Achieving Fine-Grained Flow Management Through Hybrid Rule Placement in SDNs

Gongming Zhao, Hongli Xu, *Member, IEEE*, Jingyuan Fan, Liusheng Huang, *Member, IEEE*, and Chunming Qiao, *Fellow, IEEE*

Abstract—Fine-grained flow management is useful in many practical applications, e.g., resource allocation, anomaly detection and traffic engineering. However, it is difficult to provide fine-grained management for a large number of flows in SDNs due to switches' limited flow table capacity. While using wildcard rules can reduce the number of flow entries needed, it cannot fully ensure fine-grained management for all the flows without degrading application performance. In this article, we design and implement hybrid rule placement for fine-grained flow management (to be referred to as *HiFi* here after). *HiFi* achieves fine-grained management with a minimal number of flow entries through taking a two-step approach: wildcard entry installment and application-specific exact-match entry installment. How to optimally install wildcard and exact-match flow entries, however, is intractable. Therefore, we design approximation algorithms with bounded factors to solve these problems. We consider how to achieve network-wide load balancing via fine-grained flow management as a case study. Both experiment on a testbed built with open virtual switches and extensive simulation show that *HiFi* can reduce the number of required flow entries by about 45-69 percent and reduce the control overhead by about 28-50 percent compared with the state-of-the-art approaches for achieving fine-grained flow management.

Index Terms—Software defined networks, fine-grained management, wildcard entry, exact-match entry, approximation

1 Introduction

A typical software defined network (SDN) consists of a logical controller in the control plane and a set of SDN switches in the data plane [2]. The controller monitors the network status and determines the forwarding paths of flows. The switches perform packet forwarding and traffic measurement for flows based on the flow entries configured by the controller. Under this framework, the controller is able to provide centralized control for flows to make network management flexible and improve the network resource utilization compared with traditional networks [3].

Compared with coarse-grained flow management, fine-grained flow management has irreplaceable advantages for some important applications in a network. For example, researchers have shown that it can improve the success ratio of portscan detection by about 35 percent through management of small (mice) flows or implementing fine-grained flow management [4]. It is also useful for resource allocation [3], anomaly detection [4], [5], traffic engineering [6], [7], and application identification [8], as well as load-balancing [9].

- *Gongming Zhao, Hongli Xu, and Liusheng Huang are with the School of Computer Science and Technology, University of Science and Technology of China, Hefei, Anhui 230027, China, and also with the Suzhou Institute for Advanced Study, University of Science and Technology of China, Suzhou, Jiangsu 215123, China. E-mail: zgm1993@mail.ustc.edu.cn, {xuhongli, lshuang}@ustc.edu.cn.*
- *Jingyuan Fan and Chunming Qiao are with the Department of Computer Science & Engineering, University at Buffalo, Buffalo, NY 16260. E-mail: jfan5@buffalo.edu, qiao@computer.org.*

Manuscript received 29 Dec. 2019; revised 4 July 2020; accepted 1 Oct. 2020. Date of publication 13 Oct. 2020; date of current version 27 Oct. 2020. (Corresponding author: Hongli Xu.) Recommended for acceptance by S. Chen. Digital Object Identifier no. 10.1109/TPDS.2020.3030630

SDN offers a great opportunity for fine-grained flow management [10]. In an SDN, when a packet arrives at an SDN switch, the switch will match this packet to all rules in the flow table. If there is a matched rule, the switch will forward this packet according to the rule's operation field. Otherwise, the switch will report the packet header to the controller, which determines the route for this flow and installs rules on the switch and on the other ones along the path. In this way, the controller can perform fine-grained management of the flow by deploying per-flow rule (e.g., identified by the 5-tuple, we call it exact-match rule) along its forwarding path. [11]. *In fact, if there is one exact-match rule along its route path, the controller has a chance to control this individual flow by modifying this exact-match rule.*

However, it is far from trivial to achieve fine-grained management with SDNs in practice. A strawman solution would be to install exact-match entries on all switches along the forwarding path of each flow. Clearly, such an approach will consume a huge number of flow entries. The problem is exacerbated due to the fact that Ternary Content Addressable Memory, commonly used in commercial SDN switches for storing flow tables/rules [12], [13], is usually expensive, power hungry and therefore size-limited [12], [14], [15]. Although the switch memory is growing [16] and some SDN switches (e.g., Noviswitch [17]) adopt RAM-based flow table for exact matches and TCAM -based flow table for wildcard matches. It still encounters some scalability issues: (1) RAM-based flow table may significantly increase the lookup latency [12], [14], [15]; (2) Encapsulating Packet-in messages is time-consuming and CPU-consuming for low-end CPU of switches (e.g., HP 5130 EI switches can only install 20 rules per second [18], [19]). (3) Encapsulating Flow-mod messages is time-consuming for SDN controller

[20]. For example, the ODL controller (run on Sun Fire X4150 server) can only encapsulate 600 Flow-mod messages per second [20]. Thus, to accommodate the processing capacity constraints of switches and controllers, it is necessary to reduce the use of flow entries on switches for fine-grained flow management .

A natural way to reduce the number of flow entries needed is to use default path. Usually, a default path is configured through wildcard rules that can match more than one flow [21], [22]. In order to manage some specific flows, one may install additional exact-match rules, as in OFFICER [23] and HS [24]. However, to distinguish those flows and install exact-match rules, it requires additional devices (e.g., monitor [25]) or software (e.g., statistical modular [24]) to be deployed in the network, which inevitably increases the system setup and maintenance cost.

To overcome the shortcomings of the existing approaches, in this paper, we build *HiFi*, a system targeted at providing fine-grained management for all flows in SDNs while minimizing the number of flow entries that need to be installed on switches without additional hardware and/or software. In other words, the goal of *HiFi* is two-fold: 1) ensuring that each flow will be forwarded by matching at least one exact-match rule along the path from source to destination; and 2) minimizing the number of flow entries needed on switches. *HiFi* enables this by taking a two-step approach: wildcard entry installment and exact-match entry installment. Specifically, wildcard rules are installed to limit the number of flow entries used, while exact-match rule installment can offer application-specific fine-grained flow management. Together, they can provide a desirable route for each flow from its source to destination. Compared with the default path scheme (e.g., OFFICER [23] and HS [24]), the significant advantage is that *HiFi* can fully achieve fine-grained management for all flows without the help of additional software modules or devices. It is worth noting that similar ideas have already been used in data center networks, where wildcard rules and exact-match rules are installed on internal switches and edge switches, respectively [26]. However, their solution is only suitable for hierarchical networks (e.g., Fat-Tree [27]), and cannot be efficiently applied to general networks (e.g., HyperX [28]). Even worse, an edge switch with a limited flow-table may become a bottleneck [29]. Therefore, a more general prototype, that can be applied to various networks and can relieve flow-table size constraints, should be proposed.

To apply the idea of *HiFi* to a general network topology, we formulate the optimal wildcard and exact-match entry installment problems using integer linear programs (ILP) by modeling fine-grained management requirements, flow-table size constraints, and link capacity constraints, etc. Unfortunately, neither problems has any optimal solutions in polynomial time. Hence, we design efficient approximation algorithms to solve them, and analyze the approximation factors.

When solving the wildcard entry installment subproblem, we also take into consideration the case where it is impossible to provide fine-grained management for all flows in a network simply because there are too many flows. For example, assume that the network consists of 100 switches, and the flow-table size of each switch is 4,000. As a result, it contains a total of 400,000 entries in the network. When there are 600,000 flows, there's no way to provide fine-grained management for all these flows. For such a case, we design an approximation algorithm to maximize the number of flows that can be controlled individually given the flow-table size constraints.

To validate our design, we implement *HiFi* in a testbed with open virtual switches without additional hardware and software, while taking long-term traffic statistics and flow-table size constraint into consideration. Both testbed experiments and large-scale simulations show that *HiFi* helps to reduce the number of required flow entries by 45-69 percent and reduce the control overhead by 28-50 percent compared with the state-of-the-art solutions.

The rest of this paper is organized as follows. Section 2 surveys the related work. Section 3 provides a motivational example and system workflow. In Sections 4.2 and 6.2, we define the wildcard entry installment subproblem for two different cases, and propose approximation algorithms respectively. Section 6.1 focuses on the load balancing as a case study. The experimental and simulation results are presented in Section 7. We conclude the paper in Section 8.

2 Related Works

SDN, with its ability to separate the control plane and the data plane, can provide fine-grained flow management for better network performance and more efficient network management [3], [7], [30]. A natural way for fine-grained management is to deploy fully exact-match rules for each flow. For example, R. Cohen *et al.* [31] proposed a method that installed exact-match entries on all switches along the forwarding path under flow-table size constraint. Similar methods have been also used in [3], [32]. However, since each flow will consume several flow entries on different switches, such schemes installing exact-match rules (to be referred to as ER here after) can only support a small number of flows, and some flows will be dropped [31], which is not fit for large-scale networks.

To save the number of flow entries, another way is to combine the default path and exact-match rules (to be referred to as DER here after) for fine-grained management. Specifically, DER first deploys default paths for all flows, and then installs exact-match rules for some (or all) flows to implement fine-grained management. However, such an approach has several weaknesses. First, when there is a default path from source to destination, all matching flows will be directly forwarded to the destination. Thus, the controller cannot capture flows that are active in the network, which makes the fine-grained flow management difficult. For some mice flows, their duration may be very short. As a result, we may not capture the existence of these mice flows, which will increase the risk of network attacks [5]. Second, each flow entry can only acquire the statistics information of aggregate flows. To derive the information of each individual flow for fine-grained management, some additional devices (e.g., monitor [25], [33]) or software (e.g., statistical modular [24]) are required for traffic measurement. It increases the cost.

Several works [13], [26], [34] attempted to achieve fine-grained management through well-designed routing methods without additional device/software. FlowStat [13]

TABLE 1
Number of Required Entries on Switches by Three Entry Installment Schemes

schemes	v_1	v_2	v_3	v_4	v_5	v_6	max	total	fine-grained
ER	2	2	4	2	2	2	4	14	✓
DER	1	1	3	1	1	1	3	8	partial
HiFi	1	2	3	1	1	1	3	9	✓

ER installs exact-match entries on all switches along the forwarding path of each flow. DER installs exact-match entries only for partial flows (not all flows). Our scheme installs exact-match entries on part of switches along a path (i.e., v_2 for p_1 and v_3 for p_2), and the other switches along a path are installed wildcard entries. As a result, our scheme supports fine-grained management with a small number of entries and without additional device/software.

proposed an adaptive flow-rule placement system in an attempt to provide fine-grained management, which consisted of three parts: path selection, rule placement and rule redistribution. They designed two greedy algorithms to select paths and install rules for each flow one by one. However, this method may cause some subsequent flows to be forwarded directly to the destination with already deployed wildcard rules. Thus, it is difficult for FlowStat to achieve fine-grained management for all flows. PACO [34] regarded each path as a concatenation of pathlets (e.g., subpaths). All information about the subpath was stored in each packet header, so that PACO could provide fine-grained path control with limit flow entries. However, PACO needs to add labels in the packet header, thus increasing the computing overhead on switches and the bandwidth consumption. Meanwhile, exact-match entries are usually installed on the edge switches, which may cause these edge switches become a bottleneck. To conquer these disadvantages, K. He *et al.* [26] proposed a practical system, called Presto. This system divided all the switches into two categories: edge (or ingress) switches and internal switches. Presto installed exact-match and wildcard entries on edge and internal switches respectively for fine-grained flow management. However, Presto also encounter the flow-table size constraint, especially for edge switches, which will be validated by simulations in Section 7.

