Mobility-Aware Proactive Flow Setup in Software-Defined Mobile Edge Networks

Yue Zeng, Baoliu Ye, *Member, IEEE*, Bin Tang, *Member, IEEE*, Sanglu Lu, *Member, IEEE*, Feng Xu, *Member, IEEE*, Song Guo, *Fellow, IEEE*, and Zhihao Qu, *Member, IEEE*

***Abstract*— The software-defined network (SDN) enabled mobile edge network greatly facilitates network resource management and promotes many emerging applications. However, user mobility may cause the SDN controller to set flow rules frequently, introduce additional flow setup latency, cause delay jitter, and undermine latency-sensitive services. Proactive flow setup is an effective way to eliminate flow setup latency, but existing work fails to maximize the flow setup hit ratio, a metric for evaluating the quality of proactive flow setup decisions, which is critical for latency-sensitive services. In this paper, we study how to proactively set flow rules to maximize the flow setup hit ratio under limited available network resources to eliminate the flow setup latency as much as possible. Then, we formalize the proactive flow setup problem as two integer linear programming problems under two typical routing strategies, default routing and dynamic routing. Both problems are proved to be NP-hard. To tackle these two problems, we propose a linear programming-based polynomial-time approximation algorithm for the default routing case and a greedy-based heuristic algorithm for the dynamic routing case. Extensive trace-driven experimental and simulation results verify that our algorithms can improve the flow setup hit ratio by up to 30.99% compared to existing solutions.**

***Index Terms*— Software-defined networking, mobile edge computing, 5G, approximation algorithm.**

I. Introduction

RECENTLY, many emerging online applications require the network to support ultra-low latency communications, such as augmented reality (AR), virtual reality (VR), and autonomous driving [1]. According to reports [2], [3], [4], autonomous driving requires no more than 3 ms network latency, while AR, VR, and industrial automation call for a network latency of less than 1 ms. To facilitate these promising applications, 5G is envisioned to provide users with ultra-low latency services [5].

Mobile edge computing, as a key enabling technology for 5G, pushes computing resources to the edge close to the data source, greatly shortening network latency [6], [7]. Online applications can be deployed on the edge cloud to provide users with ultra-low latency services. As another key enabling technology of 5G, SDN physically separates the network control plane from the data plane, greatly simplifying network management [8], [9]. The control plane has a global perspective over the entire network, which is convenient to allocate dedicated network resources for each service flow and ensure their isolation.

In software-defined mobile edge networks (SDMENs), as shown in Fig. 1, user mobility is common, especially for applications such as autonomous driving and in-car entertainment (AR/VR). When a user moves beyond the coverage of the currently visited small cell, it switches to a new small cell. This is exacerbated in 5G, due to the adoption of millimeter wave technology that shrinks the coverage of small cells [10], [11]. When a user accesses a new small cell, its traffic needs to wait for the SDN controller to reallocate bandwidth resources by setting flow rules, which introduces additional flow setup latency [12], [13]. This causes delay jitter, deteriorates latency-sensitive services, and worsens the user experience.

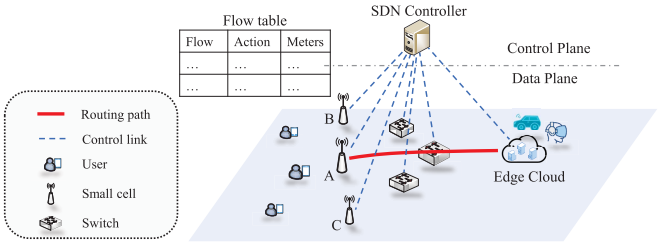


Fig. 1. A case of SDMENs. In the network, each network device maintains a flow table, which stores flow rules for instructing traffic routing and allocating bandwidth resources. The control plane manages the traffic in the data plane by setting up flow rules. When a user service flow arrives, the controller immediately plans a routing path and allocates corresponding network resources for it by setting up flow rules.

Manuscript received 28 March 2022; revised 28 September 2022 and 1 December 2022; accepted 11 January 2023. Date of publication 20 January 2023; date of current version 17 March 2023. This work was supported in part by the National Natural Science Foundation of China (Grant Nos. 61832005, 62172204, 62102131, and 61872171), in part by the Natural Science Foundation of Jiangsu Province, China (Grant Nos. BK20190058, BE2020001-3 and BK20210361), and in part by the Collaborative Innovation Center of Novel Software Technology and Industrialization. The associate editor coordinating the review of this article and approving it for publication was D. Marabissi. *(Corresponding author: Baoliu Ye.)*

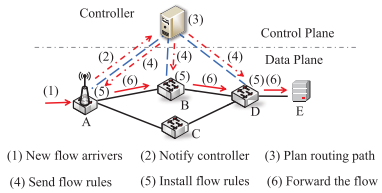
Yue Zeng, Baoliu Ye, and Sanglu Lu are with the National Key Laboratory for Novel Software Technology, Nanjing University, Nanjing 210023, China (e-mail: zengyue@smail.nju.edu.cn; yebl@nju.edu.cn; sanglu@nju.edu.cn).

Bin Tang, Feng Xu, and Zhihao Qu are with the School of Computer and Information, Hohai University, Nanjing 211100, China (e-mail: cstb@hhu.edu.cn; xufeng@hhu.edu.cn; quzhihao@hhu.edu.cn).

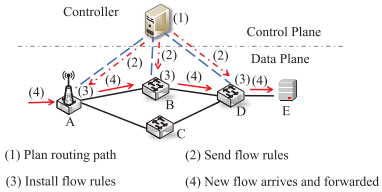
Song Guo is with the Department of Computing, The HongKong Polytechnic University, Hong Kong (e-mail: song.guo@polyu.edu.hk).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TCOMM.2023.3238396.

Digital Object Identifier 10.1109/TCOMM.2023.3238396



(a) Reactive flow setup process.



(b) Proactive flow setup process.

Fig. 2. Two flow setup processes in SDN. In the reactive flow setup process, when a new flow arrives, the small cell (as a switch) notifies the controller. Then, the controller plans a routing path for the flow and sends corresponding flow rules to the switches on this path. Finally, the flow can be forwarded after switches installed corresponding flow rules. In this case, the flow needs to wait for the flow setup process, which introduces additional flow setup latency. In the proactive flow setup process, before a flow arrives, the controller proactively plans a routing path for the flow and sends corresponding flow rules to the switches on the routing path. Then, the switches install these flow rules. In this case, the flow can be forwarded directly after it arrives, without introducing additional flow setup latency.

In SDN, the controller can set up flow rules in two ways, such as reactive and proactive, as shown in Fig. 2. For the former, the SDN controller reactively sets up flow rules for newly arrived flow requests, introducing additional flow setup latency. There has been a series of studies focused on how to shorten the flow setup latency, such as multi-controller load balancing [12], [14] and fine-grained flow management [15]. These efforts shorten the flow setup latency and facilitate traditional applications, but the inevitable flow setup latency may cause delay jitter, degrading latency-sensitive applications. The SDN controller can also set up flow rules in a proactive manner, which is especially important for mobile scenarios. In this way, the controller proactively sets up flow rules for each user on its neighboring small cells. Once a user switches to a neighbor small cell where the flow rules have been set, its flow can be forwarded directly without introducing additional flow setup latency. Then, several works have studied this topic in mobile scenarios [16], [17], [18], [19]. They proactively set up flow rules for each user's service flow based on their mobility transition probability, aiming to minimize TCAM[1] occupancy, controller overhead, or save network energy consumption, which greatly improves traditional applications. However, their objective is to maximize resource efficiency rather than the flow setup hit ratio (FSHR[2]), which is critical for latency-sensitive services.

[1]Ternary Content Addressable Memory (TCAM) is a limited and precious network resource on the switch, used to store flow rules and support high-speed flow rule matching [20], [21].

[2]FSHR is defined as the hit probability of flow rules set up for a user's flow request, which is formalized in Eq. (1). It is a metric for evaluating the quality of the proactive flow setup decision. A higher FSHR value indicates a higher probability that the flow setup latency is eliminated, and less delay jitter is introduced to degrade latency-sensitive services.

In this paper, we study how to proactively set up flow rules to obtain a high FSHR under limited available TCAM and bandwidth resources to improve latency-sensitive services, which is practical and crucial. Then, we formalize the proactive flow setup problem as two integer linear programs under two typical routing strategies, default routing and dynamic routing.[3] Both problems are proven to be NP-hard. To tackle the default routing case, we propose an approximation algorithm based on linear programming (LP) relaxation, a polynomial-time approximation scheme (PTAS[4]) in mid-scale networks. To deal with the dynamic routing case, we propose a greedy-based heuristic algorithm, a 2-approximation algorithm in a special case. Finally, we use extensive trace-driven experimental and simulation results to verify the superiority of our algorithms over existing solutions.

The main contributions of this paper can be summarized as follows.

- To the best of our knowledge, we are the first to study how to proactively set up flow rules to maximize the FSHR under limited available TCAM and bandwidth resources. Then, we formally constructed this problem as proactive flow setup problems with default routing and dynamic routing strategies, aiming to maximize the total FSHR, and they are proved to be NP-hard.
- To tackle the above two problems, we propose a polynomial-time approximate algorithm for the default routing case and an efficient heuristic algorithm for the dynamic routing case. Moreover, the heuristic algorithm proved to be a 2-approximation algorithm in a special case.
- Extensive trace-driven experimental and simulation results show that the performance of our algorithms is close to the optimal solution. Compared with existing solutions, our algorithm can significantly increase the FSHR and promote latency-sensitive services.

The rest of the paper is organized as follows. Section II briefly introduces the related work. Section III provides our motivation. Section IV formalizes the problem we studied and proves its NP-hardness. An approximate algorithm and heuristic algorithm are proposed in Sections V and VI, and are evaluated in Section VII. Section VIII concludes this paper.

II. Related Work

This section divides the related work of flow rule management into two categories, such as reactive and proactive.

A. Flow Rule Management in Reactive Manner

In SDN, the control plane manages flow rules in two ways, such as reactive and proactive. In reactive flow rule management, after a flow arrives, the controller reactively plans a routing path for the flow by setting up flow rules, which

[3]Under the default routing strategy, a flow is assigned a given default routing path to its destination [22], [23]. Under the dynamic routing strategy, each flow is dynamically planned a routing path to its destination [15], [24].

[4]An ϵ-approximation scheme is a polynomial-time approximation scheme (PTAS) if its running time is polynomial both in n (the size of the problem) [25].

is efficient and widely used. Then, a series of works [12], [14], [15], [24] investigate how to optimize the reactive flow rule management. Jin et al. [24] used experimental results to show that the flow setup process is time-consuming, and the heavy load on the SDN controller may cause long-tail flow setup latency. To overcome this problem, [12] proposed a dynamic multi-controller load balancing algorithm, in which flow requests are dynamically scheduled among multiple controllers to adapt to the dynamic traffic load. Besides, [15] proposed approximation algorithms with boundary factors to efficiently manage flow rules in the data plane to alleviate controller overhead under the limited TCAM capacity. Moreover, [14] proposes an effective dynamic controller allocation mechanism to minimize the flow setup latency. The above work significantly facilitates reactive flow rule management and greatly improves the performance of traditional applications. However, they inevitably introduce flow setup latency and fail to support ultra-low latency services with mobility.

