) for deploying efficient default paths. As in Eq. (1), there is a feasible path set for each macroflow γ. However, the number of feasible paths connecting two terminals may be exponential. Moreover, as our default path scheme is destination-oriented, 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 [9], we only construct some of 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: 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.

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.

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

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.

In the second step, we determine how to install group entries on each switch for default paths. We obtain an integer solution $\widehat{g}_v^u$ using the rounding method [19]. More specifically, we set $\widehat{g}_v^u = 1$, which means that a group entry will be installed on switch v for terminal u, with the probability of $\widetilde{g}_v^u$. If $\widehat{g}_v^u = 0$, this means that all macroflows with destination u will be forwarded through the OSPF link 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 OSPF link 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 OSPF link 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. Note that, $w_v^u = w_{e_v^u}^u$.

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

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 OSPF 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 non-OSPF links is $(1-z_v^u)\widetilde{f}(v,u)$, and the proportion of traffic through each non-OSPF 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:

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

By the second set of constraints in Eq. (2), we have $\widetilde{g}_v^u \geq 1-z_v^u$. 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.

B. Performance Analysis

This section proves the correctness of our RBDP algorithm and analyzes its approximate performance. We 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 all the macroflows Γ^u on switch v. 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. On the other hand, the algorithm chooses h' links from set $\mathbb{P}_{v,u}^s$. Moreover, the OSPF link e_v^u will be included. In all, the total output ports/links on switch $v \in V$ for Γ^u is $|\mathbb{P}_{v,u}^b| + h' + 1 \leq h$. ■

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

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 proofs of lemmas 3 and 4 have been relegated to the Appendixes A and B, respectively.

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

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

Under many 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 [9] [12]. Besides, $G(v)$ is much larger than 1 [5]. 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 $\frac{4\log n}{\alpha} + 3$.

Theorem 6: After the rounding process, 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 $\frac{3\log n}{\alpha'} + 3$.

Due to space limit, we omit the proofs of theorem 5 and theorem 6. The reader can refer to [9] [12] for the performance analysis of the randomized rounding method.

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 $\frac{4\log n}{\alpha}+3$, and the group table size constraint will not be violated by a factor of $\frac{3\log n}{\alpha'} + 3$. By using the traffic controlling method, the intensity of all the flows can be limited to specific values. Thus, we can reduce the traffic of all flows by a factor of $\frac{4\log n}{\alpha} + 3$ to satisfy the link capacity constraint.

Moreover, we want to address 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. The capacity of each switch-switch link will be a bandwidth of 1Gbps at least. Observing the practical flow traces, the maximum intensity of a macroflow may reach 10Mbps. Under this case, $\frac{c_{\min}^e}{f(\gamma)}$ will be 100. The approximation factor for the link capacity constraint is 4. Since $G(v)$ is usually at least 10^2 [5], the approximation factor for the group table constraint is 3.3. 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

Though the RBDP algorithm obtains the bi-criteria approximation performance for the DP-JFG problem, the group table size constraint may not be satisfied after the randomized rounding process. 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.

Algorithm 2 Complete RBDP Algorithm Description

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

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 OSPF links 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.

D. Discussion

- In this paper, we assume that the SDN has pre-deployed terminal-based paths for simplicity. However, the 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 has decent applicability. We should note that d-ifferent 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 frequent flow dynamics in a network, if we deploy default paths statically, the network performance may become worse. So, we should update the default paths to avoid sub-optimal flow routes that may cause network congestion. To deal with this challenge, 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 min), and compare with the current load balancing factor. If the ratio between the optimal and the current values is less than a threshold, we should trigger the RBDP algorithm and update the deployment of default paths in an SDN.

IV. Performance Evaluation

A. Performance Metrics and Benchmarks

We adopt six main metrics for performance evaluation of our proposed algorithm. Since this paper studies how to deploy 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 former four metrics are the maximum/average number of required flow/group entries on all the switches in an SDN. 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. Moreover, we adopt link load ratio (LLR) and network throughput factor (NTF) as two metrics of the routing performance. During system running, we measure the traffic load $f(e)$ of each connected link e, and the *link load ratio* is defined as: $LLR = \max\{f(e)/c(e), e \in E\}$. The 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 $0 < \delta \leq 1$.

