Grouping Service Chains of Multiple Flows in NFV-based Networks

Yang Chen, Jie Wu, and Rajorshi Biswas
Center for Networked Computing, Temple University, USA
Email: {yang.chen, jiewu, rajorshi}@temple.edu

***Abstract*—In Software Defined Networks (SDNs), flows usually request to be processed by a service chain (an ordered set of virtualized network services). SDN-enabled networks manage the routing and processing of flows by a large number of associated finer-granularity rules. These rules are maintained by switches in their local hardware such as Ternary Content Addressable Memories (TCAMs), which support high-speed parallel lookup on wildcard patterns. However, the capacity of hardware switches is limited to thousands because of their high requirements for cost and power. In order to avoid much slower matching by using software switches or even packet loss, we can group flows so that all matching rules can be placed in hardware switches. In such a grouping, all flows in each group match only one rule and will be forwarded to the same routing path, instead of each flow matching one rule. This will result in a longer delay because of processing by the longer newly-grouped service chain. In this paper, we efficiently group flows to minimize the total cost while satisfying the capacity constraint of the forwarding tables in hardware switches. We first prove the submodularity of our objective function and propose a corresponding performance-guaranteed solution. Additionally, we design an efficient heuristic solution based on the classic k-means algorithm. Furthermore, we include discussions on dynamic network situations (insertion, deletion, and update of flows) and an alternative objective. We also conduct real experiments on our testbed to indicate the practicality of our motivation. Extensive simulations are conducted to evaluate the performance of our proposed algorithms.**

***Index Terms*—TCAM, grouping, rule, submodular, k-means.**

I. Introduction

The role of modern networks has transformed to provide various types of network services (such as security, performance optimization, cross-protocol inter-operability, and value-added services) beyond providing basic connectivity services. Nowadays, network services usually requires multiple network functions, also called middleboxes, to be chained together in some order, which is known as the service chain [1, 2]. For example, in operator networks [3], data centers [4, 5], mobile networks [6], and enterprise networks [7], network operators often require traffic to pass the service chain: Firewall, IDS, and proxy in the sequence order [8–10]. Network Function Virtualization (NFV) helps in implementing network functions on regular hardwares as software middleboxes, which are deployed at switch-connected servers [11, 12]. These middleboxes add tags in packet headers to track their processing states. The tagging scheme is lightweight and effective even without switch modification [13]. In the flow table at each switch, forwarding rules define how to process the received flows based on requested policies [14]. In

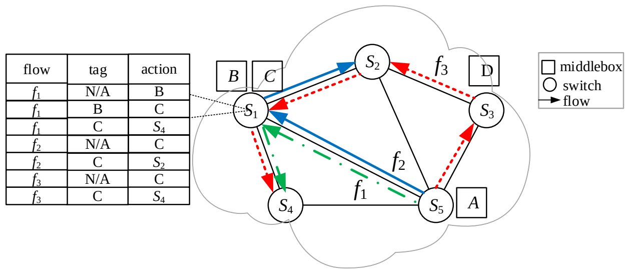

flow	tag	action
f_1	N/A	B
f_1	B	C
f_1	C	S_4
f_2	N/A	C
f_2	C	S_2
f_3	N/A	C
f_3	C	S_4

Fig. 1: An illustrating example (f_1 requires $A \rightarrow B \rightarrow C$, f_2 requires $A \rightarrow C$, and f_3 requires $A \rightarrow D \rightarrow C$).

order to enforce service chains with these tag-augmented rules, there are more forwarding actions (to switches or middleboxes deployed at its connected server), which dramatically enlarges the number of forwarding rules in switches.

For modern switches, there are two types: hardware-based and software-based switches [15]. In hardware switches, Ternary Content Addressable Memory (TCAM) enables fast lookups with flexible wildcard rule patterns. The cost and power requirements of TCAM limit the number of rules that a hardware switch can support. In general, a TCAM enabled switch can store only 5000 rules [16]. Software switches form a classic cache and main memory pair that stores and processes rules together. They have a large rule capacity but match rules much more slowly than hardware ones, which incurs a longer delay by a higher magnitude. In a Software Defined Network (SDN) that has a large number of active flows, SDN allows control applications to install forwarding policies for routing flows in a finer granularity at underlying switches [17], resulting in flow tables at TCAM hardware being easily overflowed. What makes things worse is that SDN also enables traffic routing optimization in terms of device costs, total throughput, load balancing, etc., and at the same time satisfies correct traversal of service chain for each flow, which results in a flooding number of necessary rules that need to be installed in switches. In order to reduce the network overall delay, efficiently utilizing the forwarding table entries in TCAM hardware becomes more important [18].