B. Flow Rule Management in Proactive Manner

In the proactive flow rule management, before a flow arrives, the control plane proactively sets up rules for a flow, so that the flow can be forwarded directly after arrival, which avoids introducing flow setup latency. Then, several research efforts [16], [17], [18], [19], [26] investigate this topic. First, Hana et al. [26] proposed several prediction algorithms for predicting user mobility transition probabilities in cellular networks. Then, Li et al. [16] proposed an efficient heuristic algorithm to proactively set flow rules on neighboring small cells for users' flow requests based on its mobility transition probability, aiming to minimize TCAM occupancy while satisfying a given flow setup hit ratio (FSHR). Besides, Yeunwoong et al. [17] proposed a proactive flow setup algorithm, which prioritizes setting flow rules for latency-sensitive flows to improve application performance while minimizing TCAM occupancy. Moreover, to alleviate the controller load, Maity et al. [18] proposed an effective mechanism to predict user mobility and determine the optimal controller-switch assignments to minimize the control plane cost. Finally, for energy efficiency, Bera et al. [19] proposed a flow rule placement mechanism to proactively set flow rules for each user at the location with the greatest transition probability and minimize the number of activated small cells. The above studies moderately proactively set up flow rules to improve application performance while minimizing TCAM occupation, controller load, and energy consumption. These efforts moderately improve the FSHR, but they aim to maximize resource efficiency and ignore bandwidth constraints, which prevents available resources from being fully and efficiently utilized to maximize the FSHR, which is critical for latency-sensitive services.

Different from the above studies, we investigate how to proactively set flow rules to efficiently and fully utilize available network resources to maximize FSHR, thereby eliminating flow setup latency as much as possible and improving services with ultra-low latency requirements.

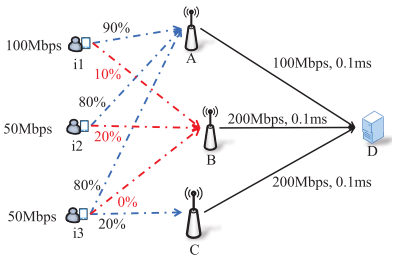


Fig. 3. An example for explaining motivation.

III. MOTIVATION

In SDMENs, as shown in Fig. 1, each network device maintains a flow table that stores flow rules, which is responsible for instructing flow forwarding and allocating bandwidth. The control plane can dynamically manage flows in the data plane by setting up flow rules. For the control plane, there are two ways to set up flow rules, such as reactive and proactive, as shown in Fig. 2. When a user moves beyond the coverage of the currently visited small cell, the user needs to access a new small cell. In the reactive flow setup mode, the SDN controller needs to set up flow rules for the flow reactively, which introduces additional flow setup latency. The experimental results in [27] show that, even in a small-scale network (10 network devices), the flow setup latency takes 9-18 ms, which causes delay jitter and deteriorates latency-sensitive services. To make matters worse, when the control plane consists of multiple controllers, the flow setup latency is even longer, which may exceed 100 milliseconds [28], which severely degrades latency-sensitive services. For example, as shown in Fig. 3, the user $i1$ is currently accessing small cell B, and its flow has an end-to-end latency (the time it takes to transmit a packet from its source to its destination) of 0.1 ms. The flow setup latency is 100 ms. If the user switches to small cell A and introduces additional flow setup latency, then its latency increases sharply to 100.1 ms. This may seriously violate the AR/VR latency requirement (1 ms), causing latency jitter and a worse user experience.

An ideal solution is to proactively set up flow rules for each user's service flow based on their mobility transition probability. That is, the control plane sets the flow rules for the user flow in advance according to its mobility transition probability. Once the set flow rule hits, the flow setup latency is eliminated. Obviously, a high flow set hit ratio (FSHR) is crucial, and the higher its value, the higher the hit probability of proactively set flow rules. The FSHR is equal to the hit probability of proactively set flow rules plus the probability that the user maintains a connection with the currently accessed small cell (which has been set up with flow rules). For example, as shown in Fig. 3, in the next time slot, user $i3$ may maintain the connection with the currently accessed small cell B with probability 0%, and switch to small cells A and C with probability 80% and 20%. If we proactively set up flow rules on small cell A for its request, then its FSHR$= 80\%+0\% = 80\%$. Obviously, if we proactively set up flow rules on more small cells (C) for a user's ($i3$) request, the

FSHR is higher. However, the existing work fails to efficiently and fully utilize network resources to maximize the FSHR. To illustrate this point, we give an example as follows.

As shown in Fig. 3, there are three users $i1$, $i2$, and $i3$ in the network. They currently access small cell B and transmit their traffic to server D at 100 Mbps, 50 Mbps, and 50 Mbps. We consider that the wireless bandwidth requirements of users can be satisfied.[5] Let $\mathcal{P}^{i,s}$ denote the mobility transition probability of user i $(i1, i2, i3)$ to small cell s (A, B, C) in the next time slot. In the next time slot, they maintain the connection with the small cell B with a mobility transition probability of 10%, 20%, and 0%, that is, $\mathcal{P}^{i1,B} = 10\%$, $\mathcal{P}^{i2,B} = 20\%$, $\mathcal{P}^{i3,B} = 0\%$. Besides, we have $\mathcal{P}^{i1,A} = 90\%$, $\mathcal{P}^{i2,A} = 80\%$, $\mathcal{P}^{i3,A} = 80\%$, $\mathcal{P}^{i3,C} = 20\%$. In the network, the available bandwidth on link $A \rightarrow D$ is 100 Mbps, the available bandwidth on links $B \rightarrow D$ and $C \rightarrow D$ is 200 Mbps, and the available TCAM on nodes A, B and C is 3 units.

In this case, Mobi-Flow [19] proactively sets up flow rules for each user ($i1$, $i2$, $i3$) on the small cell (that is, A) with the largest transition probability, which takes into account the TCAM resource constraints and ignores the bandwidth constraints, which may violate the capacity constraints. A simple improved version is to consider the constraints of TCAM and bandwidth resources, and try to set up flow rules for each user on the small cell with the largest transition probability one by one with decreasing transition probability. In this case, MobiFlow only set up flow rules on small cell A for user $i1$, and the total FSHR $= \mathcal{P}^{i1,A} + \mathcal{P}^{i1,B} + \mathcal{P}^{i2,B} + \mathcal{P}^{i3,B} = 120\%$. In fact, proactively set up flow rules on small cell A for users $i2$ and $i3$ instead of $i1$ can get total FSHR $= \mathcal{P}^{i2,A} + \mathcal{P}^{i3,A} + \mathcal{P}^{i1,B} + \mathcal{P}^{i2,B} + \mathcal{P}^{i3,B} = 190\%$. Additionally, proactively set up flow rules for user $i3$ on small cell C can further increase total FSHR by $\mathcal{P}^{i3,C} = 20\%$.

The above example inspired us to study how to proactively set up flow rules, which effectively utilize network resources to maximize total FSHR. In fact, this problem is challenging due to limited TCAM and bandwidth resources, heterogeneous mobility transition probabilities and bandwidth requirements, and complex network topologies. The above factors make it difficult to determine to proactively set up flow rules for which user requests on which small cells. What's more, dynamic routing makes the problem more complicated. For example, as shown in Fig. 2, in the default routing case, the controller only needs to determine whether to set up the flow rule for a flow and route the traffic along a default path (e.g., $A \rightarrow B \rightarrow C \rightarrow E$). In the dynamic routing case, the controller needs to determine whether to set the flow rule for a flow and its routing path (e.g., $A \rightarrow B \rightarrow D \rightarrow E$, or $A \rightarrow C \rightarrow D \rightarrow E$).

IV. Formulation and Analysis

In this section, we first provide a network model, then analyze and formalize the proactive flow setup problem under two typical routing strategies, and finally prove its complexity.

[5]This is reasonable because the user experience rate in 5G network is increased tenfold compared to 4G network, and small cells are deployed ultra-densely [29].

TABLE I
Summary of Notations

Inputs	Definition
G	Directed graph
$V, L, \mathbb{U}, \mathbb{S}$	Set of nodes, links, users, small cells
ς^i	Small cell currently accessed by user i
$\mathbb{S}^i$	Set of neighbor small cells of small cell ς^i
$\mathcal{C}_{u,v}$	Bandwidth capacity of link (u, v)
$\mathcal{C}_u$	TCAM capacity of node u
$\mathcal{P}^{i,s}$	Transition probability of user i from the currently accessed small cell to small cell s in the next time slot
f^i	Service flow of user i
$f^{i,s}$	Service flow of user i on small cell s
$p^{s,d}$	Default routing path from small cell s to edge server d
$p^{s,d}_{u,v}$	Binary constant indicates whether path $p^{s,d}$ traverses link (u, v)
b^i	Required bandwidth of flow f^i
d^i	Destination of flow f^i
Decisions	**Definition**
$x^{i,s}$	Binary variable indicates that whether to proactively set up flow rules for flow $f^{i,s}$
$f^{i,s}_{u,v}$	Binary variable indicates whether flow $f^{i,s}$ traverses link (u, v)

A. Network Model

The SDMENs consist of an SDN controller, a set of nodes, and links. The controller collects status information over the entire network and is responsible for dynamic traffic management by setting up flow rules. To eliminate the flow setup latency, the SDN controller updates the flow rules at time interval Δt, which proactively sets up the flow rules for each user's ongoing service flow.

We abstract the SDMENs as a directed graph $G = (V, L)$, where V and L indicate the node set and link set. The nodes include switches, small cells, and edge servers. Each node $u \in V$ has an available TCAM capacity $\mathcal{C}_u$, and each link $(u, v) \in L$ has an available bandwidth capacity $\mathcal{C}_{u,v}$.

In the network, the user set and small cell set are denoted as $\mathbb{U}$ and $\mathbb{S}$. Each user $i \in \mathbb{U}$ currently accesses small cell ς^i with a service flow $f^i = \{d^i, b^i\}$, where d^i and b^i indicate its destination and required bandwidth. Due to mobility, in the next time slot, user i may maintain the connection with small cell ς^i or switch to a neighbor small cell $s \in \mathbb{S}^i$. We use $f^{i,s}$ to denote the flow of user i on small cell s, then $f^{i,s} = \{s, d^i, b^i\}$. For simplicity, we call $f^{i,s}$ a flow. The transition probability of each user i from currently accessed small cell to $s \in \mathbb{S}^i \cup \varsigma^i$ in the next time slot is denoted as $\mathcal{P}^{i,s}$, which can be obtained by a modified mobility prediction algorithm [30] (detailed in Section VII.A). The notations to be used are listed in Table I.

B. Problem Analysis

In SDMEMs, the next time slot, each user i may access to a small cell $s \in \mathbb{S}^i \cup \varsigma^i$ with probability $\mathcal{P}^{i,s}$, and generate

flow $f^{i,s}$. Once a flow $f^{i,s}$ is set with flow rules, then the flow setup hits; otherwise, the flow setup hit fails. Then the FSHR of user i can be expressed as

$$\mathcal{R}_i = \sum_{s\in\mathbb{S}^i} \mathcal{P}^{i,s} x^{i,s} + \mathcal{P}^{i,\varsigma^i}, \tag{1}$$

where $x^{i,s}$ is a binary decision variable that indicates whether to proactively set up flow rules for flow $f^{i,s}$. The second term implies that the flow setup always hits if user i maintains the connection with the currently accessed small cell ς^i. Naturally, $\sum_{s\in\mathbb{S}^i\cup\varsigma^i} \mathcal{P}^{i,s} = 1$. Then, the total FSHR can be expressed as

$$\mathcal{R} = \sum_{i\in\mathbb{U}} \mathcal{R}_i = \sum_{i\in\mathbb{U}} \sum_{s\in\mathbb{S}^i} \mathcal{P}^{i,s} x^{i,s} + \sum_{i\in\mathbb{U}} \mathcal{P}^{i,\varsigma^i}. \tag{2}$$

C. Problem Formulation

Here, we consider two typical routing strategies, called default routing and dynamic routing. Under the default routing strategy [22], [23], the SDN controller uses wildcards to set the default routing path for flows with the same source and destination. When a new flow arrives and needs to be accepted, the SDN controller only needs to set up a flow rule on the small cell it visits to permit it. Then, the flow is forwarded on the default routing path set with wildcard rules. In this case, fewer TCAM resources need to be consumed, but the traffic suffers fewer dynamics.