To evaluate how well our proposed algorithm performs, we compare with other four benchmarks. The first benchmark is the most widely used OSPF method [20]. 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 [13], which is widely applied for performance optimization in data center networks. The ECMP method 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, the controller should remove some group entries on switches so as to satisfy the group table size and action buckets constraints. We denote this modified method as ECMP-G for distinguishing with ECMP. The final one is DevoFlow [11]. It combines pre-installed wildcard rules and dynamically-established exact-match rules. We will compare our algorithm with this benchmark for the number of required flow entries while achieving similar network performance.

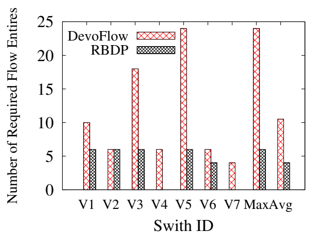

Fig. 2: Number of Required Flow Entries on Each Switch

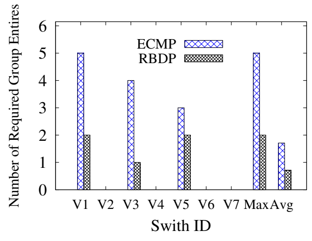

Fig. 3: Number of Required Group Entries on Each Switch

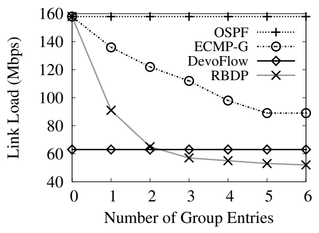

Fig. 4: Link Load vs. Number of Group Entries

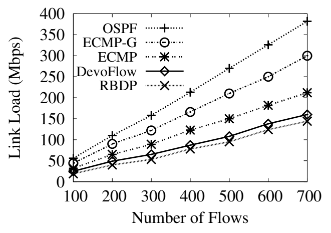

Fig. 5: Link Load vs. Number of Flows

Note that, to be fair, RBDP, OSPF, ECMP and ECMP-G all adopt destination-based prefix-match scheme for default paths in our simulations, so 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 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 adopt the virtualization technology for system implementation. Each open virtual switch (OVS, version 2.4.0) [21] and the connected Kernel-based Virtual Machines (KVMs) are implemented on a server with a core i5-3470 processor and 8GB of RAM. The topology of our SDN testbed is illustrated in Fig. 6. More specifically, $\{v_1,u_1\}$,$\{v_5,u_2,u_3,u_4\}$ and $\{v_3,u_5,u_6\}$ are run on 3 servers, respectively. These servers (acting as one virtual switch and several terminals) are connected with 4 H3C S5120-28SC-HI switches (v_2,v_4,v_6,v_7), which support the OpenFlow v1.3 standard. Besides, We use the OpenrDayLight Lithium-SR1 release [22] as the controller software running on another server with a core i5-3470 processor and 16GB of RAM. Since the controller will not participate in data forwarding, it is not explicitly included in Fig. 6 for simplicity.

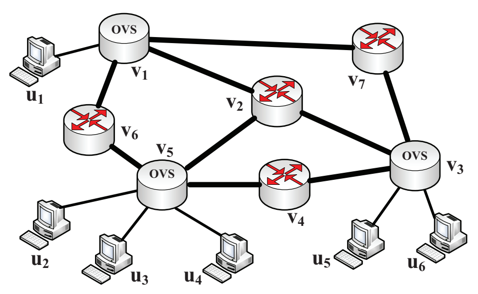

Fig. 6: Topology of the SDN testbed. Our SDN testbed consists of 7 physical/virtual switches and 6 virtual machines.

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, there are 20% elephant flows and 80% mice flows to simulate the realistic scenario [11]. We first observe the number of required flow/group entries on all switches. The testing results are presented in Figs. 2-3. Fig. 2 shows that DevoFlow and RBDP need 24 and 6 flow entries at most, respectively. RBDP can reduce the number of required flow entries by about 64% on average compared with DevoFlow. That's because DevoFlow combines aggregate routing and per-flow routing while RBDP only adopts aggregate routing. Fig. 3 shows that RBDP and ECMP need 5 and 2 group entries at most, respectively. Meanwhile, RBDP reduces the number of required group entries by about 58% on average compared with ECMP. That's because our RBDP algorithm takes the group table size constraint into consideration. The second set of testing observes the link load by changing the number of available group entries on each switch. Fig. 4 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 in an SDN. The testing results in Fig. 5 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.