3 Motivation and HiFi Overview

3.1 A Motivating Example

In this section, we give an example to illustrate the advantages and disadvantages of both ER and DER. The usage of entries for various entry installment schemes is summarized in Table 1.

Definition 1 (Controllable Flow). *If we achieve fine-grained management for a flow (i.e., a flow can be matched by at least one exact-match entry), we call this as a* controllable flow *or say that this flow* can be controlled.

As shown in Fig. 1, there are 4 flows in the network, 2 flows from v_3 to v_1 and the other 2 flows from v_3 to v_4. For simplicity, the intensity of each flow is set to 1. To achieve load balancing, the route configuration is illustrated in Fig. 1. Specifically, 2 flows follow the path $v_3 \rightarrow v_2 \rightarrow v_1$, and other 2 flows follow the path $v_3 \rightarrow v_6 \rightarrow v_5 \rightarrow v_4$. Accordingly, the maximum link load (i.e., 2) is minimized.

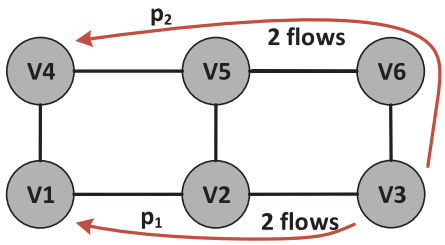


Fig. 1. A network scenario. There are 2 flows from v_3 to v_1 and the other 2 flows from v_3 to v_4. The flow intensity is set to 1 for simplicity. A load-balancing route configuration is illustrated.

To realize this routing, ER installs an exact-match entry on each switch along the forwarding path of each flow. Obviously, this scheme can support fine-grained management (i.e., 4 flows are all controllable). However, this scheme will cost more entries, e.g., the total number of consumed entries is 14 in the network (i.e., 3.5 entries per flow on average), and the maximum number of consumed entries is 4 on switch v_3. Considering the limited flow entries on commodity SDN switches, it is impractical for large-scale networks [29].

On the other hand, DER leverages the default paths to reduce the entry cost. We assume that the default path from v_3 to v_1 is $v_3 \rightarrow v_2 \rightarrow v_1$, and the default path from v_3 to v_4 is $v_3 \rightarrow v_2 \rightarrow v_1 \rightarrow v_4$. When the flows arrive, all flows will be forwarded by default paths directly. In this case, the maximum link load is 4, and no flow is controllable. To reroute some flows and achieve better load balancing, we should *deploy* additional hardware (e.g., monitor [25]) or software (e.g., statistical modular [24]) to identify those flows and determine their traffic statistics. Then, 2 flows will be re-routed to path $v_3 \rightarrow v_6 \rightarrow v_5 \rightarrow v_4$ by installing exact-match entries. As a result, only two re-routed flows can achieve fine-grained management through exact-match entries and the total number of installed flow entries is 8 in the network. Specifically, we need to install one wildcard entry and two exact-match entries on switch v_3. Although DER can save flow entries, this scheme cannot fully guarantee fine-grained management for all flows (i.e., only 2 flows are controllable). We should note that with more exact-match rules installed, more flows will be controllable.

3.2 Our Intuition

A question immediately following the above discussion is *can we do better by combining the merits of ER and DER?* Clearly, we should use as many wildcards as possible to save flow entries with the constraint that *no flow will be forwarded to the destination only with wildcard entries*. In the meantime, we should *break* the default path such that no flow will be forwarded to the destination only with wildcard entries. In other words, all flows should be controlled through at least one exact-match entry to achieve fine-grained flow control.

In Fig. 1, for flows from switch v_3 to switch v_1, we install one wildcard entry to match them on switch v_3 and switch v_1, respectively. When two flows, with egress switch v_1, arrive at v_3, they will be directly forwarded to v_2, which cannot find a matching entry for these two flows, and therefore, this flow will be reported to the controller, which can install two exact-match entries on switch v_2 for these two flows to achieve fine-

grained management. Similarly, for flows from switch v_3 to switch v_4, we install one wildcard entry on switches v_4, v_5 and v_6, respectively. Besides, we install two exact-match entries on switch v_3. As a result, the total number of installed flow entries is 9, which is almost similar to that by DER. What's more important, *our scheme can achieve fine-grained management for all flows without additional device/software*. We call this scheme as hybrid rule placement for fine-grained management or *HiFi*.

3.3 Application Scenarios for HIFI

Per-flow statistics are important for various application scenarios and many recent works are devote to achieving per-flow monitoring in SDNs, such as [13], [35]. Though fine-grained flow management, we can get per-flow statistics, which will help us better explore the advantages of centralized control in SDNs for many practical applications, e.g., resource allocation [3], anomaly detection [4], [5], traffic engineering [6], [7], and application identification [8], as well as load-balancing [9]. We give three detailed application scenarios to show the superiority of fine-grained flow management.

- Fine-grained management for all flows can improve the success ratio of anomaly detection (e.g., detecting spam, denial-of-service (DoS) attacks or worm scans [4]). For example, the information of small (mice) flows (e.g., address access patterns, connection status, traffic size and flow duration) is important for port scan detection [4]. If we adopt the coarse-grained mode, since many flows matching with a wildcard rule are aggregated into one "macroflow" [29], the controller cannot acquire the detailed information of each individual mice flow, such as traffic size and flow duration [4]. As a result, the network may encounter serious and unacceptable security issues, e.g., network attack and paralysis. On the contrary, the fine-grained management mode can provide these information through exact-match entries, and thus help to detect anomalous traffic to protect the network.
- The second example is the application-aware QoS routing. Nowadays, various applications are increasingly emerging, ranging from latency-sensitive applications to bandwidth- hungry applications. Thus, advanced traffic engineering (e.g., content-oriented processing) requires distinguishing the incoming flows of different applications through the packet headers and specifies proper routing strategy according to specific application requirements [8]. To this end, we should adopt fine-grained mode, which can provide flow-level quality of service (QoS) by matching exact-match entries. For example, we can give a higher routing priority for the traffic of specified high-level applications (e.g., adaptive video streaming and VoIP) through fine-grained flow management and management. On the contrary, if we adopt coarse-grained mode, flows that belong to different applications may be aggregated and follow the same routing policy. Thus, this mode cannot provide better QoS strategies for different applications.

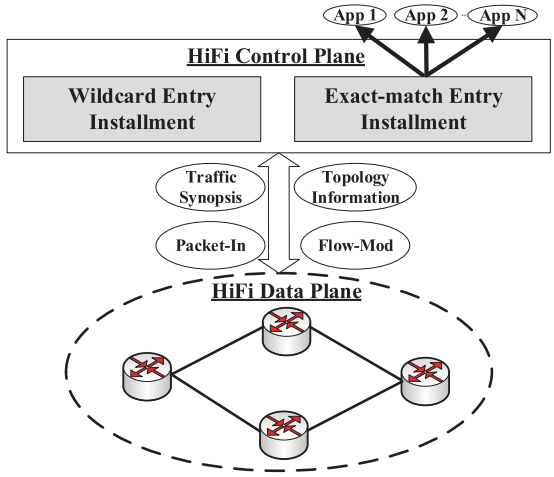


Fig. 2. HiFi architecture.

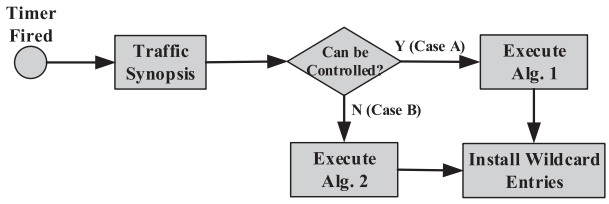
- Most of today's networks rely on middleboxes (e.g., firewalls, IPSs, NATs) to provide high security, critical performance and policy compliance capabilities [36]. Due to the dynamics and diversity of flows, different flows need to pass through different middlebox sequences (or service chains). Obviously, the fine-grained flow management mode can leverage the exact-match entry to redirect traffic to the proper middlebox. However, the coarse-grained flow mode cannot provide flow-level traffic operation control due to flow aggregation. Thus, this control mode cannot guarantee the traffic passing through the required middleboxes and may cause the network policy failure. One may say that we can design efficient wildcard rules to satisfy this requirement, it also increases the complexity of network management due to different requirements on flows. Therefore, the fine-grained management is necessary for improving the effectiveness of middleboxes.

From the above three examples, we find that the fine-grained flow management is necessary and important for many practical applications. Thus, we design *HIFI* prototype for fine-grained flow management in this paper. Note that, we try to achieve fine-grained flow management for all flows (or as many flows as possible) in the networks for simplicity in this paper. However, *HIFI* is also applicable for scenarios that only a specific set of flows need to be fine-grained controlled, which has been discussed in Section 4.5.

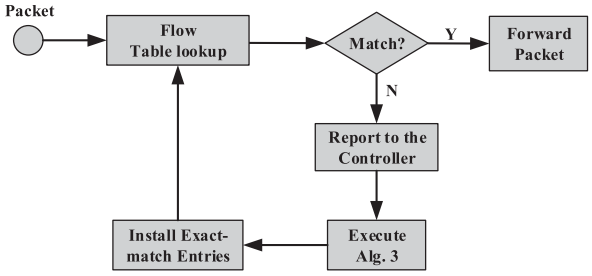
3.4 Architecture and Workflow of HiFi

As shown in Fig. 2, *HiFi* achieves fine-grained flow management through two main control plane modules: wildcard entry installment and exact-match entry installment. The former periodically determines how to install/update wildcard rules (i.e., Flow-Mod) on switches using collected traffic synopsis, while the latter installs application-specific exact-match entries (i.e., Flow-Mod) by processing flow requests (i.e., Packet-In) from the data plane.

Two constraints need to be met when installing the rules: 1) a flow should reach its destination; 2) when a flow is being forwarded to its destination, there will be at least one switch where the flow cannot find a matched wildcard rule. Thus, this switch will report the flow to the controller,



(a) Wildcard entry installment triggered by timer.



(b) Exact-match entry installment triggered by new-arrival packets.

Fig. 3. Illustration of the HiFi's workflow.

which will in turn install an exact-match rule on this switch for this flow. As a result, all following packets of this flow will be controlled through this exact-match entry. One may think that it is natural to study the joint optimization of wildcard and exact-match entries installment. However, due to traffic dynamics, it may not be feasible. If we update a wildcard entry, all flows matched with this entry will be affected, and their routes may be disrupted. Thus, it is inappropriate to update the wildcard entries frequently. Meanwhile, the exact-match entry installment is usually determined by application requirements, e.g., load balancing or throughput maximization [8], and each new-arrival flow will trigger the exact-match installment event. Thus, it is necessary to frequently update the exact-match entries to pursue different application requirements.