Under the dynamic routing strategy [15], [24], the SDN controller uses dedicated flow rules to set the routing path for each flow, which helps adjust the network traffic and accommodate more flows dynamically. However, under this routing strategy, each flow needs to be set with dedicated flow rules, which consumes more TCAM resources. These two routing strategies are designed for different application scenarios[6] and are divided into different 5G network slices. Therefore, these two routing strategies are considered separately, rather than coupled. Then, we formally construct the proactive flow setup problem with default routing and dynamic routing.

1) Proactive Flow Setup and DeFault Routing (PFSDF) Problem: In the case of default routing, each small cell s is configured with a default path to each edge server d, which is denoted as $p^{s,d}$. Then, we only need to determine whether to permit each flow $f^{i,s}$ by setting a flow rule on the small cell s.

Based on the analysis in Section IV-B, we have the objective function as follows

$$\max \sum_{i\in\mathbb{U}} \sum_{s\in\mathbb{S}^i} \mathcal{P}^{i,s} x^{i,s} + \sum_{i\in\mathbb{U}} \mathcal{P}^{i,\varsigma^i}$$

Before setting up flow rules for a flow, we need to check their network resource requirements can be met. This is to prevent network congestion or TCAM overload caused by bandwidth and TCAM capacity being exceeded. Therefore, the objective function is subject to the following constraints.

Capacity constraints: Once a flow $f^{i,s}$ traverses a link (u, v), bandwidth resources need to be reserved on the link. The traffic on each link should not exceed its bandwidth capacity, and the constraints can be expressed as

$$\sum_{i\in\mathbb{U}} \sum_{s\in\mathbb{S}^i} b^i p^{s,d}_{u,v} x^{i,s} \le \mathcal{C}_{u,v}, \forall (u,v) \in L, d = d^i, \tag{3}$$

where $p^{s,d}_{u,v}$ is a binary constant indicates whether path $p^{s,d}$ traverses link (u, v).

Under the default routing policy, when the SDN controller sets flow rules for a flow, it only needs to set the flow rules on the small cell visited by the flow to permit it. The flow rule on each small cell should not exceed its TCAM capacity, and the constraints can be expressed as

$$\sum_{i\in\mathbb{U}} x^{i,s} \le \mathcal{C}_s, \forall s \in \mathbb{S}. \tag{4}$$

Then, PFSDF problem can be formalized as follows

$$\begin{aligned} \max & \sum_{i\in\mathbb{U}} \sum_{s\in\mathbb{S}^i} \mathcal{P}^{i,s} x^{i,s} + \sum_{i\in\mathbb{U}} \mathcal{P}^{i,\varsigma^i} \\ s.t.\ & (3), (4), \\ & x^{i,s} \in \{0,1\}, \forall i \in \mathbb{U}, s \in \mathbb{S}^i. \end{aligned}$$

2) Proactive Flow Setup and DYnamic Routing (PFSDY) Problem: In the case of dynamic routing, each flow is dynamically planned with a routing path. We need to determine whether to set flow rules for each flow and its routing path. Then, we have the following constraints.

a) Flow constraints: Once a flow $f^{i,s}$ enters a node $u \in V$, it leaves that node, unless it is the source node that generates traffic or it is the destination node that consumes traffic. The constraints are expressed as

$$\sum_{\{v|(u,v)\in L\}} f^{i,s}_{u,v} - \sum_{\{v|(v,u)\in L\}} f^{i,s}_{v,u} = \begin{cases} 1, & \text{if } u = s, \\ -1, & \text{if } u = d^i, \\ 0, & \text{otherwise}, \end{cases} \quad \forall u \in V, i \in \mathbb{U}, s \in \mathbb{S}^i, \tag{5}$$

where $f^{i,s}_{u,v}$ is a binary variable indicates whether flow $f^{i,s}$ traverses link (u, v).

b) Capacity constraints: Similar to Eq. (3), the bandwidth capacity constraints can be expressed as

$$\sum_{i\in\mathbb{U}} \sum_{s\in\mathbb{S}^i} b^i f^{i,s}_{u,v} x^{i,s} \le \mathcal{C}_{u,v}, \forall (u,v) \in L. \tag{6}$$

Under the dynamic routing strategy, each flow requires dedicated flow rules. Once a flow $f^{i,s}$ traverses a node u, the SDN controller needs to set a flow rule on the node for it. The flow rule on each node should not exceed its TCAM capacity, and the constraints can be expressed as

$$\sum_{i\in\mathbb{U}} \sum_{s\in\mathbb{S}^i} \sum_{\{v|(u,v)\in L\}} f^{i,s}_{u,v} x^{i,s} \le \mathcal{C}_u, \forall u \in V. \tag{7}$$

[6]The default routing strategy consumes less TCAM resources and is suitable for scenarios where TCAM resources are scarce, such as IoT scenarios with massive devices. In this scenario, the TCAM resources allocated to serving network slices with ultra-low latency requirements are scarce. The dynamic routing strategy provides higher throughput, and is suitable for scenarios where bandwidth is scarce, such as scenarios with intensive video services. In this scenario, the bandwidth resources allocated to service network slices with ultra-low latency requirements are scarce.

In summary, PFSDY problem can be formulated as

$$\max \sum_{i\in\mathbb{U}}\sum_{s\in\mathbb{S}^i}\mathcal{P}^{i,s}x^{i,s}+\sum_{i\in\mathbb{U}}\mathcal{P}^{i,\varsigma^i}$$
$$s.t.\quad (5),(6),(7),$$
$$x^{i,s}\in\{0,1\}, f^{i,s}_{u,v}\in\{0,1\}, \forall i\in\mathbb{U}, s\in\mathbb{S}^i, (u,v)\in L.$$

Obviously, both PFSDF problem and PFSDY problem are constructed as integer linear programming problems, and solving such problems is generally complicated. Next, we formally analyze its complexity.

D. Problem Complexity

Theorem 1: PFSDF problem is NP-hard and has no FPTAS.[7]

Proof: See Appendix A. □

Theorem 2: PFSDY problem is NP-hard and has no FPTAS.

Proof: Obviously, PFSDF problem is a special instance of PFSDY problem, by fixing the routing path of each flow and setting the available TCAM of nodes other than small cells to be unlimited. Similar to the proof in Theorem 1, this theorem can be proved, and it is omitted for brevity. □

The above theorems exclude the polynomial-time optimal solution, pseudo-polynomial time algorithm, and FPTAS for PFSDF problem, unless P=NP. This means that maximizing the total FSHR is hard to achieve under limited TCAM and bandwidth resources. Although there is a PTAS for the multi-knapsack problem [25], it can not be applied to PFSDF problem. Because in this problem, flows may compete for network resources on links and nodes on their routing paths, rather than a single knapsack, and each of them may become a bottleneck. To tackle this problem, we propose a polynomial-time algorithm with guaranteed performance in the next section. Moreover, PFSDY problem is more complicated than PFSDF problem due to dynamic routing. Therefore, we propose an efficient heuristic algorithm in Section VI.

V. Algorithm Design for Default Routing

This section designs an algorithm based on LP relaxation for PFSDF problem and analyzes its performance.

A. Algorithm Design

The designed algorithm consists of two components: an LP-based relaxation algorithm and a proactive flow setup and default routing algorithm. The former solves linear programming to inspire flow setup decisions. The latter is responsible for trying different flow setup decision combinations to approach the optimal solution.

1) Linear Relaxation: By relaxing the binary variable $x^{i,s}$ in PFSDF problem, we obtain a LP as follows

$$\max \sum_{i\in\mathbb{U}}\sum_{s\in\mathbb{S}^i}\mathcal{P}^{i,s}x^{i,s}+\sum_{i\in\mathbb{U}}\mathcal{P}^{i,s^i}$$
$$s.t.\quad (3),(4),$$
$$x^{i,s}\in[0,1], \forall i\in\mathbb{U}, s\in\mathbb{S}^i.$$

[7]An ϵ-approximation scheme is a fully polynomial time approximation scheme (FPTAS) if its running time is polynomial both in n and in $\frac{1}{\epsilon}$ [25].

Algorithm 1 LP-Based Algorithm

Input: L: links; V: nodes; $\mathcal{C}_{u,v}$: available bandwidth; $\mathcal{C}_u$: available TCAM; $p^{s,d}$: default routing path; $\mathcal{P}^{i,s}$: transition probability; $\mathbb{F}$: flow set.

Output: $\mathbb{D}'$: flow set that needs to be setup with flow rules; $\mathcal{R}'$: flow setup hit ratio.

```
Compute the optimal solution X̃^{i,s} of LP;
X_I ← {f^{i,s} | X̃^{i,s} = 1};
X_F ← {f^{i,s} | 0 < X̃^{i,s} < 1};
R' ← Σ_{j∈X_I} P^{i,s};
D' ← X_I;
for f^{i,s} ∈ X_F do
  if R' < P^{i,s} then
    if constraints (3), (4) are satisfied for f^{i,s} then
      R' ← P^{i,s};
      D' ← {f^{i,s}};
    end if
  end if
end for
```

2) LP-Based Algorithm: We design an LP-based algorithm as shown in Algorithm 1, which makes flow setup decisions based on the optimal solution obtained by solving LP.

In Algorithm 1, we first solve LP to obtain its optimal solution $\tilde{X}^{i,s}$. Then, we use $\mathbb{X}_I$ and $\mathbb{X}_F$ to record the flows with positive integer and fractional values in the LP solution. Based on this, we initialize the best FSHR and the set of flows that needs to be set up with flow rules. In this operation, only flows with positive integer values are selected into the flow set to be set with flow rules, and the corresponding FSHR is calculated. This operation means that we round down the optimal solution of the LP, including integer and fractional values. This is intended to ensure constraints. Finally, we check whether there is a flow in $\mathbb{X}_F$ (fractional value) that has an FSHR greater than the current best solution $\mathcal{R}'$, and satisfies the capacity constraint (3), (4). If both are satisfied, then we replace the current best solution and replace the corresponding flow set.

3) Proactive Flow Setup and Default Routing: The Proactive Flow Setup and DeFault routing (PFS-DF) algorithm is triggered periodically to deal with user mobility. After being triggered, the SDN controller updates the small cell currently accessed by each user and the transition probability of each user, and then makes proactive flow setup decision. The detailed process of PFS-DF algorithm is shown in Algorithm 2. We first define a variable $\mathcal{R}$ and a set $\mathbb{D}$ to record the best FSHR and the corresponding flow set, and then initialize the flow setup decisions, candidate flows, and flow attributes. Then, we define a variable ℓ to limit the size of the tried flow combination set, which is determined by the constant input parameter ϵ. Further, we try all flow combinations with elements not exceeding ℓ. If a flow set satisfies the capacity constraint, then we call Algorithm 1 to utilize the updated available network resources. Subsequently, we record the best FSHR and the corresponding flow set. Finally, we make flow setup decisions based on the recorded best flow set.

Algorithm 2 Proactive Flow Setup and Default Routing

Input: $\mathbb{U}$: user set; f^i: user service flow; $\mathcal{C}_{u,v}$: available bandwidth; $\mathcal{C}_u$: available TCAM; $p^{s,d}$: default routing path; $\mathcal{P}^{i,s}$: transition probability; ϵ: constant parameter.
Output: $x^{i,s}$: flow setup decision.