From the testing results, our RBDP algorithm can (1) reduce the link load by about 60% compared with the OSPF method using the same number of flow entries and additional 10% group entries, and (2) achieve similar routing performance as DevoFlow while reducing 64% flow entries. Moreover, our proposed algorithm can (3) reduce the link load by about 45% compared with ECMP-G using the same number of flow/group entries, and (4) save the group entries about 58% compared with ECMP, whose routing performance is worse than ours.

C. Simulation Evaluation

1) Simulation Setting: We select two practical topologies. The first topology, denoted by (a), is for campus networks, and contains 100 switches and 200 servers from [23]. The second one is the fat-tree topology [24], denoted by (b), contains 80 switches and 128 servers. It has been widely applied in many datacenters. 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. The authors of [11] 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 each flow according to this 2-8 distribution and the expected traffic demand of each flow is 0.5Mbps. Besides,

No. of Flow Entries	Topology (a)		Topology (b)	
	Max.	Avg.	Max.	Avg.
DevoFlow	5k	4.4k	5k	4.5k
RBDP	2k	1.9k	1.28k	1.1k
ECMP	2k	1.9k	1.28k	1.1k
OSPF	2k	1.9k	1.28k	1.1k

TABLE I: Comparison on Number of Flow Entries

No. of Group Entries	Topology (a)		Topology (b)	
	Max.	Avg.	Max.	Avg.
ECMP	1.4k	0.9k	1.1k	0.8k
ECMP-G	0.3k	0.23k	0.3k	0.24k
RBDP	0.3k	0.23k	0.3k	0.24k

TABLE II: Comparison on Number of Group Entries

the flow table size is set as 5k for the following reasons. On the 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 [5]). On the other hand, even if some commodity switches have lager rules, these rules may have to be shared by various functions (*e.g.*, security, management and flow statistics collection [6]). The number of available group entries for default paths is set as 300 on each switch by default for the same reason.

2) Simulation Results: We run three 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 I and II. From Table I, 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 4.4k 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 [6]. 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 II 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 1.1k group entries at most and about 0.8k group entries on average while RBDP only needs about 0.3k group entries at most and 0.24k group entries on average. In other words, our proposed algorithm can reduce group entries about 70% 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. 7-10. Figs. 7 and 8 show that the link load ratio increases with the number of flows

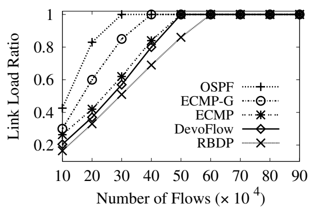

Fig. 7: LLR vs. Number of Flows for Topology (a)

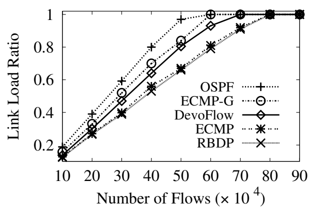

Fig. 8: LLR vs. Number of Flows for Topology (b)

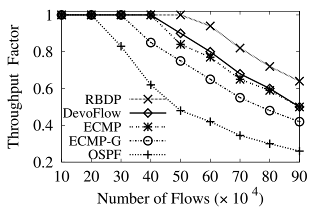

Fig. 9: NTF vs. Number of Flows for Topology (a)

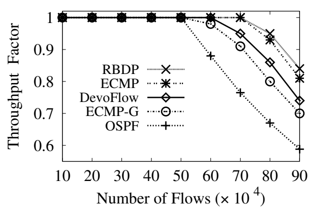

Fig. 10: NTF vs. Number of Flows for Topology (b)

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 link load ratio about 42% and 31% 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 II. 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%. Figs. 9-10 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 110% compared with the OSPF method, and achieves better throughput factor compared with DevoFlow and ECMP.

The last set of simulations shows how the number of available group entries affects the routing performance on two topologies. By default, there are 400k flows in Figs. 11-12. These two figures show that, the link load ratio of our RBDP algorithm is better 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 about 32% and 24% compared with OSPF and ECMP-G, respectively. By default, there are 800k flows in Figs. 13 and 14. 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 98% and 30% compared with OSPF and ECMP-G, respectively.