In *HiFi, we will trigger 1) wildcard entry installment using timer, and 2) exact-match entry installment using packets*, which are described as follows:

Step 1 of HiFi: As shown in Fig. 3a, when a timer (e.g., 10 min) expires, *HiFi* first estimates the traffic synopsis based on traffic matrix prediction [37], and decides if it is feasible for all flows to be controlled individually (which will be discussed in more detail in Section 4.3).

- If yes, which is referred to as *case A*, *HiFi* determines how to install wildcard entries to minimize the maximum flow entry utilization ratio among all switches (Section 4.2).
- Otherwise, we cannot provide fine-grained management for all flows, called *case B*. *HiFi* would maximize the number of controllable flows subject to the flow-table size constraint (Section 6.2).

Step 2 of HiFi: As illustrated in Fig. 3b, when a packet arrives at a switch, it will be handled by the flow table if a matching entry exists. Otherwise, *HiFi* would determine how to install exact-match entries for this flow (Section 6.1).

Under the HiFi architecture, wildcard entry installment module is triggered by timer and exact-match entry installment module is triggered by new-arrival packets. These two modules can run in parallel and entries on different switches can be installed in parallel.

4 Wildcard Entry Installment for Case A

In this section, we first assume that there is a feasible solution which allows all flows to be individually controlled (i.e., we are dealing with case A), and address the wildcard entry installment problem for case A (WEI-A). Moreover, we design an efficient algorithm, and then analyze the approximation performance.

4.1 Network Model

An SDN network typically consists of three device sets: a cluster of controllers; an SDN switch set, $V = \{v_1, \ldots, v_n\}$, with $n = |V|$; and a terminal set, $U = \{u_1, \ldots, u_m\}$, with $m = |U|$. The controllers monitor the network status, and are responsible for route selection of all flows in the network. The switches perform packet forwarding and traffic measurement for flows based on the flow entries configured by the controller. 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 graph $G = (U \cup V, E)$, where E is the link set in the data plane.

Assume that there is a set of wildcard rules, denoted as $\mathbb{R} = \{r_1, r_2, \ldots, r_{m'}\}$, with $m' = |\mathbb{R}|$. For example, a natural way for setting wildcard rules is as follows: we adopt the destination-based wildcard rule (e.g., destination-based OSPF method) for simplicity. Each wildcard rule r_i only specifies the destination u_i, and can match all the sources in the network. In this case, m' is equal to the number of destinations (e.g., m) in a network. The setting of these wildcard rules has been widely used in different applications, e.g., traffic engineering [29] and statistics collection [38]. Note that our proposed method is also applicable for other wildcard settings, which will be discussed in Section 4.5.

4.2 Formulation for the WEI-A Problem

Due to the prior work of traffic matrix prediction on SDNs [37], [39], it is reasonable to assume that we can obtain a flow set, denoted as Π. Moreover, the set of flows passing through switch v to destination u is denoted as Π_u^v, and Π_u denotes the set of flows with destination u. The OSPF path and the destination of each flow $f \in \Pi$ are denoted as h_f and $d(f)$, respectively. Note that, we have tested the influence of prediction accuracy on performance in Section 7.3.4.

To achieve fine-grained management, each flow will match at least one exact-match entry along its forwarding path. In other words, *each flow should not be forwarded by matching wildcard entries on all switches along the path*. In order to determine how to install wildcard entries for each destination u, we first build a tree T_u rooted at u that branches according to the flow set Π_u. For simplicity, we use the variable q_u^v to denote whether the controller will install exact-match entries on switch v for flows Π_u^v or not. There are two cases for each switch v on tree T_u.

1) $q_u^v = 0$. We install a wildcard entry on switch v for destination u. From the view of saving flow entries, it is no need to install exact-match entries even for a selected fraction of flows Π_u^v.

2) $q_u^v = 1$. The controller will install exact-match entries on switch v for flows Π_u^v. Thus, it requires $|\Pi_u^v|$ exact-match entries on switch v, or we need to reserve $|\Pi_u^v|$ entries for a flow set Π_u^v. Note that how to deploy exact-match entries depends on the current traffic and practical requirements, and will be discussed in Section 6.1.

One may think we could install a wildcard rule with a lower priority, while installing exact-match entries with higher priority for a selected fraction of flows in Π_u^v, which potentially leads to a lower cost of flow entries on switches. In fact, it is not practical on commodity switches. Assume that we have installed a wildcard entry on switch v for flows in Π_u^v. When a new flow arrives, flows will be forwarded directly through switch v by matching a wildcard entry. That means, we have no chance to install exact-match entries with higher priority for a selected fraction of flows in Π_u^v on switch v once flows arrive. Thus, we have to achieve fine-grained management on other switches for flows in Π_u^v and there is no need to install exact-match entries for a selected fraction of flows in Π_u^v on switch v.

As a result, it will cost a total number $b(v)$ of entries (including wildcard entries and reserved exact-match entries) on switch v. Each commodity switch is usually equipped with a limited number of flow entries (e.g., 4,000 entries per switch [29]), and these entries will be shared by routing/measurement/security functions [24], [40]. A natural idea is to minimize the maximum number of required flow entries among all switches. However, it may not be fair for heterogeneous switches in the network. Thus, we expect to minimize the maximum flow entry utilization ratio among all switches in the network, so that the remaining flow entries on each switch can accommodate more flows with exact-match rules. Accordingly, we formulate this problem as follows:

$$\min\ \beta$$

$$S.t. \begin{cases} \sum_{v \in h_f} q_{d(f)}^v \geq 1, & \forall f \in \Pi \\ b(v) = \sum_{u \in U: v \in T_u} (q_u^v \cdot |\Pi_u^v| + 1 - q_u^v), & \forall v \in V \\ b(v) \leq \beta \cdot s(v), & \forall v \in V \\ q_u^v \in \{0,1\}, & \forall v, u \end{cases}. \tag{1}$$

The first set of inequalities denotes that each flow will match at least one exact-match entry, which means that each flow is controllable. The second set of equalities means that the total number of required entries on each switch v is $b(v)$. Note that, $q_u^v \cdot |\Pi_u^v|$ and $1 - q_u^v$ denote the number of reserved exact-match entries and the wildcard entry on switch v for destination u, respectively. The third set of inequalities denotes that the number of consumed flow entries on switch v should not exceed $\beta \cdot s(v)$, where β is the flow entry utilization ratio and $s(v)$ is the flow-table size on switch v. The objective is to minimize the maximum flow entry utilization ratio among all switches, that is, $\min\ \beta$.

4.3 Algorithm Design for WEI-A

This section develops a rounding-based wildcard entry installment algorithm to solve the WEI-A problem. The proposed algorithm consists of three main steps. The first step will relax the integer program, denoted as LP_1, by relaxing variable q_u^v. We can solve LP_1 in polynomial time with a linear program solver, and obtain the fractional solutions, denoted as $\widetilde{q}_u^v$, $\forall v \in V, u \in U$. In the second step, the fractional solution $\widetilde{q}_u^v$ will be rounded to the 0-1 solution $\widehat{q}_u^v$ to decide where to install wildcard entries for each destination u. The set of unvisited flows is denoted as Π', initialized as all flows in Π. Moreover, we initialize $\widehat{q}_u^v = 0$, $\forall v \in V, u \in U$. We arbitrarily choose an unvisited flow, denoted as f, from set Π'. The algorithm chooses a switch with maximum $\widetilde{q}_{d(f)}^v$ among all $v \in h_f$, and set $\widehat{q}_{d(f)}^v = 1$. That is, we will not install a wildcard entry on switch v for destination $d(f)$. Thus, all flows in set $\Pi_{d(f)}^v$ can be controlled on switch v. We update $\Pi' = \Pi' - \Pi_{d(f)}^v$. This step will terminate when all flows are visited. In the third step, we install wildcard entries based on rounding solutions. For each destination u and switch $v \in T_u$, we install one wildcard entry on switch v for destination u if $\widehat{q}_u^v = 0$. The algorithm is described in Algorithm 1.

Algorithm 1. Wildcard Entry Installment for WEI-A

```
Step 1: Solving the Relaxed WEI-A Problem
Construct the relaxed problem LP_1
Obtain the fractional solutions q̃_u^v, ∀v ∈ V, u ∈ U
Step 2: Deriving the 0-1 Solution
Π' = Π
Let Φ represent the empty set
q̂_u^v = 0, ∀v ∈ V, u ∈ U
while Π' ≠ Φ do
  Arbitrarily choose an unvisited flow f from Π'
  Choose a switch with maximum q̃_{d(f)}^v among all v ∈ h_f,
  and set q̂_{d(f)}^v = 1, Π' = Π' − Π_{d(f)}^v
end while
Step 3: Installing Wildcard Entries
for Each destination u ∈ U do
  for Each switch v ∈ T_u do
    if q̂_u^v = 0 then
      Install a wildcard entry on switch v for u
    end if
  end for
end for
```

Note that, after the above algorithm completes, if we find that the total number of required flow entries on some switch exceeds its flow-table size, it means that we cannot provide fine-grained management for all flows due to flow-table size constraint. We will discuss that case in Section 6.2 (the WEI-B problem).

4.4 Performance Analysis

We analyze the approximation performance of the proposed algorithm. Let θ be the maximum number of switches visited by each flow. We first give a lemma according to the rounding operation.

Lemma 1. *After rounding operation, we have* $\widetilde{q}_u^v \geq \frac{1}{\theta} \cdot \widehat{q}_u^v$, $\forall v \in V, u \in U$.

Proof. We prove this lemma by the following two cases of each variable $\widehat{q}_u^v$. On one hand, $\widehat{q}_u^v = 0$. Obviously, $\widetilde{q}_u^v \geq \frac{1}{\theta} \cdot \widehat{q}_u^v$. On the other hand, $\widehat{q}_u^v = 1$. According to the second step of Algorithm 1, we know that there exists a flow f that allows $\widetilde{q}_u^v =$

$\max\{\widetilde{q}_u^{v'}, \forall v' \in h_f\}$. By the first set of inequalities in Eq. (1), we can prove that $\widetilde{q}_u^v \geq \frac{1}{\theta}$. Combining the above two cases, we have $\widetilde{q}_u^v \geq \frac{1}{\theta} \cdot \widehat{q}_u^v, \forall v \in V, u \in U$. □

After solving the linear program in the first step of the proposed algorithm, we derive the fractional solutions $\widetilde{q}_u^v$, $\forall v \in V, u \in U$, and the result $\widetilde{\beta}$ for the relaxed WEI-A problem. Obviously, $\widetilde{\beta}$ is the lower bound for the optimal solution obtained from Eq. (1). According to the algorithm description, the required number of flow entries on switch v is

$$\begin{aligned} &\sum\nolimits_{u \in U: v \in T_u} (\widehat{q}_u^v \cdot |\Pi_u^v| + 1 - \widehat{q}_u^v) \\ \leq &\sum\nolimits_{u \in U: v \in T_u} (\theta \cdot \widetilde{q}_u^v \cdot (|\Pi_u^v| - 1) + 1) \\ \leq &\theta \cdot \widetilde{\beta} \cdot s(v) \leq \theta \cdot \beta \cdot s(v). \end{aligned} \tag{2}$$

The first inequality is correct because $|\Pi_u^v| - 1 \geq 0$ for each $v \in T_u$. Thus, we can conclude that:

Theorem 2. *The proposed algorithm can achieve the θ-approximation for the WEI-A problem.*

4.5 Discussion

- In some practical scenarios, only a specific set of flows (or applications) requires to be controlled with fine-grained management. For example, in a bank network system, there are 10 servers depositing key data, and only flows towards these key servers should be controlled. The other flows can be aggregated for network scalability and resource reusability. To deal with this case, we only need to change the flow set Π in Eq. (1) to the set of flows towards these key servers. Then we can minimize the maximum number of required flow entries to provide fine-grained management for these flows.
- In this paper, we install wildcard entries according to destination-based OSPF. Note that it is also applicable for other wildcard entry installment schemes. For example, assume that we want to install wildcard entries based on egress switches. To deal with this case, we only need to build the tree T_v for each switch $v \in V$ according to the flows whose egress switch is v. Thus, our proposed algorithm has strong applicability.