$\mathcal{R} \leftarrow 0$;
$\mathbb{D} \leftarrow \emptyset$;
$x^{i,s} \leftarrow 0, \forall i \in \mathbb{U}, s \in \mathbb{S}^i$;
$\mathbb{F} \leftarrow \{f^{i,s} | \forall i \in \mathbb{U}, s \in \mathbb{S}^i\}$;
$f^{i,s}.src, f^{i,s}.dst, f^{i,s}.bw \leftarrow (s, d^i, b^i), \forall i \in \mathbb{U}, s \in \mathbb{S}^i$;
$d \leftarrow |L| + |\mathbb{S}|$;
$\ell \leftarrow \min\{\lceil \frac{d}{\epsilon} \rceil - (d+1), n\}$;
for all subsets $\mathbb{F}' \subseteq \mathbb{F}$ with $|\mathbb{F}'| \leq \ell$ **do**
  if constraints (3), (4) are satisfied for $\mathbb{F}'$ **then**
    $\mathbb{F}'' \leftarrow \mathbb{F} - \mathbb{F}'$;
    $\mathcal{C}'_{u,v} \leftarrow \mathcal{C}_{u,v} - \sum_{f^{i,s} \in \mathbb{F}'} b^i p^{s,d^i}_{u,v}, \forall (u,v) \in L$
    $\mathcal{C}'_s \leftarrow \mathcal{C}_s - \sum_{f^{i,s} \in \mathbb{F}'} 1, \forall s \in \mathbb{S}$
    $\mathcal{R}', \mathbb{D}' \leftarrow$ call Algorithm 1 with the updated flow set $\mathbb{F}''$ and available network resources $\mathcal{C}'_{u,v}, \mathcal{C}'_u$;
    if $\mathcal{R} < \sum_{(i,s) \in \mathbb{F}'} \mathcal{P}^{i,s} + \mathcal{R}'$ **then**
      $\mathcal{R} \leftarrow \sum_{(i,s) \in \mathbb{F}'} \mathcal{P}^{i,s} + \mathcal{R}'$;
      $\mathbb{D} \leftarrow \mathbb{F}' + \mathbb{D}'$;
    end if
  end if
end for
for $f^{i,s} \in \mathbb{D}$ **do**
  $x^{i,s} \leftarrow 1$;
end for

B. Algorithm Analysis

To facilitate algorithm analysis, we define

$$\zeta = \max\{|\mathbb{S}^i|, \forall i \in \mathbb{U}\}. \tag{8}$$

In a typical cellular network [31], each small cell is surrounded by 6 small cells, where $\zeta = 6$.

Theorem 3: PFS-DF algorithm runs in polynomial time by setting ϵ as a constant greater than or equal to $\frac{d}{\delta+d+1}$, where δ is an arbitrary positive constant and $d = |L| + |\mathbb{S}|$.

Proof: As defined in Algorithm 2, $\ell = \min\{\lceil \frac{d}{\epsilon} \rceil - (d+1), n\}$. Since $\epsilon \geq \frac{d}{\delta+d+1}$ and δ is an positive constant, $\ell = \min\{\lceil \frac{d}{\epsilon} \rceil - (d+1), n\} = \min\{\lceil \delta \rceil, n\} \leq \lceil \delta \rceil$, which means that ℓ is less than or equal to a constant. Next, we analyze Algorithm 1. The algorithm first solves the LP, which can be effectively solved in polynomial time by the ellipsoid algorithm [32]. Obviously, the rest of Algorithm 1 runs in polynomial time, including initialization parameters and checking capacity constraints. Therefore, Algorithm 1 runs in polynomial time. Then, we further analyze the Algorithm 2. The algorithm first needs $O(\zeta|\mathbb{U}|)$ to initialize the parameters. Then, the algorithm tries $O(n^\ell)$ flow combinations. Since each flow set can be checked in polynomial time whether the capacity constraints are satisfied, and Algorithm 1 is polynomial time, Algorithm 2 also runs in polynomial time. □

Theorem 4: PFS-DF algorithm is a $\frac{\ell+1}{\ell+d+1}$-approximation algorithm.

Proof: By treating each link and node as a knapsack, PFSDF problem is a special case of the multi-dimensional knapsack problem. The above designed algorithm is proved to be $\frac{\ell+1}{\ell+d+1}$-approximation algorithm, please refer to [25] for details. □

Theorem 5: PFS-DF algorithm is a PTAS, when d does not exceed a constant κ.

Proof: Generally speaking, each SDN controller is responsible for a limited network scale, so that it can quickly respond to dynamic requests in the data plane. In such a network, there are limited network nodes and links, that is, $d \leq \kappa$. In this case, we analyze the algorithm as follows.

First, we prove that PFS-DF runs in polynomial time for $\epsilon \in [0, 1]$. We define a constant $\eta = \frac{\kappa}{\epsilon} - \kappa$. Since $d \leq \kappa$ and $\ell = \min\{\lceil \frac{d}{\epsilon} \rceil - (d+1), n\}$, then $\ell \leq \lceil \frac{d}{\epsilon} \rceil - (d+1) \leq \lceil \frac{\kappa}{\epsilon} \rceil - (\kappa+1) \leq \frac{\kappa}{\epsilon} - \kappa \leq \eta$. That is, $\ell \leq \eta$. As shown in PFS-DF algorithm, the parameter ϵ only affects the number of flow combinations. Since $\ell \leq \eta$, the algorithm still tries polynomial flow combinations. Similar to the proof of Theorem 3, PFS-DF algorithm still runs in polynomial time.

Next, we prove that PFS-DF algorithm is a $1 - \epsilon$-approximation algorithm. If $\lceil \frac{d}{\epsilon} \rceil - (d+1) \geq n$, then the PFS-DF algorithm has tried all flow combinations and it is clear that the optimal solution can be obtained. Otherwise, $\ell = \min\{\lceil \frac{d}{\epsilon} \rceil - (d+1), n\} = \lceil \frac{d}{\epsilon} \rceil - (d+1)$. Thus, $\frac{\ell+1}{\ell+d+1} \leq 1 - \epsilon$. Based on Theorem 4, our algorithm is a $(1-\epsilon)$-approximation algorithm.

□

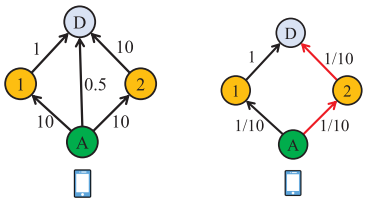
VI. Algorithm Design for Dynamic Routing

In this section, we propose a greedy-based heuristic algorithm for PFSDY problem. The algorithm includes two subfunctions, i.e., bottleneck bypass routing and bottleneck flow setup decision. The former is responsible for determining the routing path of each flow, and the latter greedily tries to set flow rules for flows on network bottlenecks. The heuristic algorithm calls both to make flow setup and routing decisions.

A. Bottleneck Bypass Routing

Inspired by the example in Section III, a flow cannot be set with flow rules when its required network resources cannot be satisfied. Therefore, we tend to route flows to low-load links and nodes rather than potential bottlenecks with high loads. For this purpose, we construct a weighted graph for each flow based on the physical graph, where the greater the proportion of available resources consumed, the greater the weight. In this way, we can find a path that bypasses the bottleneck links and nodes by finding a path with the smallest weight, using an existing algorithm (Dijkstra's algorithm [33]).

1) Construct Weighted Graph: The weighted graph is constructed as follows. To ensure capacity constraints, we first prune those links and nodes in the physical graph that cannot satisfy the network resource requirements of the flow. Besides, we need to establish the relationship between network resource occupancy ratio and link weight. In this way, we can find the path that bypasses the bottleneck links and nodes by finding the path with minimum weight.



(a) Physical Graph (b) Weighted Graph

Fig. 4. An example shows how to construct a weighted graph based on a physical graph.

Algorithm 3 Bottleneck Bypass Routing

Input: $G=(V,L)$: physical graph; $f^{i,s}$: flow; $\mathcal{C}_{u,v}$: available bandwidth; $\mathcal{C}_u$: available TCAM.

Output: $p^{i,s}$: routing path of $f^{i,s}$.

1: **for** each link $(u,v)\in L$ **do**
2: **if** $\mathcal{C}_{u,v}\geq f^{i,s}.bw$ and $\mathcal{C}_u\geq 1$ **then**
3: $G'.addedge(u,v)$;
4: $G'.w_{u,v}\leftarrow \max\{\frac{1}{\mathcal{C}_u},\frac{b^i}{\mathcal{C}_{u,v}}\}$;
5: **end if**
6: **end for**
7: $p^{i,s}\leftarrow G'.Dijkstra(f^{i,s}.src, f^{i,s}.dst)$;

When a flow $f^{i,s}$ traverses a link (u,v), it needs to occupy the TCAM on node u and the bandwidth on the link, both of which may be bottleneck resources. Therefore, we assign a weight to each link as

$$w_{u,v}=\max\{\frac{1}{\mathcal{C}_u},\frac{b^i}{\mathcal{C}_{u,v}}\},\quad \text{if } (\mathcal{C}_{u,v}\geq b^i \text{ and } \mathcal{C}_u\geq 1),\forall(u,v)\in L \tag{9}$$

where $\mathcal{C}_{u,v}$ and $\mathcal{C}_u$ indicate the available bandwidth on link (u,v) and the available TCAM on node u.

To facilitate readers' understanding, as shown in Fig. 4, we give a simple example to illustrate how to construct a weighted graph for a flow based on a physical graph. The physical graph is shown in Fig. 4(a), and each link is marked with its available bandwidth. Each node has one unit of TCAM resource, which is enough to set the flow rules for this flow. There is a user i in the network, and its service flow is $f^i=\{D,1\}$, which means that its destination is D and requires 1 unit of bandwidth. We need to reserve network resources for user i on small cell A, that is, $f^{i,A}$. Then, we construct a weighted graph for it as follows. We first prune those links that fail to satisfy the flow's network resource requirements, such as link (A,D). Then, based on Eq. (9), we assign weight to each link and obtain a weighted graph as shown in Fig. 4(b). Finally, we can find the path to bypass the high-load link (potential bottleneck) by finding the path with the smallest weight, such as $A\rightharpoonup 2\rightharpoonup D$.

2) Algorithm Description: The bottleneck bypass routing algorithm is shown in Algorithm 3. We first construct a weighted graph based on the physical graph (lines 1-6). Then, we use the well-known Dijkstra algorithm to find a minimum weight path for flow $f^{i,s}$, which traverses low-load links and nodes.

Algorithm 4 Bottleneck Flow Setup Decision

Input: $f^{i,s}$: flow; $\mathcal{C}_{u,v}$: available bandwidth; $\mathcal{C}_u$: available TCAM; $p^{i,s}$: routing paths; $f^{i,s}_{u,v}$: flow routing decision; $\mathbb{F}$: candidate flow set; $\mathcal{L}_{u,v}$: link load factor; $\mathcal{L}_u$: node load factor.

Output: $\mathbb{F}'''$: The set of flows that need to be setup with flow rules.

1: Calculate $\Phi_B^{i,s}$ and $\Phi_T^{i,s}$ based on Eqs. (12) and (13);
2: $\mathbb{L}\leftarrow\{(u,v)\in L|\mathcal{L}_{u,v}>1\}$;
3: $\mathbb{N}\leftarrow\{u\in V|\mathcal{L}_u>1\}$;
4: **while** $\mathbb{L}!=\emptyset$ or $\mathbb{N}!=\emptyset$ **do**
5: $\mathfrak{L}\leftarrow\arg\max_{(u,v)}\{\mathcal{L}_{u,v},\forall(u,v)\in\mathbb{L}\}$;
6: $\mathfrak{N}\leftarrow\arg\max_u\{\mathcal{L}_u,\forall u\in\mathbb{V}\}$;
7: **if** $\mathcal{L}_{\mathfrak{L}}\geq\mathcal{L}_{\mathfrak{N}}$ **then**
8: $\mathbb{L}=\mathbb{L}-\{\mathfrak{L}\}$;
9: $\mathbb{F}'\leftarrow\{f^{i,s},f^{i,s}\in\mathbb{F}|\mathfrak{L}\in p^{i,s}\}$;
10: Sort $\mathbb{F}'$ in descending order of priority index $\Phi_B^{i,s}$;
11: **else**
12: $\mathbb{N}=\mathbb{N}-\{\mathfrak{N}\}$;
13: $\mathbb{F}'\leftarrow\{f^{i,s},f^{i,s}\in\mathbb{F}|\mathfrak{N}\in p^{i,s}\}$;
14: Sort $\mathbb{F}'$ in descending order of priority index $\Phi_T^{i,s}$;
15: **end if**
16: $\mathbb{F}''\leftarrow\emptyset$;
17: **for** each flow $f^{i,s}\in\mathbb{F}'$ **do**
18: **if** the capacity constraint are satisfied for $\mathbb{F}''+f^{i,s}$ **then**
19: $\mathbb{F}''\leftarrow\mathbb{F}''+f^{i,s}$;
20: **end if**
21: **end for**
22: $f^{i',s'}\leftarrow\arg\max\{\mathcal{P}^{i,s},\forall f^{i,s}\in\mathbb{F}'\}$;
23: **if** $\mathcal{P}^{i',s'}>\sum_{f^{i,s}\in\mathbb{F}''}\mathcal{P}^{i,s}$ and the capacity constraint is satisfied for $f^{i',s'}$ **then**
24: $\mathbb{F}''\leftarrow f^{i',s'}$;
25: **end if**
26: $\mathbb{F}'''\leftarrow\mathbb{F}'''+\mathbb{F}''$;
27: **end while**

B. Bottleneck Flow Setup Decision

In this section, we design a bottleneck flow setup decision algorithm, which identifies network bottlenecks and determines which flows on the network bottlenecks to set flow rules. Specifically, we first define the load factor to identify network bottlenecks, and then define the priority indicator to determine which flows are preferentially set with flow rules.