In this paper, we propose to control the number of forwarding rules using *service chain grouping*. Service chain grouping combines selected service chains in a directed graph and performs topological sorting [19] to generate a new service chain. All corresponding flows in each group serve as a new flow and get processed by the same newly-grouped service chain, which reduces the number of forwarding rules in a coarser routing granularity. We can apply the tagging technology to tag each packet's header for all flows in one group with a unique rule ID [20]. The tagging scheme is lightweight and augments only tens lines of code to middleboxes [13]. The hardware does not need to be changed, either. Then we use the ID as one attribute for matching the forwarding rules in switches. The grouping tagging action will be done once and only once when the flow comes out of source. After that, all flows in the same group act as a new flow and the rule matching will have no difference without the grouping. So the overhead is low. However, grouping flows together generates longer service chains, which will incur a larger overall transmission delay. The tag attached to the packets of a flow also indicates whether the packets will be processed by a NF or not. For example, if a packet is not tagged with decrypter but it travels the decrypter, it will not be processed by decrypter. However, because of increased hop the delay will increase little bit. It is a trade of between the delay increased by overloaded switch and increased hops. We have added this clear explanation in this revision. If there are more flows to be grouped together as one flow, we will have fewer flows, which means fewer rules in switches but longer newly-grouped service chains, resulting in a larger overall flow transmission delay. Then there is a trade-off between the total number of service chains (or groups) and the average length of service chains in newly generated groups, also the overall end-to-end delay of all flows.

We illustrate our problem in a toy network with five switches (circles) $S_1, S_2, S_3, S_4,$ and S_5 in Fig. 1. Four kinds of middleboxes $A, B, C,$ and D, shown in squares, are placed in the network. We use $\rightarrow$ to represent the dependency relation between two middleboxes in a service chain. For example, $A \rightarrow B$ means A needs to be processed before B. There are three flows $f_1, f_2,$ and f_3 in the network, each of whose path is in a different shaped, directed line. f_1 requires the service chain of $A \rightarrow B \rightarrow C$, f_2 requires the service chain of $A \rightarrow C$, and f_3 requires the service chain of $A \rightarrow D \rightarrow C$. Then we use the forwarding table of switch S_1 as a motivating example. According to the service chain requests, f_1 needs to be processed by middleboxes B and C while f_2 and f_3 only need to be processed by middlebox C at S_1. There are 7 rules in S_1 to implement the forwarding policies of all flows, which include 3 for f_1, 2 for f_2, and 2 for f_3 in order to differentiate each state of each flow. However, if we merge the service chains of f_1 and f_3 and serve them as a new flow f_4, the number of rules is reduced to 5. f_1 and f_3 use the same 3 rule entries in the forwarding table of S_1, but at the expense of f_3 being processed by an extra middlebox B. Both their service chains are extended to $A \rightarrow D \rightarrow B \rightarrow C$, resulting in a larger end-to-end delay. If we group all their service chains into one, the total number of rules will be reduced to 4. This indicates the tradeoff between the number of forwarding rules and the average length of service chains.

In this paper, we aim at efficiently grouping service chains with the minimum total cost (traffic rate times service chain length) while the total number of service chains is limited

$f_1(3)$: A→B→C
$f_2(2)$: A→C→D
$f_3(1)$: A→D→E→F
$f_4(6)$: B→F→G
$f_5(4)$: C→E

(a) Original service chains.

$f_1+f_2+f_3(6)$:
A→B→C→D→E→F
$f_4(6)$: B→F→G
$f_5(4)$: C→E

(b) Optimal grouping to 3 chains.

$f_1+f_2+f_3+f_5(10)$:
A→B→C→D→E→F
$f_4(6)$:
B→F→G

(c) Optimal grouping to 2 chains.

$f_1+f_4(9)$:
A→B→C→F→G
$f_2+f_3+f_5(7)$:
A→C→D→E→F

(d) Another grouping to 2 chains.