5 Wildcard Entry Installment for Case B

In Section 4.2, we assume that the flow-table size of each switch is enough to support all flows with fine-grained management. However, when there are too many flows in a large-scale network, we may not be able to provide fine-grained management for all flows due to flow-table size constraint. In this section, we solve the Wildcard Entry Installment problem for case B (WEI-B).

5.1 Definition of the WEI-B Problem

Under this situation, we just select partial flows for fine-grained management and others for coarse-grained management so as to serve all flows with flow-table size constraint. We formulate this problem as follows:

$$\max \sum\nolimits_{f \in \Pi} \xi_f$$

$$S.t. \begin{cases} \xi_f \leq \sum_{v \in h_f} q_{d(f)}^v, & \forall f \in \Pi \\ \sum_{u \in U: v \in T_u} (q_u^v \cdot |\Pi_u^v| + 1 - q_u^v) \leq s(v), & \forall v \in V, \\ \xi_f, q_u^v \in \{0, 1\}, & \forall f, v, u \end{cases} \tag{3}$$

where ξ_f denotes whether flow f is controllable or not. The first set of inequalities denotes that flow f is controllable if this flow will match at least one exact-match entry along the path. The second set of inequalities describes that the required flow entries on each switch $v \in V$ should not exceed its flow-table size $s(v)$. The objective is to maximize the number of controllable flows, which is helpful for different applications, such as traffic engineering or attack detection [3], [4].

Algorithm 2. Maximizing Controllable Flows for WEI-B

```
1: Π̄ = Φ
2: while |V| > 0 do
3:   Step 1: Choosing a switch with the maximum profit
4:   for each switch v_i ∈ V do
5:     Apply the FPTAS method of 0-1 knapsack to compute the
       maximum profit p(v_i) for each switch v_i with knapsack
       size s(v_i) − w(v_i)
6:   end for
7:   Select switch v' with the maximum profit
8:   The installed wildcard rules on v' is denoted as ℝ'
9:   for each wildcard rule r_u ∈ ℝ' do
10:    Π̄ = Π̄ + Π_u^{v'}
11:  end for
12:  V = V − {v'}
13:  Step 2: Updating the profit of each flow set
14:  for each switch v_i ∈ V do
15:    for each wildcard rule r_j ∈ ℝ do
16:      p(Π_{u_j}^{v_i}) = |Π_{u_j}^{v_i} − Π̄|
17:    end for
18:  end for
19: end while
```

5.2 Algorithm Design for WEI-B

We give an approximation algorithm based on 0-1 knapsack to solve this problem. Before algorithm description, we consider a special case in which there is only one switch in the network. Assume that we have installed wildcard entries for all flows on switch v, and the number of occupied flow entries is $w(v)$. Then we need to replace some wildcard entries with exact-match entries to control some flows for fine-grained management. We can regard this special case as the 0-1 knapsack problem [41]. More specifically, the size of knapsack (or switch v) is the number of residual flow entries, i.e., $s(v) - w(v)$. For each destination u, Π_u^v can be regarded as an individual object. If we install exact-match entries for flows with destination u, then we should install $|\Pi_u^v|$ exact-match entries (one for each flow) and delete the corresponding wildcard entry. Thus, it will increase $|\Pi_u^v| - 1$ flow entries. That is, the cost of Π_u^v is $c(\Pi_u^v) = |\Pi_u^v| - 1$. Besides, the profit of each set Π_u^v, denoted as $p(\Pi_u^v)$, is the

number of uncontrolled flows in set Π_u^v. It can be solved by the previous knapsack algorithms, e.g., [42].

This algorithm consists of $|V|$ (i.e., the number of all switches) iterations and each iteration has two steps. In the first step, we adopt the fully polynomial time approximation scheme (FPTAS) algorithm [42] to solve the 0-1 knapsack problem for each residual switch. Then we choose a switch, denoted as v', with the maximum profit among all the residual switches. The FPTAS method for the 0-1 knapsack problem also determines the value of $q_u^{v'}$ for all $u \in \{u \in U : v' \in T_u\}$ (i.e., the individual objects that are put into the knapsack v'). In the second step, the algorithm updates the profit of each object Π_u^v. For simplicity, let $\overline{\Pi}$ be the set of controllable flows. The profit of an individual object $p(\Pi_{u_j}^{v_i})$ is updated as $p(\Pi_{u_j}^{v_i}) = |\Pi_{u_j}^{v_i} - \overline{\Pi}|$. The algorithm will terminate until all switches have been checked. The detailed algorithm is described in Algorithm 2.

5.3 Performance Analysis

In the following, Q_G denotes the set of controllable flows by Algorithm 2 (for WEI-B). In the lth iteration of Algorithm 2, the controllable flow set is G'_l, and the incremental profit is denoted as X'_l. That is, $X'_l = \omega(G'_l \backslash \bigcup_{i=1}^{l-1} G'_i)$, where $\omega(\cdot)$ denotes its cardinality.

Lemma 3. *Algorithm 2 (for WEI-B) achieves a $(2+\epsilon)$ -approximation.*

Proof. Let ψ be the approximation ratio of the FPTAS algorithm for 0-1 knapsack. Consider an instant that Algorithm 2 has executed l-1 iterations. In the lth iteration, the algorithm chooses the switch v'_l. Assume that the optimal solution will select a flow set, denoted as O_l, from switch v'_l. If we choose O_l instead of G'_l in this iteration, the incremental profit becomes $\omega(O_l \backslash \bigcup_{i=1}^{l-1} G'_i)$, denoted as X''_l. Obviously, we have $\psi \cdot X'_l \geq X''_l = \omega(O_l \backslash \bigcup_{i=1}^{l-1} G'_i) \geq \omega(O_l \backslash Q_G)$. It follows

$$\begin{aligned} \psi \cdot \omega(Q_G) &= \sum_{l=1}^{n} \psi \cdot X'_l \geq \sum_{l=1}^{n} \omega(O_l \backslash Q_G) \\ &= \sum_{l=1}^{n} \omega(O_l \backslash Q_G) \geq \omega(\bigcup_{l=1}^{n} O_l \backslash Q_G) \\ &= \omega(OPT \backslash Q_G) \geq [\omega(OPT) - \omega(Q_G)]. \end{aligned} \tag{4}$$

Thus, we have

$$(1+\psi) \cdot \omega(Q_G) \geq \omega(OPT). \tag{5}$$

The FPTAS method achieves the $(1+\epsilon)$-approximation for 0-1 knapsack problem [42], where ϵ is an arbitrarily small value. Thus, by Eq. (5), the proposed algorithm achieves $(2+\epsilon)$-approximation for our problem. □

Lemma 4. *The time complexity of Algorithm 2 (for WEI-B) is $O(\frac{n^2 \cdot m^2}{\epsilon} + n^2 \cdot m)$, where n, m are the number of switches, the number of terminals in a network, respectively.*

Proof. There are at most n iterations in Algorithm 2, and each iteration consists of two main steps. In the first step, we need run at most n times FPTAS algorithm and the time complexity of each FPTAS algorithm is $O(\frac{m^2}{\epsilon})$ [42]. Thus, the time complexity for step 1 is $O(\frac{n \cdot m^2}{\epsilon})$. In the second step, we update the profit of each wildcard rule set on each switch, which takes $O(n \cdot m)$ time. As a result, the total time complexity of Algorithm 2 is $O(\frac{n^2 \cdot m^2}{\epsilon} + n^2 \cdot m)$, where n, m are the number of switches, the number of terminals in a network, respectively. □

6 Exact-Match Entry Installment

In this section, we describe step 2 of *HiFi*: exact-match entry installment, which is triggered by new packet arrival events. When a packet arrives at a switch and there is no matched entries, the switch will report the packet header to the controller. The controller will install exact-match entries to achieve specific application requirements, e.g., load balancing or throughput maximization. This section focuses on load balancing as a case study.

Note that, the installation of exact-match entries depends on the performance goal and the management policies, which are set by the system administrator and vary greatly in practice. Nevertheless, the fine-grained flow management architecture has an open design that can accommodate any implementation of exact-match entry installment module. For the purpose of completeness and numerical evaluation, we provide load balancing as a case study in this section with one implementation. We think its properties should not be viewed as limitation of our fine-grained flow management design because this specific implementation can be replaced with other implementations in practice.

6.1 Exact-Match Entry Installment for Load Balancing

This section studies the exact-match entry installment for load balancing (MT-LB) as a typical case. In Sections 4.2 and 6.2, the flow set is obtained through long-term traffic matrix prediction. In this section, when a packet arrives at a switch and there is no matched entry, the switch will report the packet header to the controller. Thus, instead of the long-term traffic observation in Sections 4.2 and 6.2, we care for the current flow set Γ, and the traffic size (or intensity) of each flow $\gamma \in \Gamma$ is denoted by $f(\gamma)$. We use two different notations to emphasize the difference in flow set.

This section studies the exact-match entry installment for load balancing (MT-LB) as a typical case. To obtain better network performance, we should dynamically update the flow routes so as to adapt to the traffic dynamics [43]. Thus, instead of the long-term traffic observation in Section 4.2, we care for the current flow set Γ, and the traffic size (or intensity) of each flow $\gamma \in \Gamma$ is denoted by $f(\gamma)$. With the system running, the flow set Γ will be updated.