From these results, we can draw some conclusions. First, from Table I, RBDP reduces the number of required flow entries about 60% compared with DevoFlow. Second, RBDP

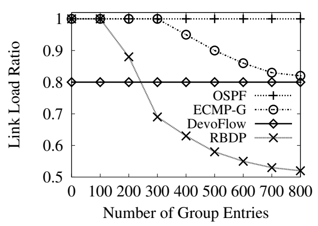

Fig. 11: LLR vs. Number of Group Entries for Topology (a)

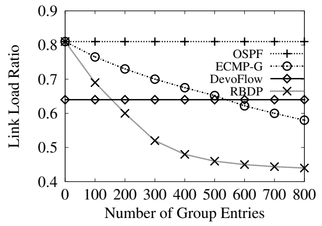

Fig. 12: LLR vs. Number of Group Entries for Topology (b)

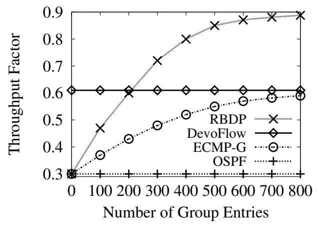

Fig. 13: NTF vs. Number of Group Entries for Topology (a)

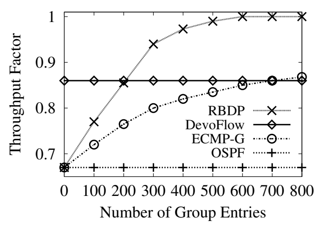

Fig. 14: NTF vs. Number of Group Entries for Topology (b)

reduces the number of required group entries about 70% compared with ECMP from Table II. Third, from Figs. 7-10, RBDP improves routing performance about 35% on average compared with OSPF while using additional 10% group entries, and improves routing performance about 13% on average compared with DevoFlow while reducing the flow entries about 60% and using additional 10% group entries. Moreover, RBDP can achieve similar performance compared with ECMP while reducing the number of required group entries by about 70%. Fourth, RBDP can increase routing performance about 30% compared with ECMP-G using the same number of flow/group entries by Figs. 11-14.

V. Related Works

Since routing is a critical issue to achieve better network performance in an SDN, 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.* [10] designed a dynamic flow scheduling for datacenter 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 [11] [15], these works can not be applied directly to scenarios with a massive number of flows. This challenge can be solved by dropping some flows or using wildcard routing. Works [9] [25] dropped some flows so that the controller could install per-flow rules for other flows with flow table size constraint. 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 [11] is an efficient solution for SDNs. Many works based on wildcard routing have been proposed to minimize the rule space consumption. DevoFlow [11] combined pre-installed wildcard rules and dynamically-established exact rules. DomainFlow [26] 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 [11], OSPF was the widely-used method for destination-based wildcard routing. However, these works suffered from worse network performance, for many flows would be forwarded through one single path. To improve the network performance, multi-path forwarding has been proposed. ECMP [13] was the most popular technique for spreading traffic among available paths, which has been applied in [1]. ECMP split a set of flows uniformly over a group of next hops to achieve load balancing. WCMP [14] focused on how to establish weighted multi-path to divide traffic based on the hash table, which likely violated the action buckets constraint in each group entry. Niagara [15] established weighted multi-path through prefix and suffix. This problem was completely different from ours.

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 is much more than the flow table size. Meanwhile, 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. Thus, 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.

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 the SDN platform and the extensive simulation results on the Mininet show the high efficiency of our algorithm.

Acknowledgement

The corresponding author of this paper is Hongli Xu. This research of Zhao, Xu, Huang and Wang is supported by NSFC under No.61472383, U1301256, 61472385 and 61728207, and NSF of Jiangsu in China under No. BK20161257.

References

[1] C.-Y. Hong, S. Kandula, R. Mahajan, M. Zhang, V. Gill, M. Nanduri, and R. Wattenhofer, “Achieving high utilization with software-driven wan,” in *ACM SIGCOMM*, 2013, pp. 15–26.

[2] D. Li, Y. Shang, and C. Chen, “Software defined green data center network with exclusive routing,” in *IEEE INFOCOM*, 2014, pp. 1743–1751.

[3] O. N. Foundation, “Openflow switch specification version 1.3.4,” 2014.

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

[5] B. Stephens, A. Cox, W. Felter, C. Dixon, and J. Carter, “Past: Scalable ethernet for data centers,” in *CoNEXT*. ACM, 2012, pp. 49–60.