1) Network Load Factor: To determine the network bottleneck, we define the network load factor as follows.

$$\mathcal{L}_u=\frac{\sum_{f^{i,s}\in\mathbb{F},v|(u,v)\in L}f^{i,s}_{u,v}}{\mathcal{C}_u},\forall u\in V, \tag{10}$$

$$\mathcal{L}_{u,v}=\frac{\sum_{f^{i,s}\in\mathbb{F}}f^{i,s}_{u,v}b^i}{\mathcal{C}_{u,v}},\forall(u,v)\in L, \tag{11}$$

where $\mathcal{L}_u$ denotes the node load factor, and $\mathcal{L}_{u,v}$ denotes the link load factor. The larger the load factor value, the higher the network load. Once their value exceeds 1, they become network bottlenecks, that is, their load exceeds their capacity.

2) Priority Indicator: To determine which flows on the network bottleneck are preferentially set flow rules, we define the priority indicator as follows.

$$\Phi_B^{i,s} = \frac{\mathcal{P}^{i,s}}{b^i}, \forall i \in \mathbb{U}, s \in \mathbb{S}^i, \tag{12}$$

$$\Phi_T^{i,s} = \mathcal{P}^{i,s}, \forall i \in \mathbb{U}, s \in \mathbb{S}^i, \tag{13}$$

where $\Phi_B^{i,s}$ and $\Phi_T^{i,s}$ are defined based on TCAM and bandwidth resources, respectively. The motivation behind Eq. (12) is that we give priority to setting up flow rules for flows with high transition probability, which helps to obtain a high FSHR. Moreover, the fewer bottleneck resources a flow requires, the higher its priority, which helps to utilize network resources efficiently.[8] The motivation for defining Eq. (13) is similar to Eq. (12). It should be noted that setting flow rules on a node for each flow only consumes one unit of TCAM.

3) Algorithm Description: The details of the bottleneck flow setup decision algorithm as shown in Algorithm 4. We first calculate the priority indicator for each flow. Then, we obtain those bottleneck links and nodes. Furthermore, we iteratively determine which flows on the link or node with the highest load factor to set flow rules. Specifically, we first obtain the link and node with the highest load factor. If the bottleneck with the largest load factor is a link (node), then we get the flows on this link (node) and sort those flows in descending order by their priority indicator, and vice versa (lines 7-15). Further, we determine whether each flow can be set with flow rules one by one based on capacity constraints (lines 17-21). If the capacity constraints are satisfied, then this flow can be set with flow rules. Finally, if a flow can obtain greater benefits and does not violate the capacity constraint, we replace the above flow setup decision with this flow (22-25). Finally, we update the flow set that is determined to be set with flow rules. This iteration is terminated until all bottleneck links and nodes have performed the above steps.

The above-designed algorithm is inspired by Ext-Greedy algorithm in [25]. In this algorithm, we greedily set flow rules for each flow based on their value density (priority indicator).

C. Proactive Flow Setup and Dynamic Routing

Similar to PFS-DF algorithm, the Proactive Flow Setup and DYnamic routing (PFS-DY) algorithm is also triggered periodically to deal with user mobility. The PFS-DY algorithm makes flow setup and routing decisions based on Algorithm 3 and Algorithm 4. Its details are shown in Algorithm 5.

In the algorithm, we first initialize the flow setup decisions, flow routing decisions, candidate flow set, and flow attributes (lines 1-4). Then, we call Algorithm 3 to plan the routing path for each flow and delete those flows that have no feasible routing path (lines 6-12). Besides, we calculate the load factors of nodes and links. Further, we set flow rules for those flows that do not traverse potential bottleneck links or nodes, and update available network resources (lines 14-23). Finally, we call Algorithm 4 to determine which flows on the network bottlenecks should be set flow rules, and update flow setup decisions and available network resources (lines 24-28). The algorithm terminates until all flows are set up with flow rules, or no flow has a feasible routing path.

[8]The probability term in Eq. (12) is distributed in [0-1], and the bandwidth term b^i is in MB and distributed in (0, 1000]. Therefore, such a division without normalization is acceptable for float-type variables.

Algorithm 5 Proactive Flow Setup and Dynamic Routing

Input: $\mathbb{U}$: user set; f^i: user service flow; $\mathcal{C}_{u,v}$: available bandwidth; $\mathcal{C}_u$: available TCAM.
Output: $x^{i,s}$: flow setup decision; $f_{u,v}^{i,s}$: flow routing decision.

1: $x^{i,s} \leftarrow 0, \forall i \in \mathbb{U}, s \in \mathbb{S}^i$;
2: $f_{u,v}^{i,s} \leftarrow 0, \forall i \in \mathbb{U}, s \in \mathbb{S}^i, (u,v) \in L$;
3: $\mathbb{F} \leftarrow \{f^{i,s} | \forall i \in \mathbb{U}, s \in \mathbb{S}^i\}$;
4: $f^{i,s}.src, f^{i,s}.dst, f^{i,s}.bw \leftarrow (s, d^i, b^i), \forall i \in \mathbb{U}, s \in \mathbb{S}^i$;
5: **while** $\mathbb{F} != \emptyset$ **do**
6: **for** each flow $f^{i,s} \in \mathbb{F}$ **do**
7: $p^{i,s} \leftarrow$ call Algorithm 3;
8: $f_{u,v}^{i,s} \leftarrow 1, \forall (u,v) \in p^{i,s}$;
9: **if** $p^{i,s} = NoPath$ **then**
10: $\mathbb{F} \leftarrow \mathbb{F} - f^{i,s}$;
11: **end if**
12: **end for**
13: Calculate $\mathcal{L}_{u,v}$ and $\mathcal{L}_u$ based on Eq. (10) and (11);
14: **for** each flow $f^{i,s} \in \mathbb{F}$ **do**
15: **if** $\mathcal{L}_{u,v} \leq 1, \forall (u,v) \in p^{i,s}$ **then**
16: **if** $\mathcal{L}_u \leq 1, \forall u \in p^{i,s}$ **then**
17: $x^{i,s} \leftarrow 1$;
18: $\mathbb{F} \leftarrow \mathbb{F} - f^{i,s}$;
19: $\mathcal{C}_{u,v} \leftarrow \mathcal{C}_{u,v} - b^i, \forall (u,v) \in p^{i,s}$;
20: $\mathcal{C}_u \leftarrow \mathcal{C}_u - 1, \forall u \in p^{i,s}$;
21: **end if**
22: **end if**
23: **end for**
24: $\mathbb{F}''' \leftarrow$ call Algorithm 4;
25: $x^{i,s} \leftarrow 1, \forall (i,s) \in \mathbb{F}'''$;
26: $\mathbb{F} \leftarrow \mathbb{F} - \mathbb{F}'''$;
27: $\mathcal{C}_{u,v} \leftarrow \mathcal{C}_{u,v} - b^i, \forall (u,v) \in p^{i,s}, f^{i,s} \in \mathbb{F}'''$;
28: $\mathcal{C}_u \leftarrow \mathcal{C}_u - 1, \forall u \in p^{i,s}, f^{i,s} \in \mathbb{F}'''$;
29: **end while**

D. Algorithm Analysis

Theorem 6: PFS-DY algorithm produces a feasible solution in $O(|L||\mathbb{U}|^2 \log |V| + (|L| + |V|) \cdot (|L||\mathbb{U}| + |V||\mathbb{U}| + |\mathbb{U}|^2 \log |\mathbb{U}|))$.

Proof: We first analyze the feasibility of the solution produced by PFS-DY algorithm as follows. As shown in Algorithms 3 and 5, we search for a feasible routing path for each flow and delete those flows that have no feasible routing path. A flow can be set with flow rules only if it has a feasible routing path and thus satisfies the flow constraints. Besides, as shown in PFS-DY algorithm, we check the capacity constraints before setting flow rules for a flow. A flow can be set with flow rules only when the capacity constraints are met. Based on the above analysis, it is obvious that PFS-DY algorithm always produces a feasible solution.

We next analyze the complexity of PFS-DY algorithm as follows. We first analyze the complexity of Algorithm 3,

which requires $O(|L|)$ to construct a weighted graph, and $O(|L|\log|V|)$ to search for the least weight routing path. Therefore, the complexity of Algorithm 3 is $O(|L|\log|V|)$. Then, we analyze Algorithm 4, which requires $O(|L|+|V|+\zeta|\mathbb{U}|)$ to calculate the parameters. Then, the algorithm iterates $O(|L|+|V|)$ times in the worst case. Each iteration runs in $O(|L|+|V|+\zeta|\mathbb{U}|\log\zeta|\mathbb{U}|)$. Therefore, its complexity is $O((|L|+|V|)\cdot(|L|+|V|+\zeta|\mathbb{U}|\log\zeta|\mathbb{U}|))$. Finally, we analyze Algorithm 5, which requires $O(\zeta|\mathbb{U}||L|))$ to initialize the flow setup decision, flow routing decision, candidate flow set, and flow attributes. Then, the algorithm iterates $O(\zeta\mathbb{U})$ times in the worst case. Each iteration runs in $O(\zeta|L||\mathbb{U}|\log|V|+(|L|+|V|)\cdot(|L|+|V|+\zeta|\mathbb{U}|\log\zeta|\mathbb{U}|))$. Based on the above analysis and ζ is a constant, we can conclude that the complexity of PFS-DY algorithm is $O(|L||\mathbb{U}|^2\log|V|+(|L|+|V|)\cdot(|L||\mathbb{U}|+|V||\mathbb{U}|+|\mathbb{U}|^2\log|\mathbb{U}|))$. □

Theorem 7: PFS-DY algorithm is a 2-approximation algorithm in a special case, in which the topology is a tree, and there is only one bottleneck link or node.

Proof: In fact, tree topologies are common [34]. For example, small cells are connected to macrocells, and macrocells are connected to the edge server, which forms a tree topology [35]. In a tree network, there is only one feasible shortest path between each pair of nodes. Therefore, the routing path of each flow in our algorithm is the same as the routing path in the optimal solution. Since there is only one bottleneck link (node) in the network, all flows can be set flow rules except those that traverse the bottleneck link (node). In this case, the PFSDY problem is equivalent to the 0-1 knapsack problem. The Algorithm 4 is designed to make flow setup decisions for flows on the bottleneck link (node), and it makes decisions based on their value density (priority indicator). This algorithm is a modified version of the Ext-Greedy algorithm, and its approximation ratio is proved to be 2 [25]. □

VII. Evaluation

In this section, we first present the evaluation settings and then discuss the simulation and experimental results.