We first introduce how to construct a feasible path set $\mathbb{P}_{v_1}^{v_2}$ from switch v_1 to switch v_2, which is determined by the management policies and performance objectives. If there are too many feasible paths that satisfy the management policies, we may include only a certain number of the best ones under a certain performance criterion, such as having the shortest number of hops or having the large capacities. Then, we explore a feasible path set $\mathbb{P}_\gamma$ for each flow γ. Under the proposed *HiFi* framework, when a flow γ arrives at the network, since some wildcard entries may be installed on some switches, flow γ may be directly forwarded to a switch, denoted by v_γ, in which there is no matching entry for this flow. The forwarding path from the source to switch v_γ is denoted by p_γ^w. Then, we derive a feasible path set $\mathbb{P}_\gamma$

as follows: for each path $p \in \mathbb{P}_{v_\gamma}^{e(\gamma)}$, where $e(\gamma)$ denotes the egress switch of flow γ, we construct a path p' by combining p_γ^w, p, and the link between the egress switch $e(\gamma)$ and the destination $d(\gamma)$. If this path has no loop, we add it to $\mathbb{P}_\gamma$.

Algorithm 3. Exact-Match Entry Installment for Load Balancing

```
Step 1: Solving the Relaxed MT-LB Problem
Relax Eq. (6) by replacing the fourth line of integer constraints with 0 ≤ y_γ^p ≤ 1
Driven the fractional solutions ŷ_γ^p, ∀γ ∈ Γ, p ∈ ℙ_γ
Step 2: Route Selection for Load Balancing
Obtain an integer solution ŷ_γ^p by rounding method
for Each flow γ ∈ Γ do
  for Each each path p ∈ ℙ_γ do
    if ŷ_γ^p = 1 then
      Route flow γ to path p
    end if
  end for
end for
```

The MT-LB problem will select one feasible path from $\mathbb{P}_\gamma$ for each flow γ to achieve load balancing. Let $c(e)$ and $l(e)$ denote the capacity of link e and the traffic load on link e, which is available to the controller by OSPF-TE [32]. The load-balancing factor λ is defined as $\lambda = \max\{\frac{l(e)}{c(e)},\ \forall e \in E\}$. We expect to minimize the load-balancing factor, i.e., $\min\ \lambda$.

We give the formulation of the MT-LB problem. Let an indicator variable $y_\gamma^p \in \{0,1\}$ denote whether flow γ will be routed on a path $p \in \mathbb{P}_\gamma$ or not. Let $I(\gamma,p,v)$ be a binary value for exact-match entry installment: if switch v has already installed the wildcard entry for destination $d(\gamma)$, and the next hop of this wildcard entry point to overlaps with the next hop of switch v on path p, then there is no need to install an exact-match entry on switch v, i.e., $I(\gamma,p,v)=0$; otherwise $I(\gamma,p,v)=1$. MT-LB solves the following problem:

$$\min\ \lambda$$

$$S.t. \begin{cases} \sum_{p\in\mathbb{P}_\gamma} y_\gamma^p = 1, & \forall \gamma \in \Gamma \\ \sum_{\gamma\in\Gamma}\sum_{p\in\mathbb{P}_\gamma: v\in p} y_\gamma^p \cdot I(\gamma,p,v) \le \delta(v), & \forall v \in V \\ \sum_{\gamma\in\Gamma}\sum_{p\in\mathbb{P}_\gamma: e\in p} y_\gamma^p f(\gamma) \le \lambda \cdot c(e), & \forall e \in E \\ y_\gamma^p \in \{0,1\}, & \forall p,\gamma \end{cases} . \tag{6}$$

The first set of equations requires that each flow $\gamma \in \Gamma$ will be forwarded through a single path from $\mathbb{P}_\gamma$. The second set of inequalities describes the flow-table size constraint on each switch v, where $\delta(v)$ is the number of residual flow entries on switch v, with $\delta(v) \le s(v)$. The third set of inequalities states that the traffic load on each link e should not exceed $\lambda \cdot c(e)$, where λ is the load-balancing factor (less than or equal to 1). The objective is to minimize the load-balancing factor λ.

6.2 Algorithm Design for MT-LB

Observing Eq. (6), there are standard rounding-based approximate methods for this problem. An example is relaxation and random rounding [44]. It relaxes Eq. (6) by replacing the fourth line of integer constraints with $0 \le y_\gamma^p \le 1$, and obtains a linear programming problem. We can solve it using the linear programming solver, and derive the optimal solution. Then, we round y_γ^p to zero or one probabilistically based on its fractional value. The algorithm for MT-LB is described in Algorithm 3. We use this method in the numerical evaluation of the proposed work.

In the practical scenarios, many flows may burst in the network, and the controller is unable to provide exact-match entries for each individual flow due to flow-table size constraint. To deal with this case and make our solution more practical, we will choose some switch pairs with less traffic amount, and deploy default paths for these flows. Meanwhile, the controller removes the exact-match rules for these flows so as to set aside some entries for accommodating potential arrival flows.

7 Performance Evaluation

To explore feasibility and efficiency of *HiFi*, we perform both experimental study and simulation evaluation. In this section, we first introduce the performance metrics and methodology. Then we give the results of experimental evaluation and simulation evaluation.

7.1 Performance Metrics and Methodology

In this section, we evaluate *HiFi* through small-scale testbed implementation and large-scale simulations, and compare it with the following existing approaches. (1) The first one is the RLJD algorithm [31], which installs exact-match entries on all switches along each forwarding path. To enhance the competitiveness of this algorithm, we modify the per-flow routing strategy in the final step by heuristically aggregate per-flow entries based on destination, while for each flow leaving at least one switch to keep the per-flow entry, in order to satisfy the "controllable" constraint. After this modification, RLJD can achieve better performance than the original one while compared with *HiFi*. (2) The second one is Presto [26], which is designed for hierarchical networks to achieve load balancing. Specifically, it installs exact-match entries on edge switches to control new arrival flows and installs wildcard entries on internal switches to relieve the load of core switches. For a fair comparison, we also extend this design to non-hierarchical network. Specifically, we install an exact-match entry only on the egress switch for each flow and install wildcard entries on other switches. Presto may encounter resource bottleneck at edge switches, which will be shown in the following simulations.

To compare the performance of three algorithms, we use the following performance metrics in our evaluation: (1) The number of Packet-in messages; (2) The number of controllable flows; (3) The number of required flow entries; (4) The control overhead (the communication traffic volume to/from the controller); (5) Flow setup delay; (6) Packet loss ratio; (7) The maximum throughput of the network; and (8) Load-balancing factor λ. When a flow arrives at a switch, and it does not match any existing entries on the flow table, the OpenFlow Agent of the switch will encapsulate the packet into a Packet-in message and send to the controller for requesting routing strategy. We measure the maximum number of encapsulated Packet-in messages on any switch during the simulation as the first metric. Note that, too many Packet-in messages will consume the CPU cycles,

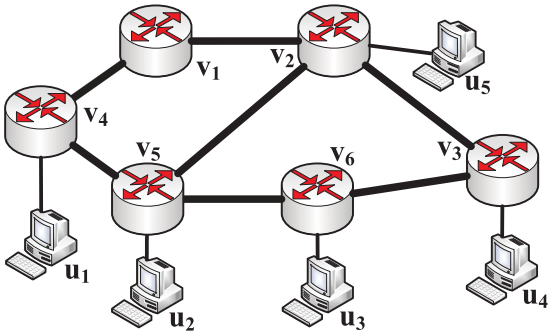


Fig. 4. Topology of the SDN Platform. We implement our proposed algorithm and other benchmarks on a small-scale testbed. Our testbed is mainly composed of three parts: a controller, six Open vSwitches and five virtual machines.

communication bandwidth and memory in both the switch and the controller, which may cause worse network performance (i.e., high packet loss ratio and high flow setup latency) [45]. Once receiving the Packet-in message of this flow, the controller controls this flow, then we obtain the second metric. The controller computes the route and sends Flow-mod commands to corresponding switches for entry installment. We measure the maximum number of required flow entries on any switch at any time and the maximum communication traffic volume to/from the controller during the simulation as the third and fourth metrics. Once the flow entries are installed, the flow can be forwarded to destination according to matched flow entries. We measure the maximum flow setup delay and packet loss ratio of all flows as the fifth and sixth metrics. Meanwhile, we measure the maximum throughput that the network can support and the traffic link $f(e)$ of each link e. Then, we compute the load-balancing factor $\lambda = \max\{f(e)/c(e), e \in E\}$.

The simulations are performed under two scenarios. The first scenario has no flow-table size (FTS) constraint, assuming that the switches have sufficient entries to handle all flows. This hypothetical scenario tests how well three algorithms perform when the FTS is sufficient. The second scenario has an FTS constraint and tests the performance of these algorithms when the FTS is limited.

7.2 Testbed Evaluation

7.2.1 Implementation On the Platform

Our testbed is built on a real topology obtained from the Internet Topology Zoo [46], called Epoch [47], which contains 6 nodes (switches) and 7 links. We randomly deploy several terminals on our small-scale testbed to simulate different flows, and the topology is illustrated in Fig. 4. Our SDN platform is mainly composed of three parts: a controller, 6 Open vSwitches (Version 2.5.3) [48] and 5 virtual machines (acting as terminals). Each Open vSwitch and its connected virtual machines are run on a server with a core i5-3470 processor and 8 GB of RAM. The link capacity is set as 200 Mbps for simplicity. We use the OpenDaylight Lithium-SR1 release [49] as the controller software running on a server with a core i7-8700k processor and 16GB of RAM.

We implement our tests with a set of synthetic and realistic workloads. Similar to previous works [24], [26] , our synthetic workloads include: (1) random flows, each terminal sends to several random terminals; (2) server flows, random terminals send the traffic to a number of designated terminals. These flows can simulate the traffic of mail servers and web servers; (3) associative flows, these flows simulate the communications between a terminal and a number of designated terminals, e.g., traffic between the finance department and the database. The authors of [29] have shown that less than 20 percent of the top-ranked flows may be responsible for more than 80 percent 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 1 Mbps.

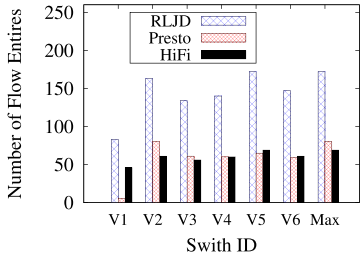


Fig. 5. No. of flow entries on each switch.

We use iperf3.3 [50] to simulate diverse kinds of flows, such as different packet size and traffic duration. First, for each server terminal, we start several iperf servers to monitor different ports. Then, we generate TCP/UDP flows with different packet size and traffic duration to simulate different applications. At last we obtain traffic behavior data (i.e., packet loss ratio, throughput, latency *etc.*) through iperf3.3 tool.

7.2.2 Testing Results

We run four sets of experiments on the SDN platform and execute each experiment 50 times and average the numerical results for accuracy. The first two sets of experiments are performed without flow-table size constraint, the default number of flows is set as 300. The last two sets of experiments are performed with flow-table size constraint and the default flow-table size is set as 100.

In the first experiment, we observe the number of required flow entries and the number of Packet-in messages on all switches. The testing results are shown in Figs. 5 and 6. Fig. 5 indicates that *HiFi*, Presto and RLJD need 69, 80 and 172 flow entries at most, respectively, which means that our proposed algorithm can reduce the maximum number of required flow entries by about 15 and 60 percent compared with Presto and RLJD, respectively. Fig. 6 shows that *HiFi*, Presto and RLJD generate 129, 149 and 231 Packet-in messages at most on any switch during the experiment, respectively. That is because our proposed system has pre-

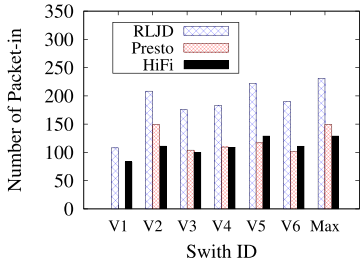


Fig. 6. No. of packet-in on each switch.