[6] Z. A. Qazi, C.-C. Tu, L. Chiang, R. Miao, V. Sekar, and M. Yu, “Simple-fying middlebox policy enforcement using sdn,” *ACM SIGCOMM computer communication review*, vol. 43, no. 4, pp. 27–38, 2013.

[7] J. Fan, M. Jiang, and C. Qiao, “Carrier-grade availability-aware mapping of service function chains with on-site backups,” in *Quality of Service (IWQoS), 2017 IEEE/ACM 25th International Symposium on*. IEEE, 2017, pp. 1–10.

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

[9] R. Cohen, L. Lewin-Eytan, J. S. Naor, and D. Raz, "On the effect of forwarding table size on sdn network utilization," in *INFOCOM, 2014 Proceedings IEEE*. IEEE, 2014, pp. 1734–1742.

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

[11] A. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, and S. Banerjee, "Devoflow: Scaling flow management for high-performance networks," *Acm Sigcomm Computer Communication Review*, vol. 41, no. 4, pp. 254–265, 2011.

[12] H. Xu, Z. Yu, X.-Y. Li, C. Qian, L. Huang, and T. Jung, "Real-time update with joint optimization of route selection and update scheduling for sdns," in *Network Protocols (ICNP), 2016 IEEE 24th International Conference on*. IEEE, 2016, pp. 1–10.

[13] A. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. A. Maltz, P. Patel, and S. Sengupta, "Vl2: a scalable and flexible data center network," in *ACM SIGCOMM computer communication review*, vol. 39, no. 4. ACM, 2009, pp. 51–62.

[14] J. Zhou, M. Tewari, M. Zhu, A. Kabbani, L. Poutievski, A. Singh, and A. Vahdat, "Wcmp: Weighted cost multipathing for improved fairness in data centers," in *Proceedings of the Ninth European Conference on Computer Systems*. ACM, 2014, p. 5.

[15] N. Kang, M. Ghobadi, J. Reumann, A. Shraer, and J. Rexford, "Efficient traffic splitting on commodity switches," in *CoNEXT*, 2015, pp. 1–6.

[16] H. Xu, H. Huang, S. Chen, and G. Zhao, "Scalable software-defined networking through hybrid switching," in *Proc. IEEE INFOCOM*, 2017.

[17] Z. Hu and J. Luo, "Cracking network monitoring in dcns with sdn," in *Computer Communications (INFOCOM), 2015 IEEE Conference on*. IEEE, 2015, pp. 199–207.

[18] S. Even, A. Itai, and A. Shamir, "On the complexity of time table and multi-commodity flow problems," in *Symposium on Foundations of Computer Science*, 1975, pp. 184–193.

[19] A. Srinivasan, "Approximation algorithms via randomized rounding: a survey," *Series in Advanced Topics in Mathematics, Polish Scientific Publishers PWN*, pp. 9–71, 1999.

[20] M. Y. O. Network, "Ospf network design solutions," 2003.

[21] "Open vswitch: open virtual switch." http://openvswitch.org/.

[22] "Linux foundation collaborative project," http://opendaylight.org/.

[23] "The network topology from the monash university," http://www.ecse.monash.edu.au/twiki/bin/view/InFocus/LargePacket-switchingNetworkTopologies.

[24] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," *Acm Sigcomm Computer Communication Review*, vol. 38, no. 4, pp. 63–74, 2008.

[25] M. Huang, W. Liang, Z. Xu, W. Xu, S. Guo, and Y. Xu, "Dynamic routing for network throughput maximization in software-defined networks," in *IEEE INFOCOM*, 2016, pp. 1–9.

[26] Y. Nakagawa, K. Hyoudou, C. Lee, S. Kobayashi, O. Shiraki, and T. Shimizu, "Domainflow: Practical flow management method using multiple flow tables in commodity switches," in *ACM CoNext*, 2013, pp. 399–404.

Appendix A
Proof of Lemma 3

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

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

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

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

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

Appendix B
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. The algorithm derives an integer solution $\widehat{g}_v^u$ one by one. We consider the situation that the controller processes the first variable g_v^u. 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 OSPF link from switch v to terminal u. After rounding 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):

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

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

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

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

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

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

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

Similarly, after we install the required group entries, the expected traffic load on each link e is same as $\widetilde{f}(e)$. ■