A. Settings

1) Evaluation Environment: We test our algorithms in both experimental and simulation environments. The experimental environment is based on our Ryu [36]/Mininet-WiFi [37] testbed, running Ryu 4.34, OpenFlow v1.3 and OpenVSwitch v2.3. Mininet-WiFi is an extended version of Mininet [38] for wireless emulation. Besides, the numerical simulation is based on Python 3.7. They are run on a server with a 12-core CPU and 16G RAM, equipped with Ubuntu 18.04 operating system.

2) Topology: The tested network topology refers to the ITU-T standard [35], in which small cells and macro cells are connected in a typical Star, or Ring topology [39]. Each of them contains 4 macro cells (as switches) and 16 small cells. Specifically, in the Star topology, each macro cell is directly connected with four small cells, while in the Ring topology, each macro cell is connected to a ring with four small cells. Besides, all macro cells are connected to a small edge cloud (Fat-Tree [40] with 4 Pods). Each small cell covers a 400m $\times$ 400m area [10], and their deployment strategy is grid-based [19].

3) Mobile Model: In our evaluation, the mobility model is derived from taxis, and the mobility transition probability is derived from the taxi data set [41] using a modified Markov-based mobility prediction algorithm [30].[9] The taxi data set is obtained from the trajectory of 500 taxis in San Francisco within 30 days. To adapt to our experiment, we selected mobile data of 1600m $\times$ 1600m hotspot area from each data set.

4) Service Flow: The service flow in our experiment is generated based on a real Internet traffic matrix [42]. The traffic matrix is collected from the European Research and Educational Network for 4 months. It contains information about each service flow, such as its source, destination, and required bandwidth.

5) Metrics and Parameter Settings: In numerical simulations, we evaluate macro-scale behaviors such as FSHR, bandwidth utilization, TCAM utilization, and execution time. There is no need for a packet-level emulator (Mininet-WiFi) that simulates the micro-scale behavior in the network. In the experimental evaluation, we evaluate micro-scale behavior, such as end-to-end latency, which involves the processing of packets. All the data points are collected from 100 runs. The flow rule update period $\Delta t = 10$.[10] Each link has an available bandwidth $\alpha = 10000$ (MB), each node has an available TCAM $\beta = 200$ (flow rules)[11] and $\ell = 0$ (defined in Algorithm 2). The number of users is distributed in [100-500],[12] where each user generates a flow. In the default routing strategy, each flow uses a shortest path from its source to its destination as its routing path. For dynamic routing, each flow dynamically searches for a routing path. Unless otherwise specified, the above parameters are adopted by default.

6) Solutions to Compare: we compare our algorithms with the following solutions.

- Mobi-Flow: This algorithm [19] is the state-of-the-art algorithm for proactive flow setup, which proactively sets

[9]Note that we need to make slight modifications to adapt the original location-based mobility prediction algorithm [30] to our small-cell-based mobility prediction. Specifically, the entire area is divided into multiple locations according to the coverage of small cells, where each location corresponds to a small cell. Once a user moves from one location to another, it means that the user switches from one small cell to another small cell. Then, we can obtain the historical data of the small cells accessed by the user and the corresponding time. Further, divide the whole time into time slots equally, and each time slot corresponds to a small cell (if there are two, record the latter, which indicates that the user switches to this small cell). After the above operations, we can obtain the historical information about the small cell accessed by the user in each time slot. By replacing the location-based history information with the small-cell-based history in Eq. (1) in [30], we can obtain our small-cell-based mobility transition probabilities via the mobility prediction algorithm [30].

[10]This is suitable for the user to switch to a new small cell every 11.3-50.2 s in typical mobile scenarios [11]. It should be noted that the smaller the coverage area of the small cell, the more frequent the user switching, and in this case, the flow rule update period needs to be smaller.

[11]In the 5G network, the physical link bandwidth is about 25-50G, and the device TCAM capacity is about thousands [43]. These physical resources are divided into network slices to serve various services, and a portion of the physical resources are available for our mission-critical services, so the parameter settings are typical and reasonable (most TCAMs are used to serve massive IoT users).

[12]This is a realistic density for users with ultra-low latency and mobility needs, such as vehicle density in autonomous driving (tens to hundreds per square kilometer) [44].

up flow rules for each user in the neighbouring small cell with the largest transition probability.

- MoRule: This algorithm [16] greedily sets up flow rules for each user in the neighboring small cells.
- Reactive: This algorithm reactively sets up flow rules for each user's service flow.
- OPT-DF, OPT-DY: These two algorithms indicate the optimal solution under default routing and dynamic routing, and they are obtained by solving PFSDF problem and PFSDY problem with Gurobi [45] optimizer.

B. Numerical Simulation

1) Compared With the Optimal Solution: Here, we show the gap between our algorithms and the optimal solutions in total FSHR and execution time under default and dynamic routing. The desired performance in the optimal solution is that the total FSHR is maximized. The mobility transition probability is generated from the taxi data set. We set $\alpha = 1000$, $\beta = 50$, which is a case for simulating the small-scale scenario with limited bandwidth and TCAM resources.[13] The simulation result is shown in Fig. 5.

The total FSHR performance of our algorithm under the default routing strategy is shown in Fig. 5(a). PFS-DF algorithm is always close to the optimal solution in terms of total FSHR under the different number of users. Their gap is no more than 5.08%, and their standard deviation is less than 2.49. For example, when the number of users is 60, the total FSHR of the PFS-DF algorithm and the optimal solution is 49.16 and 50.73, which means that the gap between them is only 3.09%. The total FSHR performance of our algorithm under the dynamic routing strategy is shown in Fig. 5(b). PFS-DY algorithm is also close to the optimal solution under the different number of users. Their gap is no more than 0.94%, and their standard deviation is less than 1.16. Moreover, as shown in Fig. 5(c) and Fig. 5(d), the execution time of both PFS-DY algorithm and PFS-DF algorithm is always significantly shorter than that of the optimal solution. Besides, the execution time of OP-DY algorithm and OP-DF algorithm increases exponentially, which is unacceptable for the real world environment. The above results verify that both PFS-DY and PFS-DF are close to optimal solutions and have significantly shorter execution times under default routing and dynamic routing strategies, respectively.

2) Average Flow Setup Hit Ratio: To make the metrics more intuitive, we next evaluate the algorithms in terms of average FSHR, rather than total FSHR. The result of different algorithms under two routing strategies and two network topologies are shown in Fig. 6.

As shown in Fig. 6(a), the average FSHR of PFS-DF algorithm is significantly higher than that of other algorithms. For example, when the number of users is 500, the average FSHR of the PFS-DF, Mobi-Flow, MoRule, and Reactive algorithms are 73.17%, 57.23%, 51.00%, and 40.69% respectively. This means that our algorithm can improve FSHR performance by

[13]Once the bandwidth and TCAM resources are sufficient, then the average FSHR of our algorithms and the optimal solution is always 100%, and such result evaluation is meaningless.

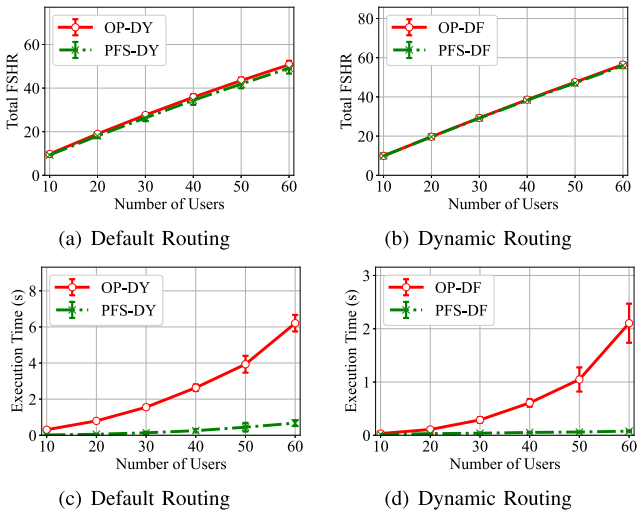


Fig. 5. Compared with the optimal solution under two routing strategies.

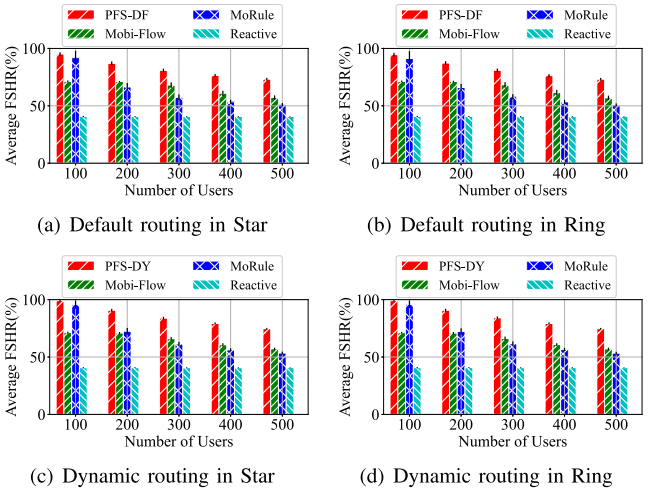


Fig. 6. Results under two network topologies and two routing strategies.

27.85%, 43.47%, and 79.82% compared to the Mobi-Flow, MoRule, and Reactive algorithms, respectively. This is because our algorithm can more efficiently and fully utilize TCAM and bandwidth resources to serve user requests. Besides, as the number of users increases, the average FSHR of all algorithms gradually decreases, which is because more bandwidth requirements exacerbate the bandwidth bottleneck. As shown in Fig. 6(b), in the Ring topology, the average FSHR of PFS-DF algorithm is still significantly higher than other algorithms. When the number of users is 500, the average FSHR of the PFS-DF, Mobi-Flow, MoRule, and Reactive algorithms are 74.72%, 57.16%, 53.45%, and 40.74% respectively. The results show that the four algorithms have lower average FSHR in the Ring topology compared to the Star topology. This is because, in Ring topology, the small cell and the macro cell are connected into a ring, and the flow may traverse small cells on the ring before reaching its destination, which leads to more bandwidth consumption and aggregates the shortage of network resources.

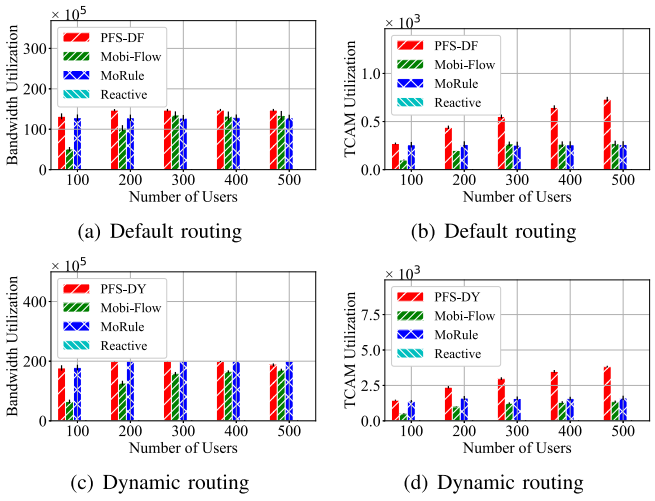


Fig. 7. Bandwidth utilization and TCAM utilization under two routing strategies.