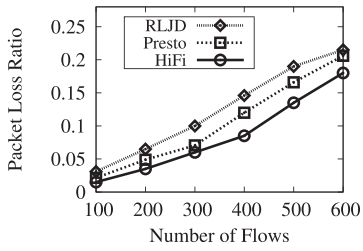


Fig. 7. Packet loss ratio versus no. of flows.

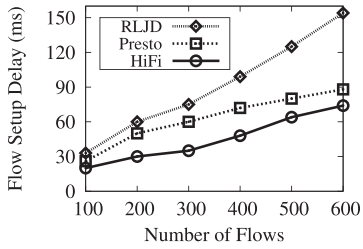


Fig. 8. Flow setup delay versus no. of flows.

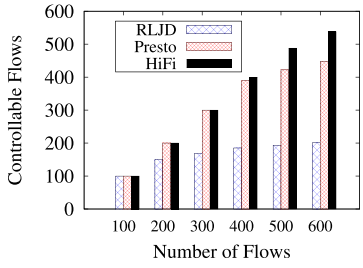


Fig. 9. No. of controllable flows versus no. of flows.

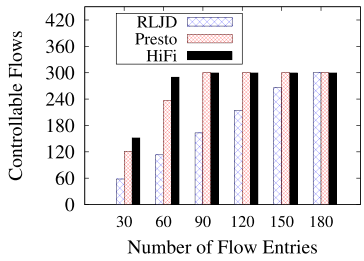


Fig. 10. No. of controllable flows versus no. of flow entries.

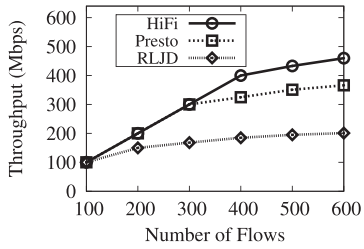


Fig. 11. Throughput versus no. of flows.

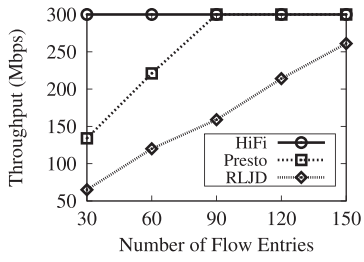


Fig. 12. Throughput versus no. of flows entries.

deployed some wildcard entries, which has reduced the interaction between data plane and control plane when flows arrive at switches. Note that, the performance gap between *HiFi* and Presto will be more obvious in large-scale simulations, which has been shown in Section 7.3. The work presented in [45] has illustrated that encapsulating Packet-in messages is time-consuming and cpu-consuming for low end CPU of switches and most commodity switches can only encapsulate Packet-in messages at the rate of 150 per second. The following experiments also indicate that too many Packet-in messages may cause high packet loss ratio and high latency.

In the second experiment, we observe the packet loss ratio and flow setup delay by changing the number of flows in the network. As shown in Figs. 7 and 8, when we generate 500 new flows simultaneously using iperf3.3 in the network, *HiFi* can reduce the maximum packet loss ratio of any flows by about 29 and 19 percent compared with RLJD and Presto, respectively. Meanwhile, *HiFi* only needs 74ms at most to setup a new flow while RLJD needs 154 ms and Presto needs 88ms to setup a new flow. Thus, the less interaction between data plane and control plane (i.e., less Packet-in messages and less Flow-mod commands/flow entries) of *HiFi* will make the lower packet loss ratio and lower flow setup delay as shown in Figs. 7 and 8.

The third set of experiments compares the number of controllable flows by varying the number of flows in the network and the number of available flow entries on the switches. As shown in Fig. 9, when there are 100 flows in the network, all three algorithms can control all flows with fine-grained management. That is because the flow-table size (i.e.,100) is sufficient to handle all flows (i.e.,100 flows). However, when there are more flows in the network, *HiFi* can control more flows than the other two algorithms under flow-table size constraint. That's because *HiFi* requires fewer entries per flow on average than RLJD, and distributes exact-match entries on all switches more evenly than Presto. Specifically, when there are 600 flows in the network, *HiFi* can control 538 flows while RLJD and Presto control only 201 and 448 flows, respectively. In other words, *HiFi* improves the number of controllable flows about 20 percent compared with Presto. As shown in Fig. 10, when there are 300 flows in the network, and we change the number of available flow entries, we observe that our proposed algorithm can control more flows when the number of available flow entries is limited (i.e., only 60 available entries on each switch).

The last set of experiments observes the maximum throughput in the network by changing the number of flows in the network and the number of available flow entries on the switches. The results are shown in Figs. 11 and 12. As shown in Fig. 11, we set the flow-table size as 100. Due to our proposed system uses fewer flow entries, *HiFi* can achieve higher network throughput compared with other algorithms with the increasing of flows. For example, when there are 500 flows in the network, *HiFi* improves the network throughput by about 122 and 24 percent compared with RLJD and Presto, respectively. Fig. 12 shows that *HiFi* can achieve better network throughput

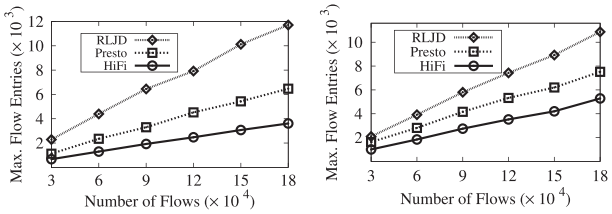


Fig. 13. Max. flow entries versus no. of flows without FTS constraint. *Left plot*: (a) Fat-Tree; *right plot*: (b) Campus.

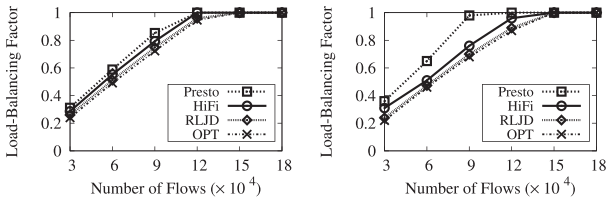


Fig. 14. Load-balancing factor versus no. of flows without FTS constraint. *Left plot*: (a) Fat-Tree; *right plot*: (b) Campus.

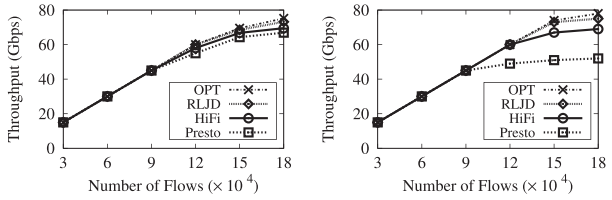


Fig. 15. Throughput versus no. of flows without FTS constraint. *Left plot*: (a) Fat-Tree; *right plot*: (b) Campus.

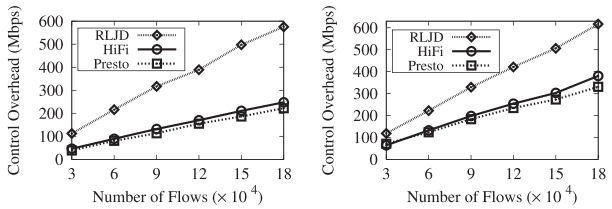


Fig. 16. Control overhead versus no. of flows without FTS constraint. *Left plot*: (a) Fat-Tree; *right plot*: (b) Campus.

when the flow entries are limited. That is because *HiFi* can select partial flows for fine-grained management and others for coarse-grained management so as to serve all flows with flow-table size constraint as illustrated in Section 6.2.

From these testing results, we state that *HiFi* can achieve better traffic behavior (i.e., lower packet loss ratio and flow setup delay) with less resource usage (i.e., less flow entry cost compared with the other two algorithms) while keeping fine-grained flow management. Moreover, our proposed algorithm can control more flows than the other two algorithms under the same FTS constraint.

7.3 Simulation Evaluation

7.3.1 Simulation Settings

We select two practical topologies. The first topology, denoted as (a), is a hierarchical Fat-Tree topology [27]. This topology has been widely used in many datacenter networks. It contains 16 core switches, 32 aggregation switches, 32 edge switches and 128 servers. The second one is a non-hierarchical campus network from [51], denoted as (b), containing 100 switches and 200 terminals. For both topologies, we execute each simulation 10 times and average the results. The link capacity is set as 1 Gbps for simplicity. The flow size is drawn from random flows, server flows and associative flows discussed in Section 7.2.1.

7.3.2 Performance Comparison Without FTS Constraint

The first set of simulations compares *HiFi*, RLJD and Presto in the scenario without FTS constraint. The results are shown in Figs. 13, 14, 15, and 16. The horizontal axis of Fig. 13 is the number of flows in the network, ranging from 3×10^4 to 18×10^4. The maximum number of required entries increases for all three solutions. In comparison, *HiFi* uses much fewer entries than RLJD and Presto. For example, when there are 12×10^4 flows in the Fat-Tree network, our solution requires 2,476 entries at most while Presto and RLJD require 4,520 and 7,920 entries, respectively. That means *HiFi* reduces the maximum number of required entries by about 45 and 69 percent compared with Presto and RLJD, respectively.

We plot the load-balancing factor of these algorithms in Fig. 14. To observe the performance of different applications (i.e., load-balancing or throughput maximization), we add another benchmark, denoted as OPT. We note that OPT may denote different solutions for various applications. For load balancing, OPT can be derived by solving relaxed MT-LB using a linear program solver. This figure shows that our solution only increases load-balancing factor by about 3 and 5 percent compared with RLJD and OPT, respectively. Besides, *HiFi* achieves better load-balancing factor than Presto, especially in a non-hierarchical campus network. For example, when there are 9×10^4 flows, *HiFi* reduces the load-balancing factor by about 34 percent compared with Presto. It means that Presto cannot achieve satisfactory performance in a non-hierarchical network while *HiFi* can achieve nearly optimal performance in both non-hierarchical and hierarchic networks. For network throughput, the OPT solution will gradually increase the traffic intensity until the link capacity is fully utilized. Fig. 15 shows that our proposed algorithm achieves better network throughout than Presto. For example, when there are 15×10^4 flows in the campus network, *HiFi* increases network throughput by about 38 percent compared with Presto and achieves similar throughput compared with PLJD and OPT.

Fig. 16 shows that *HiFi* can achieve similar control communication overhead compared with Presto and achieve much smaller control communication overhead compared with RLJD. As the number of flows increases, RLJD installs more flow entries than *HiFi* and Presto, which results in higher control overhead than two other solutions. For example, when there are 12×10^4 flows in the campus network, the control overhead of RLJD, Presto and *HiFi* will reach 421 Mbps, 234 Mbps and 253 Mbps, respectively.

7.3.3 Performance Comparison With FTS Constraint

The second set of simulations compares *HiFi*, RLJD and Presto in the scenario with FTS constraint, where the FTS constraint is set as 4,000 on each switch by default [29]. We first compare the number of controllable flows by changing the number of flows in the network. The results are shown in Fig. 17. We claim that our solution can control more flows

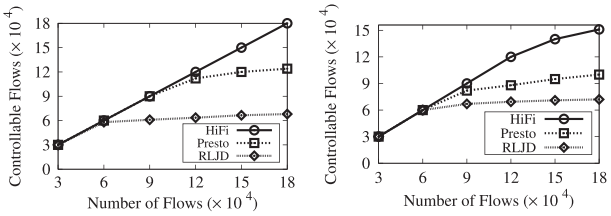


Fig. 17. No. of controllable flows versus no. of flows with FTS constraint. *Left plot*: (a) Fat-Tree; *right plot*: (b) Campus.

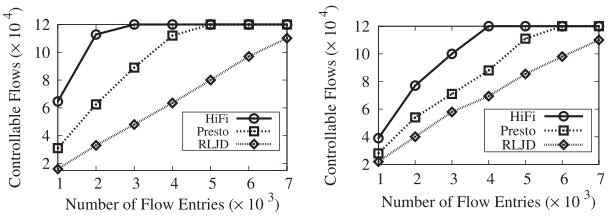


Fig. 18. No. of controllable flows versus no. of flow entries with FTS constraint. *Left plot*: (a) Fat-Tree; *right plot*: (b) Campus.

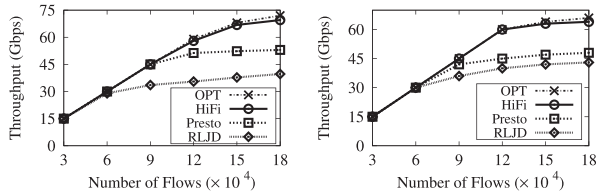


Fig. 19. Throughput versus no. of flows with FTS constraint. *Left plot*: (a) Fat-Tree; *right plot*: (b) Campus.

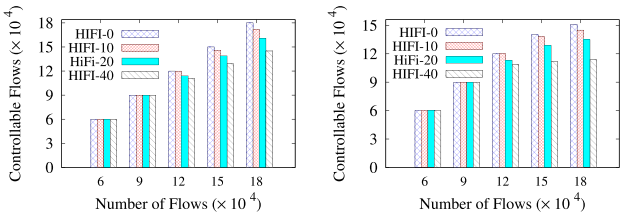


Fig. 20. No. of controllable flows versus no. of flows with FTS constraint. *Left plot*: (a) Fat-Tree; *right plot*: (b) Campus.

than the other two algorithms. For example, as shown in the right plot of Fig. 17, when there are 15×10^4 flows, our proposed algorithm can control about 14×10^4 flows while the other two algorithms can only control 7.1×10^4 and 9.5×10^4 flows, respectively. It means that *HiFi* increases the number of controllable flows by about 48 and 97 percent compared with Presto and RLJD, respectively. Besides, when the number of flows in the network is constant (i.e., 12×10^4), we observe the number of controllable flows by varying the FTS constraint. As shown in Fig. 18, we can also claim that our proposed algorithm is able to control more flows than the other two algorithms under the same FTS constraint. That is because (1) *HiFi* requires fewer flow entries than RLJD per flow on average; and (2) *HiFi* can distribute exact-match entries on all switches evenly while Presto only installs exact-match entries on ingress switches.

Fig. 19 shows that when the number of flow entries is constant (i.e., 4,000), the network performance (e.g., throughput) of our algorithm is much better than that of Presto and RLJD. For example, when there are 15×10^4 flows in the Fat-Tree network, our proposed algorithm can improve throughput by about 28 and 77 percent compared with Presto and RLJD, respectively. Moreover, *HiFi* can achieve similar throughput compared with OPT, which means high effectiveness of our proposed approximation algorithms.

From these simulation results, we can draw some conclusions. First, from Figs. 13, 14, 15, and 16, when there is no FTS constraint, *HiFi* can reduce the number of maximum flow entries by about 45 and 69 percent compared with Presto and RLJD, respectively. Accordingly, *HiFi* decreases the control overhead by about 40 percent compared with RLJD. Besides, our proposed algorithm can achieve similar performance (e.g., load-balancing factor, throughput) compared with RLJD and OPT. Moreover, *HiFi* can improve network performance by about 38 percent compare with Presto while using a similar (or less) number of flow entries in a non-hierarchical network. Second, from Figs. 17, 18, and 19, when the FTS is limited, our algorithm can improve the number of controllable flows by about 48 and 97 percent compared with Presto and RLJD, respectively.

7.3.4 The Impact of Estimation Error

As shown in the end of Section 3.4 (workflow section), when a timer (e.g., 10min) expires, HiFi needs to estimate the traffic synopsis (i.e., the number of flows between each source-destination pair) to determine how to install wildcard entries. Traffic prediction is an important research issue in the field of routing optimization and many works have devoted to improve the prediction accuracy [37], [39], [52], [53], [54], [55]. Due to the plenty prior works on traffic prediction, the prediction error has been greatly reduced. For example, the recent work [52] has evaluated their prediction model on seven traffic datasets (including three real-world traffic datasets). The results show that the estimation errors for all seven traffic datasets are less than 18 percent. In this section, we evaluate the impact of traffic estimation error on network performance. Though these solutions may have some estimation errors, we find that even with these traffic estimation errors, our proposed algorithm can still achieve a satisfactory performance, which has been validated through the extensive simulations as follows.

We have added two sets of simulations to test the impact of traffic estimation error on network performance as shown in Figs. 20 and 21. Specifically, we assume that the actual number of flows from source u_1 to destination u_2 is $\overline{N}(u_1, u_2)$. We simulate the case of X% estimation error as the predicted number of flows $\widetilde{N}(u_1, u_2)$ from u_1 to u_2 is equal to $(1 - X\%) \cdot \overline{N}(u_1, u_2)$ or $(1 + X\%) \cdot \overline{N}(u_1, u_2)$. We have compared the network performance under different prediction errors (i.e., 0, 10, 20 and 40 percent) and the FTS constraint is by default set as 4,000 on each switch [29]. For simplicity, we use HIFI-0, HIFI-10, HIFI-20 and HIFI-40 to denote the estimation errors of 0, 10, 20 and 40 percent, respectively, in the simulation.

The first set of simulations shows the number of controllable flows by changing the number of flows in the network. We claim that the number of controllable flows is only slightly reduced when the estimation error is less than 20 percent (or even 40 percent). For example, as shown in the left plot of Fig. 20, when there are 15×10^4 flows, HIFI-0 (i.e., estimation error = 0 percent) can control about 15×10^4 flows while the results of other estimation errors can control 14.6×10^4, 13.9

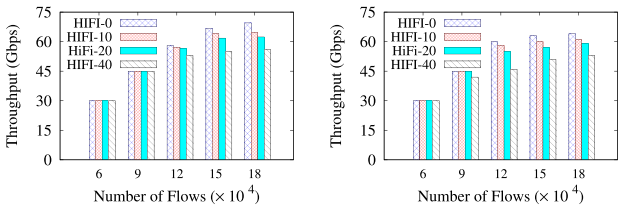


Fig. 21. Throughput versus no. of flows with FTS constraint. *Left plot*: (a) Fat-Tree; *right plot*: (b) Campus.

$\times 10^4$ and 12.9 $\times 10^4$ flows, respectively. It means that the result of 10 percent estimation error only reduces the number of controllable flows by about 3 percent compared with HIFI-0.

The second set of simulations presents the network throughput by changing the number of flows in the network. The results are shown in Fig. 21. We can conclude that estimation error only has little influence on network throughput. For example, when there are 18×10^4 flows in campus network, HIFI-0 can achieve nearly 65 Gbps throughput while the results of other estimation errors can achieve throughput nearly 62 Gbps, 59 Gbps, and 53 Gbps, respectively. It means that the result of HIFI-10 only reduces the throughput by about 4 percent compared with HIFI-0.

From the above simulations, we can conclude that the influence of estimation accuracy on network performance is acceptable in most situations. One may say that we may cannot lose control of any required flows in some scenarios (e.g., bank network system as shown in Section 4.5). In this situation, we assume that each required source-destination pair (i.e., related with 10 key servers) has at least one flow. In this way, the first set of inequalities of Eq. (1) can guarantee fine-grained management for all flows of required source-destination pairs.

8 Conclusion

In this paper, we have designed *HiFi*, which provides fine-grained flow management with a limited number of flow entries using a novel hybrid (wildcard and exact-match) rule placement scheme. Several algorithms with bounded approximation factors have been designed for wildcard entry installment and exact-match entry installment, respectively. We have implemented *HiFi* on our commodity SDN platform, and simulation results have shown the high efficiency and effectiveness of *HiFi*.

Acknowledgments

This article was supported in part by the National Science Foundation of China (NSFC) under Grants 61822210, 61936015, and U1709217; and in part by Anhui Initiative in Quantum Information Technologies under Grant AHY150300. Some preliminary results of this article were published in the Proceedings of IEEE INFOCOM 2020 [1].

References

[1] G. Zhao, H. Xu, J. Fan, L. Huang, and C. Qiao, "HiFi: Hybrid rule placement for fine-grained flow management in SDNs," in *Proc. IEEE Conf. Comput. Commun.*, 2020, pp. 1–10.

[2] N. Gude *et al.*, "NOX: Towards an operating system for networks," *ACM Comput. Commun. Rev.*, vol. 38, no. 3, pp. 105–110, 2008.

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

[4] J. Mai, C.-N. Chuah, A. Sridharan, T. Ye, and H. Zang, "Is sampled data sufficient for anomaly detection?" in *Proc. 6th ACM SIGCOMM Conf. Internet Meas.*, 2006, pp. 165–176.

[5] Y. Zhang, "An adaptive flow counting method for anomaly detection in SDN," in *Proc. 9th ACM Conf. Emerg. Netw. Experiments Technol.*, 2013, pp. 25–30.

[6] S. Agarwal, M. Kodialam, and T. Lakshman, "Traffic engineering in software defined networks," in *Proc. IEEE Conf. Comput. Commun.*, 2013, pp. 2211–2219.

[7] S. Jain *et al.*, "B4: Experience with a globally-deployed software defined WAN," in *Proc. ACM SIGCOMM Conf. SIGCOMM*, 2013, pp. 3–14.

[8] T. Pan, X. Guo, C. Zhang, J. Jiang, H. Wu, and B. Liuy, "Tracking millions of flows in high speed networks for application identification," in *Proc. IEEE Conf. Comput. Commun.*, 2012, pp. 1647–1655.

[9] A. Craig, B. Nandy, I. Lambadaris, and P. Ashwood-Smith, "Load balancing for multicast traffic in SDN using real-time link cost modification," in *Proc. IEEE Int. Conf. Commun.*, 2015, pp. 5789–5795.

[10] H. Xu, X.-Y. Li, L. Huang, H. Deng, H. Huang, and H. Wang, "Incremental deployment and throughput maximization routing for a hybrid SDN," *IEEE/ACM Trans. Netw.*, vol. 25, no. 3, pp. 1861–1875, Jun. 2017.