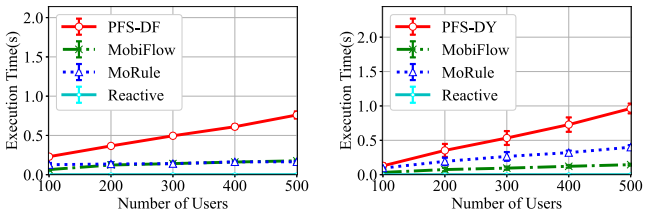
As shown in Fig. 6(c), in the Star topology, PFS-DY algorithm is superior to other algorithms in average FSHR. When the number of users is 500, the average FSHR of the PFS-DY, Mobi-Flow, MoRule, and Reactive algorithms are 73.14%, 56.98%, 50.80%, and 40.73% respectively. This means that our algorithm can improve average FSHR by 28.36%, 43.98%, and 79.57% compared to the Mobi-Flow, MoRule, and Reactive algorithms. Compared with the default routing, dynamic routing has a higher average FSHR. This is because dynamic routing provides more routing opportunities and alleviates the shortage of network resources. As shown in Fig. 6(d), in the Ring topology, PFS-DY algorithm is also superior to other algorithms in average FSHR. When the number of users is 500, PFS-DY algorithm improves the FSHR performance by 30.99%, 40.05%, and 83.48% compared with the Mobi-Flow, MoRule, and Reactive algorithms, respectively.

3) Bandwith and TCAM Utilization: Fig. 7 shows the bandwidth and TCAM utilization of our algorithms in Star topology with limited available bandwidth and TCAM resources. As shown in Fig. 7(a), the bandwidth utilization of PFS-DF algorithm is higher than other algorithms. In Reactive algorithm, no flow rules are proactively set, so no bandwidth resources are reserved. Only when the user switches to a new small cell, the Reactive algorithm reactively sets up flow rules for its flow to allocate bandwidth. With the increase in the number of users, the bandwidth utilization of these algorithms does not increase significantly, because the bandwidth they require has been exhausted. As shown in Fig. 7(a), TCAM utilization of PFS-DF algorithm is higher than that of other algorithms. This means that under the same given TCAM and bandwidth resources, our algorithm can utilize TCAM and bandwidth resources more efficiently to set flow rules for more flows. This is because, a flow can be set with flow rules only when its TCAM and bandwidth resources on all nodes and links on its routing path are satisfied. Our algorithm is designed to jointly consider and efficiently utilize both limited TCAM and bandwidth resources. However, MoRule ignores the limitation of bandwidth resources, so that TCAM and bandwidth resources cannot be efficiently utilized, where flows may not be set with flow rules because bandwidth resources are exhausted, although TCAM resources are idle. Besides, MobiFlow only sets flow rules for each user's flow in the small cell with the greatest transition probability, failing to efficiently utilize the available resources. Besides, as the number of users increases, the increase in TCAM utilization slows down due to limited TCAM and bandwidth resources.

As shown in Fig. 7(b), both PFS-DY and MoRule have high bandwidth utilization, because they can make full use of network resources. Besides, the bandwidth utilization of MoRule is slightly higher than that of PFS-DY. This is because PFS-DY gives priority to setting flow rules for flows with small bandwidth requirements, which may result in a small amount of available bandwidth not being used by flows with large bandwidth. It should be noted that PFS-DY is more resource efficient and can achieve higher FSHR, as mentioned above. Moreover, in these two algorithms, when the number of users is 200, the bottleneck bandwidth resource is exhausted, and the bandwidth utilization does not increase with the increase of the number of users. Besides, when the number of users is 500, PSF-DY consumes less bandwidth than when the number of users is 400. This is because PSF-DY has the potential to set flow rules for flows with shorter paths, which consumes less non-bottleneck bandwidth. Finally, PFS-DY, MobiFlow and MoRule algorithms consume more bandwidth resources in dynamic routing compared to default routing. This is because the shortest path is usually used as the default routing path; while the dynamic routing path may be longer when the available bandwidth on the shortest path is insufficient, which results in more bandwidth resource consumption. As shown in Fig. 7(b), the TCAM utilization of PSR-DY algorithm is still better than other algorithms for dynamic routing. It should be noted that although more flow rules need to be set, it will not degrade service performance. This is because our algorithm is used to set up flow rules in a proactive manner rather than a reactive. This operation is completed before the user accesses the new small cell, and the user traffic can be forwarded directly without delay.[14] The above results verify that our algorithm can utilize TCAM and bandwidth resources more efficiently to improve network performance, which is hard to achieve.

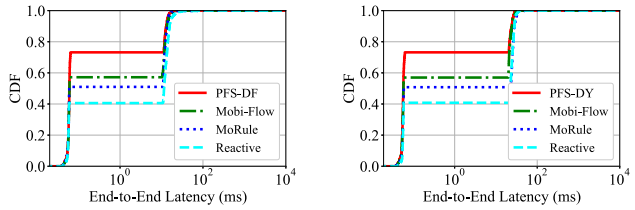
4) Execution Time: To further illustrate the practicality of our algorithms, we evaluate their execution time and discuss their practicality in real-world settings. As shown in Fig. 9(a), the execution time of our algorithm is longer than that of MoRule and MobiFlow. For example, when the number of users is 500, our algorithm needs 0.76s, while MobiFlow and MoRule need 0.18s and 0.16s. This is because more factors are considered in our algorithm to obtain a higher FSHR, which is worthwhile. Since, our algorithm is responsible for proactively setting up flow rules, a high FSHR is critical for it. By contrast, setting up flow rules in a reactive manner delays user traffic, and a short execution time is critical. Similar results can be

[14]The flow setup latency is 91-164 ms [28], and the user switches to a new small cell every 11-50 s [11]. Therefore, it is reasonable to believe that the proactive flow setup process is completed before the user switches to a new small cell.



(a) Default routing (b) Dynamic routing

Fig. 8. Execution time under two routing strategies.



(a) Default routing (b) Dynamic routing

Fig. 9. End-to-end latency under two routing strategies.

found in Fig. 9(b). When the number of users is 500, our algorithm takes 0.96s, while MoRule and MobiFlow take 0.14s and 0.40s.

Note that the execution time of our algorithms is always within 1 s, which is practically applicable for mission-critical applications[15] and typical mobile users (switching to a new small cell every 11-50 s [11]). Moreover, in SDMENs, SDN controllers are deployed in a distributed manner to manage the entire network, with each controller only responsible for a small area. Furthermore, our algorithm is responsible for network slices that contain mission-critical applications where users are small in number, especially in small areas. Finally, the computing power of the server in the edge data center is more powerful, and the algorithm can also be accelerated by multi-core parallel computing, hardware acceleration, distributed computing, etc.

C. Experimental Result

1) End-to-End Latency: To evaluate the impact of the FSHR on the end-to-end latency, we test our algorithm in our mininet-WiFi/RYU-based testbed. It should be noted that in the experiment, the default node TCAM and link bandwidth are still adopted. In this experiment, we tested the algorithms we designed under default routing and dynamic routing. In the default routing case, the average FSHR of the PFS-DF, Mobi-Flow, MoRule, and Reactive algorithms are 73.2%, 57.2%, 51.0%, and 40.6%, which corresponds to the simulation result with 500 users as shown in Fig. 6(a). For the dynamic routing case, the average FSHR of these four algorithms are 73.2%, 57%, 50.8%, and 40.8%, which corresponds to the result with 500 users in Fig. 6(c). With reference to [13], we use *ping*[16] to generate 500 flows instantaneously to test the end-to-end latency, where the source and destination of the flow are randomly generated. The tested topology is Star, and the experimental results are shown in Fig. 9.

As shown in Fig. 9(a), in the default routing case, 73.2% of flows in our algorithm have a latency of less than 1 ms, while 42.8%, 49.0%, and 59.4% of the flows in Mobi-Flow, MoRule, Reactive algorithms have a latency of more than 10.49, 10.86 and 10.77 ms. This is because our algorithm has a higher FSHR compared with other algorithms, and more flows are proactively set up with flow rules. When a user accesses a new small cell, its service flow can be directly forwarded without introducing additional flow setup latency, which greatly reduces the end-to-end latency. The experimental results further verify the superiority of our algorithm.

Similar results can be found in the dynamic routing case, as shown in Fig. 9(b). In our algorithm, 73.2% of the flows have a latency of less than 1 ms, while in Mobi-Flow, MoRule, and Reactive algorithms, 43.0%, 49.2%, and 59.2% of the flows have a latency exceeding 20.08, 23.04, and 21.74 ms. The experimental results show that the algorithms have a longer end-to-end latency under the dynamic routing strategy. This is because, under the dynamic routing strategy, the SDN controller needs to set flow rules for all network elements on the routing path of each flow. In this case, more network elements need to be set flow rules, resulting in a longer flow setup latency, thereby prolonging the end-to-end latency. The above experimental results verify that our algorithm can provide better performance for latency-sensitive services.

VIII. Conclusion

In SDMENs, user mobility introduces additional flow setup latency, which undermines the latency-sensitive applications. In this paper, we are the first to study how to proactively set up flow rules to maximize FSHR under limited bandwidth and TCAM resources, thereby eliminating flow setup latency as much as possible to improve the performance of latency-sensitive services. Considering the two typical routing strategies, we formally construct this problem as proactive flow setup problems with default routing and dynamic routing, aiming to maximize the total FSHR. These two problems are rigorously proved to be NP-hard, and FPTAS is ruled out. To tackle these two problems, we propose a polynomial-time approximation algorithm for the default routing case and an efficient heuristic algorithm for the dynamic routing case. They determine which small cells to proactively set up flow rules and their routes for each user's service flow. Finally, extensive trace-driven experimental and simulation results verify that our algorithm is close to the optimal solution and can significantly increase the FSHR and improve latency-sensitive services.

We saw that the proposed scheme is applicable to typical client-server communication applications. In the future,

[15]Because our algorithms are designed to proactively set flow rules, not reactively. Our algorithms attempt to set flow rules for flows before they arrive, which allows these flows to be forwarded directly without waiting for the flow setup process (including algorithm execution). Once a flow arrives and there is no corresponding flow rule, then the controller is reactively triggered to set flow rules for it with a low complexity reactive flow setup algorithm (not ours), which is done in about 10-100ms [27], [28].

[16]The ultra-low latency required by mission-critical applications refers to end-to-end latency, and *ping* is a typical command to test end-to-end latency, so this is reasonable. Besides, since each flow is allocated the required bandwidth, it is independent of the background traffic, which is critical to ensure the ultra-low latency of the service.

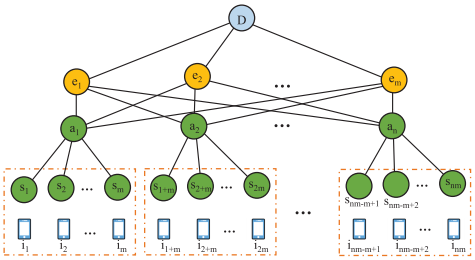


Fig. 10. A special case of PFSDF problem for NP-hard proof.

we plan to extend our research to client-to-client communication applications, where the source and destination have mobility, such as vehicle-to-vehicle communication.

APPENDIX A
PROOF OF THEOREM 1

Proof: The multiple knapsack problem can be described as follows: Given a set of n items and a set of m knapsacks, each item j has a weight w^j and a price τ^j, and each knapsack k has a capacity c^k. We need to determine which items to include in each knapsack so that the total value is as large as possible and does not exceed the capacity of each knapsack. Let the binary variable $y^{j,k}$ indicate whether to include item n in knapsack m. Then, the multiple knapsack problem can be formalized as

$$\begin{aligned} \max \ & \sum_{j=1}^{n}\sum_{k=1}^{m}\tau^j y^{j,k} \\ s.t. \ & \sum_{j=1}^{n} w^j y^{j,k} \le c^k, k \in \{1,2,\ldots,m\}, \\ & \sum_{k=1}^{m} y^{j,k} \le 1, j \in \{1,2,\ldots,n\}, \\ & y^{j,k} \in \{0,1\}, \forall j \in \{1,2,\ldots,n\}, k \in \{1,2,\ldots,m\}. \end{aligned} \tag{14}$$

We considered a topology as shown in Fig. 10. There are $n \cdot m$ users in the network, and they currently access small cell A (omitted in the figure for simplicity). The transition probability of each user i ($i \in \{i_1, i_2, \ldots, i_{mn}\}$) to small cell s ($s \in \{s_1, s_2, \ldots, s_{nm}\}$) and A are $\mathcal{P}^{i,s}$ and $1-\mathcal{P}^{i,s}$, respectively. The service flow of user i is $f^i = \{D, b^i\}$. The default routing path of small cell s_h ($h \in \{1,2,\ldots,nm\}$) to the edge server D is $s_h \rightharpoonup a_{(h-1)//m+1} \rightharpoonup e_{(h-1)\%m+1} \rightharpoonup D$. The available bandwidth on link (e_l, D) is $\mathcal{C}_{e_l,D}$, and the available TCAM on small cell s ($s \in \{a_1, a_2, \ldots, a_n\}$) is 1. The available bandwidth and TCAM resources on other nodes and links are ∞. Let $\tau^j = \mathcal{P}^{i_{h//m}, s_{h//m}}$, $w^j = b^{(j-1)\cdot m+k}, \mathcal{C}_{e_l,D} = c^k, j \in \{1,2,\ldots,n\}, k \in \{1,2,\ldots,m\}, l \in \{1,2,\ldots,m\}, h \in \{1,2,\ldots,nm\}$. We define $z^{j,k} = x^{i_{(j-1)\cdot m+k}, s_{(j-1)\cdot m+k}}$.

Then, PFSDF problem can be simplified to

$$\begin{aligned} \max & \sum_{j=1}^{n}\sum_{k=1}^{m}\tau^j z^{j,k} + \sum_{i\in\{i_1,i_2,\ldots,i_{nm}\}} \sum_{s\in\{s_1,s_2,\ldots,s_{nm}\}} (1-\mathcal{P}^{i,s}) \\ & \sum_{j=1}^{n} w^j z^{j,k} \le c^k, \forall k \in \{1,2,\ldots,m\}, \\ & \sum_{k=1}^{m} z^{j,k} \le 1, \forall j \in \{1,2,\ldots,n\}, \\ & z^{j,k} \in \{0,1\}, \forall j \in \{1,2,\ldots,n\}, k \in \{1,2,\ldots,m\}, \end{aligned} \tag{15}$$

where $\sum_{i\in\{i_1,i_2,\ldots,i_{nm}\}} \sum_{s\in\{s_1,s_2,\ldots,s_{nm}\}} (1-\mathcal{P}^{i,s})$ is a constant. Obviously, this problem (Eq. (15)) is equivalent to the multiple knapsack problem (Eq. (14)). Thus, PFSDF problem is NP-hard.

Since the multiple knapsack problem is an instance of PFSDF problem, and the multiple knapsack problem is proved to be strongly NP-hard [25], then PFSDF is also strongly NP-hard. Since $\sum_{s\in\mathbb{S}^i\cup\varsigma^i} \mathcal{P}^{i,s} = 1$, based on (2), we have $R^*_{PFSDR} \le |\mathbb{U}|$, where R^*_{PFSDR} denotes the optimal solution for PFSDF problem. Since PFSDF problem is both strongly NP-hard and its objective function has a polynomial boundary, it cannot have FPTAS unless P=NP [46]. ■

REFERENCES

[1] M. Ahmad et al., "A low latency and consistent cellular control plane," in *Proc. ACM SIGCOMM*, 2020, pp. 648–661.
[2] H. Bagheri et al., "5G NR-V2X: Toward connected and cooperative autonomous driving," *IEEE Commun. Standards Mag.*, vol. 5, no. 1, pp. 48–54, Mar. 2021.
[3] Z. Zhu et al., "Research and analysis of URLLC technology based on artificial intelligence," *IEEE Commun. Standards Mag.*, vol. 5, no. 2, pp. 37–43, Jun. 2021.
[4] *Service Requirements for the 5G System (Release 18)*, Technical Specification Group Services and System Aspects, document 3GPP TS 22.261, 2021.
[5] 5G-PPP. (2021). *5G KPIs*. [Online]. Available: https://5g-ppp.eu/kpis/
[6] Y. C. Hu, M. Patel, D. Sabella, N. Sprecher, and V. Young, "Mobile edge computing—A key technology towards 5G," *ETSI White Paper*, vol. 11, no. 11, pp. 1–16, 2015.
[7] S. Song, C. Lee, H. Cho, G. Lim, and J.-M. Chung, "Clustered virtualized network functions resource allocation based on context-aware grouping in 5G edge networks," *IEEE Trans. Mobile Comput.*, vol. 19, no. 5, pp. 1072–1083, May 2020.
[8] F. Z. Yousaf, M. Bredel, S. Schaller, and F. Schneider, "NFV and SDN—Key technology enablers for 5G networks," *IEEE J. Sel. Areas Commun.*, vol. 35, no. 11, pp. 2468–2478, Nov. 2017.
[9] ONF. (2021). *Software-Defined Networking (SDN) Definition*. [Online]. Available: https://www.opennetworking.org/sdn-definition/
[10] D. Xu et al., "Understanding operational 5G: A first measurement study on its coverage, performance and energy consumption," in *Proc. ACM SIGCOMM*, 2020, pp. 479–494.
[11] Y. Li, Q. Li, Z. Zhang, G. Baig, L. Qiu, and S. Lu, "Beyond 5G: Reliable extreme mobility management," in *Proc. ACM SIGCOMM*, 2020, pp. 344–358.
[12] J. Xie, D. Guo, X. Li, Y. Shen, and X. Jiang, "Cutting long-tail latency of routing response in software defined networks," *IEEE J. Sel. Areas Commun.*, vol. 36, no. 3, pp. 384–396, Mar. 2018.
[13] R. Khalili, Z. Despotovic, and A. Hecker, "Flow setup latency in SDN networks," *IEEE J. Sel. Areas Commun.*, vol. 36, no. 12, pp. 2631–2639, Dec. 2018.
[14] S. Bera, S. Misra, and N. Saha, "Traffic-aware dynamic controller assignment in SDN," *IEEE Trans. Commun.*, vol. 68, no. 7, pp. 4375–4382, Jul. 2020.
[15] G. Zhao, H. Xu, J. Fan, L. Huang, and C. Qiao, "Achieving fine-grained flow management through hybrid rule placement in SDNs," *IEEE Trans. Parallel Distrib. Syst.*, vol. 32, no. 3, pp. 728–742, Mar. 2021.
[16] H. Li, P. Li, and S. Guo, "MoRule: Optimized rule placement for mobile users in SDN-enabled access networks," in *Proc. IEEE Global Commun. Conf.*, Dec. 2014, pp. 4953–4958.
[17] Y. Kyung, "Mobility-aware prioritized flow rule placement in software-defined access networks," in *Proc. Int. Conf. Inf. Netw. (ICOIN)*, Jan. 2021, pp. 59–61.
[18] I. Maity, S. Misra, and C. Mandal, "CORE: Prediction-based control plane load reduction in software-defined IoT networks," *IEEE Trans. Commun.*, vol. 69, no. 3, pp. 1835–1844, Mar. 2021.

[19] S. Bera, S. Misra, and M. S. Obaidat, "Mobi-flow: Mobility-aware adaptive flow-rule placement in software-defined access network," *IEEE Trans. Mobile Comput.*, vol. 18, no. 8, pp. 1831–1842, Aug. 2019.
[20] B. Zhao, R. Li, J. Zhao, and T. Wolf, "Efficient and consistent TCAM updates," in *Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, Jul. 2020, pp. 1241–1250.
[21] I. Maity, S. Misra, and C. Mandal, "DART: Data plane load reduction for traffic flow migration in SDN," *IEEE Trans. Commun.*, vol. 69, no. 3, pp. 1765–1774, Mar. 2021.
[22] P. Wette and H. Karl, "Which flows are hiding behind my wildcard rule? Adding packet sampling to OpenFlow," in *Proc. ACM SIGCOMM*, 2013, pp. 541–542.
[23] X. Yang, H. Xu, X. Yu, C. Qian, G. Zhao, and H. Huang, "Cooperative flow statistics collection with per-switch cost constraint in SDNs," *IEEE Trans. Commun.*, vol. 69, no. 8, pp. 5325–5336, Aug. 2021.
[24] X. Jin et al., "Dynamic scheduling of network updates," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 4, pp. 539–550, 2014.
[25] H. Kellerer, U. Pferschy, and D. Pisinger, *Knapsack Problems*. USA: Springer, 2004, pp. 235–283.
[26] H. Gebrie, H. Farooq, and A. Imran, "What machine learning predictor performs best for mobility prediction in cellular networks?" in *Proc. IEEE Int. Conf. Commun. Workshops (ICC Workshops)*, May 2019, pp. 1–6.
[27] H. K. Ravuri, M. T. Vega, T. Wauters, B. Da, A. Clemm, and F. De Turck, "An experimental evaluation of flow setup latency in distributed software defined networks," in *Proc. IEEE Conf. Netw. Softwarization (NetSoft)*, Jun. 2019, pp. 432–437.
[28] A. Panda, W. Zheng, X. Hu, A. Krishnamurthy, and S. Shenker, "SCL: Simplifying distributed SDN control planes," in *Proc. NSDI*, 2017, pp. 329–345.
[29] M. Series, *IMT Vision–Framework and Overall Objectives of the Future Development of IMT for 2020 and Beyond*, Recommendation ITU, document 2083, Sep. 2015.
[30] L. Song, U. Deshpande, U. C. Kozat, D. Kotz, and R. Jain, "Predictability of WLAN mobility and its effects on bandwidth provisioning," in *Proc. 25th IEEE Int. Conf. Comput. Commun. (INFOCOM)*, Apr. 2006, pp. 8–14.
[31] (2021). *Cellular Network*. [Online]. Available: https://en.wikipedia.org/wiki/Cellular_network
[32] L. G. Khachiyan, "A polynomial algorithm in linear programming," in *Doklady Akademii Nauk*, vol. 244, no. 5. Moscow, Russia: Russian Academy of Sciences, 1979, pp. 1093–1096.
[33] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*. Cambridge, MA, USA: MIT Press, 2001.
[34] (2021). *Network Topology*. [Online]. Available: https://www.omnisci.com/technical-glossary/network-topology
[35] "Transport network support of IMT-2020/5G," ITU, Tech. Rep. GSTR-TN5G, 2018.
[36] NTT. (2021). *RYU Network Operating System*. [Online]. Available: http://osrg.github.com/ryu/
[37] R. D. R. Fontes and C. E. Rothenberg, "MiniNet-WiFi: A platform for hybrid physical-virtual software-defined wireless networking research," in *Proc. ACM SIGCOMM*, 2016, pp. 607–608.
[38] B. Lantz, B. Heller, and N. McKeown, "A network in a laptop: Rapid prototyping for software-defined networks," in *Proc. ACM SIGCOMM*, 2010, pp. 1–6.
[39] A. Marotta et al., "Impact of CoMP VNF placement on 5G coordinated scheduling performance," in *Proc. EuCNC*, 2017, pp. 1–6.
[40] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," in *Proc. ACM SIGCOMM*, 2008, pp. 63–74.
[41] P. Michal, S.-D. Natasa, and G. Matthias. (2021). *Taxi Trace*. [Online]. Available: https://crawdad.org/epfl/mobility/20090224
[42] S. Uhlig, B. Quoitin, J. Lepropre, and S. Balon, "Providing public intradomain traffic matrices to the research community," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 36, no. 1, pp. 83–86, 2006.
[43] I. Maity, A. Mondal, S. Misra, and C. Mandal, "CURE: Consistent update with redundancy reduction in SDN," *IEEE Trans. Commun.*, vol. 66, no. 9, pp. 3974–3981, Sep. 2018.
[44] (2022). *Vehicle Abundance: Countries Compared*. [Online]. Available: https://www.nationmaster.com/country-info/stats/Transport/Vehicle-abundance#date
[45] (2021). *Gurobi*. [Online]. Available: https://www.gurobi.com/
[46] V. V. Vazirani, *Approximation Algorithms*. Berlin, Germany: Springer, 2001.