[11] K. Bu, X. Wen, B. Yang, Y. Chen, L. E. Li, and X. Chen, "Is every flow on the right track?: Inspect SDN forwarding with rulescope," in *Proc. 35th Annu. IEEE Int. Conf. Comput. Commun.*, 2016, pp. 1–9.

[12] J. P. Sheu and Y. C. Chuo, "Wildcard rules caching and cache replacement algorithms in software-defined networking," *IEEE Trans. Netw. Service Manage.*, vol. 13, no. 1, pp. 19–29, Mar. 2016.

[13] S. Bera, S. Misra, and A. Jamalipour, "FlowStat: Adaptive flow-rule placement for per-flow statistics in SDN," *IEEE J. Sel. Areas Commun.*, vol. 37, no. 3, pp. 530–539, Mar. 2019.

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

[15] B. Agrawal and T. Sherwood, "Modeling TCAM power for next generation network devices," in *Proc. IEEE Int. Symp. Perform. Anal. Syst. Softw.*, 2006, pp. 120–129.

[16] M. Rui, H. Zeng, C. Kim, J. Lee, and M. Yu, "SilkRoad: Making stateful layer-4 load balancing fast and cheap using switching ASICs," in *Proc. Conf. ACM Special Interest Group Data Commun.*, 2017, pp. 15–28.

[17] SDN programmable network switch, 2020. [Online]. Available: https://noviflow.com/noviswitch/

[18] G. P. Katsikas, T. Barbette, D. Kostic, R. Steinert, and G. Q. M. Jr, "Metron: NFV service chains at the true speed of the underlying hardware," in *Proc. 15th USENIX Symp. Netw. Syst. Des. Implementation*, 2018, pp. 171–186.

[19] Hewlett Packard. HPE FlexNetwork 5130 EI switch series, 2020. [Online]. Available: https://h50146.www5.hpe.com/products/networking/datasheet/HP_5130EI_Swi tch_Series_J.pdf.

[20] C. Metter, S. Gebert, S. Lange, T. Zinner, P. Trangia, and M. Jarschel, "Investigating the impact of network topology on the processing times of SDN controllers," in *Proc. IFIP/IEEE Int. Symp. Integr. Netw. Manage.*, 2015, pp. 1214–1219.

[21] M. Rifai *et al.*, "Too many SDN rules? Compress them with minnie," in *Proc. IEEE Global Commun. Conf.*, 2015, pp. 1–7.

[22] 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 25th Int. Conf. Netw. Protocols*, 2017, pp. 1–10.

[23] X.-N. Nguyen, D. Saucez, C. Barakat, and T. Turletti, "OFFICER: A general optimization framework for OpenFlow rule allocation and endpoint policy enforcement," in *Proc. IEEE Conf. Comput. Commun.*, 2015, pp. 478–486.

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

[25] B. Claise *et al.*, "Cisco systems NetFlow services export version 9," *J. Int. Eng. Task Force*, 2004.

[26] K. He, E. Rozner, K. Agarwal, W. Felter, J. Carter, and A. Akella, "Presto: Edge-based load balancing for fast datacenter networks," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 45, pp. 465–478, 2015.

[27] 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.

[28] J. H. Ahn, N. Binkert, A. Davis, M. McLaren, and R. S. Schreiber, "HyperX: Topology, routing, and packaging of efficient large-scale networks," in *Proc. Conf. High Perform. Comput. Netw. Storage Anal.*, 2009, Art. no. 41.

[29] 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 Comput. Commun. Rev.*, vol. 41, no. 4, pp. 254–265, 2011.
[30] H. Xu, Z. Yu, X.-Y. Li, C. Qian, and L. Huang, "Real-time update with joint optimization of route selection and update scheduling for SDNs," in *Proc. IEEE 24th Int. Conf. Netw. Protocols*, 2016, pp. 1–10.
[31] 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 Conf. Comput. Commun.*, 2014, pp. 1734–1742.
[32] D. Katz, K. Kompella, and D. Yeung, "Traffic engineering (TE) extensions to OSPF version 2," *Ietf Rfc*, vol. 170, no. 4, p. 1, 2003.
[33] J. Liu, Y. Lai, and S. Zhang, "FL-GUARD: A detection and defense system for DDoS attack in SDN," in *Proc. Int. Conf. Cryptogr. Secur. Privacy*, 2017, pp. 107–111.
[34] L. Luo, H. Yu, and S. Luo, "Scalable fine-grained path control in software defined networks," 2016, *arXiv:1611.09011*.
[35] T. Zhang, L. Linguaglossa, M. Gallo, P. Giaccone, and D. Rossi, "FlowMon-DPDK: Parsimonious per-flow software monitoring at line rate," in *Proc. Netw. Traffic Meas. Anal. Conf.*, 2018, pp. 1–8.
[36] A. Gember, P. Prabhu, Z. Ghadiyali, and A. Akella, "Toward software-defined middlebox networking," in *Proc. 11th ACM Workshop Hot Topics Netw.*, 2012, pp. 7–12.
[37] A. Azzouni and G. Pujolle, "NeuTM: A neural network-based framework for traffic matrix prediction in SDN," in *Proc. IEEE/IFIP Netw. Operations Manage. Symp.*, 2018, pp. 1–5.
[38] 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 Conf. Comput. Commun.*, 2017, pp. 1–9.
[39] P. Cortez, M. Rio, M. Rocha, and P. Sousa, "Internet traffic forecasting using neural networks," in *Proc. IEEE Int. Joint Conf. Neural Netw. Proc.*, 2006, pp. 2635–2642.
[40] Z. A. Qazi, C.-C. Tu, L. Chiang, R. Miao, V. Sekar, and M. Yu, "Simple-fying middlebox policy enforcement using SDN," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 43, no. 4, pp. 27–38, 2013.
[41] G. P. Ingargiola and J. F. Korsh, "Reduction algorithm for zero-one single knapsack problems," *Manage. Sci.*, vol. 20, no. 4-part-i, pp. 460–463, 1973.
[42] M. Bansal and V. Venkaiah, "Improved fully polynomial time approximation scheme for the 0-1 multiple-choice knapsack problem," Int. Inst. Inform. Technol., Tech. Rep., 2004.
[43] X. Jin *et al.*, "Dynamic scheduling of network updates," in *Proc. ACM SIGCOMM Conf. SIGCOMM*, 2014, pp. 539–550.
[44] T. Friedrich and T. Sauerwald, "Near-perfect load balancing by randomized rounding," in *Proc. 41st Annu. ACM Symp. Theory Comput.*, 2009, pp. 121–130.
[45] A. Wang, Y. Guo, F. Hao, T. Lakshman, and S. Chen, "Scotch: Elastically scaling up SDN control-plane using vSwitch based overlay," in *Proc. 10th ACM Int. Conf. Emerg. Netw. Experiments Technol.*, 2014, pp. 403–414.
[46] The internet topology zoo, Accessed: Jul. 2020. [Online]. Available: http://www.topology-zoo.org/
[47] The epoch topology, Accessed: Jul. 2020. [Online]. Available: http://www.topology-zoo.org/maps/Epoch.jpg
[48] Open vSwitch: Open virtual switch, Accessed: Apr. 2020. [Online]. Available: http://openvswitch.org/
[49] Linux foundation collaborative project, Accessed: Apr. 2020. [Online]. Available: http://opendaylight.org/
[50] Iperf3.3, Accessed: Apr. 2020. [Online]. Available: http://software.es.net/iperf/news.html#iperf-3–3-released
[51] Simulating network topologies, Accessed: Apr. 2020. [Online]. Available: http://www.ecse.monash.edu.au/twiki/bin/view/InFocus/LargePacket-switch ingNetworkTopologies
[52] A. Saha, N. Ganguly, S. Chakraborty, and A. De, "Learning network traffic dynamics using temporal point process," in *Proc. IEEE Conf. Comput. Commun.*, 2019, pp. 1927–1935.
[53] A. Bayati, K. K. Nguyen, and M. Cheriet, "Multiple-step-ahead traffic prediction in high-speed networks," *IEEE Commun. Lett.*, vol. 22, no. 12, pp. 2447–2450, Dec. 2018.
[54] Y. Li, H. Liu, W. Yang, D. Hu, X. Wang, and W. Xu, "Predicting inter-data-center network traffic using elephant flow and sublink information," *IEEE Trans. Netw. Service Manage.*, vol. 13, no. 4, pp. 782–792, Dec. 2016.
[55] W. Hao, C. Li, C. Kai, Z. Li, and Y. Geng, "FLOWPROPHET: Generic and accurate traffic prediction for data-parallel cluster computing," in *Proc. IEEE 35th Int. Conf. Distrib. Comput. Syst.*, 2015, pp. 349–358.

Gongming Zhao received the PhD degree in computer software and theory from the University of Science and Technology of China, China, in 2020. He is currently an associate professor in the University of Science and Technology of China, China. His current research interests include software-defined networks and cloud computing.

Hongli Xu (Member, IEEE) received the BS degree in computer science from the University of Science and Technology of China, China, in 2002, and the PhD degree in computer software and theory from the University of Science and Technology of China, China, in 2007. He is a professor with the School of Computer Science and Technology, University of Science and Technology of China (USTC), China. He was awarded the Outstanding Youth Science Foundation of NSFC, in 2018. He has won the Best Paper Award or the Best Paper Candidate in several famous conferences. He has published more than 100 papers in famous journals and conferences, including the *IEEE/ACM Transactions on Networking*, *IEEE Transactions on Mobile Computing*, *IEEE Transactions on Parallel and Distributed Systems*, INFOCOM, ICNP, etc. He has also held more than 30 patents. His main research interest is software defined networks, edge computing, and Internet of Thing.

Jingyuan Fan received the BEng and MS degrees from Fudan University, China, and the University of California, Los Angeles, Los Angeles, California, in 2012 and 2014, respectively, and the PhD degree in computer science from the State University of New York at Buffalo, Buffalo, New York, in 2019. His research interests lie in the field of computer networks and distributed systems.

Liusheng Huang (Member, IEEE) received the MS degree in computer science from the University of Science and Technology of China, China, in 1988. He is currently a senior professor and PhD supervisor of the School of Computer Science and Technology, University of Science and Technology of China, China. He has published six books and more than 300 journal/conference papers. His research interests are in the areas of Internet of Thing, vehicular ad hoc network, information security, and distributed computing.

Chunming Qiao (Fellow, IEEE) is currently a SUNY distinguished professor and also the chair of the Computer Science and Engineering Department, University at Buffalo, Buffalo, New York. He has published extensively with an H-index of more than 69 (according to Google Scholar). He holds seven U.S. patents, and served as a consultant for several IT and telecommunications companies since 2000. His current focus is on connected and autonomous vehicles. He has contributions to optical and wireless network architectures and protocols. Two of his papers have received the best paper awards from the IEEE and joint ACM/IEEE venues. His research has been funded by a dozen major IT and telecommunications companies, and more than a dozen NSF grants.