Fig. 2: A motivating example, where $f(r)$ represents a flow f with the traffic rate r.

to the capacity of forwarding tables in TCAMs. We first formulate our problem mathematically. Then we prove the submodularity of our cost function and propose a corresponding solution with an approximation ratio. Besides that, we adjust the k-means method to solve our problem with intuitive insights. Additionally, we include discussions about service chain updates, load balancing, and the extended model for actual end-to-end delay. We use real-world data to verify the trade-off between the total number of service chains and the actual end-to-end delay, where the number of rules can exceed the capacity of TCAMs. Extensive simulations are conducted to evaluate the performance of our proposed algorithms in various scenarios.

The remainder of this paper is organized as follows. Section II describes the model and formulates the problem. Section III introduces our algorithms based on the submodular property of our objective function. In Section IV, we provide alternative solutions based on the k-means algorithm. We include several discussions in Section V. Section VI includes the experiments. Section VII surveys related works. Finally, Section VII concludes the paper.

II. Model and Formulation

In this section, we talk about our motivation, network model, problem formulation, and the problem hardness.

A. Motivation

SDN enables a centralized controller to install fine-grained forwarding rules into the forwarding table of each underlying switches in data plane, consisting of hardware switches and software switches. The Ternary Content Addressable Memory (TCAM) allows fast lookups with flexible wildcard rule patterns. However, the cost and power requirements limit the number of rules that the hardware switches can support. In other words, the capacity of forwarding tables is quite restricted. Software switches form a classic cache and main memory pair that stores and processes rules together. They have a large rule capacity but match rules much more slowly than hardware ones, which incurs a longer delay by a higher magnitude. In this paper, we propose to reduce the number of forwarding entries, restricted by the capacity of TCAM switches, by grouping service chains in order to avoid this kind of performance degradation.

We use a motivating example, shown in Fig. 2, to illustrate the service chain grouping with different constraints. We are given five flows f_1, f_2, f_3, f_4, and f_5, whose traffic rates are 3, 2, 1, 6, and 4, respectively. Squares with different letters are different kinds of middleboxes. We use directed arrows to represent the dependency relationships (processing order) among middleboxes. The required service chain by each flow is shown in Fig. 2(a). For example, f_1 needs to be processed by middlebox A first, then middlebox B, and finally middlebox C. Our objective is to minimize the total cost, which is the sum of each grouped service chain length times the total traffic rate to be processed by that service chain. If we are only allowed to have three chains, the optimal grouping strategy is shown in Fig. 2(b). The service chains of f_1, f_2, and f_3 are merged into one grouped service chain by using one topological order of all middleboxes based on their dependency relations in each chain. The length of the newly grouped chain is 6 and the total traffic rate of these three flows is $3+2+1=6$. Then the cost of the newly grouped service chain is $6 \cdot 6 = 36$. The service chains of the other two flows do not need to be merged and remain as a single service chain in the final result. The total cost is $6 \cdot 6 + 3 \cdot 6 + 2 \cdot 4 = 62$.

If we are only allowed to have two chains, the optimal grouping strategy is shown in Fig. 2(c). The total cost is $3 \cdot 6 + 6 \cdot 10 = 78$, which is larger than that with three chains. This is because with a smaller number of service chains, more flows need to be assigned to one group with a more likely longer service chain. It makes the cost for each flow increase, resulting in a larger total cost. Fig. 2(d) shows another grouping strategy with a total cost of $9 \cdot 5 + 7 \cdot 5 = 80$, which is larger than that of Fig. 2(c). It is not optimal for our objective of the total cost. However, the length of its longest grouped chain is 5, which is shorter than that in Fig. 2(c). Additionally, its largest total traffic rate of all grouped service chains is 9, while the value is 10 in Fig. 2(c). As a result, Fig. 2(d) has a more balancing distribution of traffic rate, making it less likely to violate the link capacity constraint. This illustrates that the grouping problem is not trivial with different constraints and objectives.

B. Network Model

We are given a set of unsplittable flows $F = \{f\}$ because flow splitting may not be feasible for applications that are sensitive to TCP packet ordering (e.g. video applications). Note that split flows can be treated as multiple unsplittable flows. For the network function load balancer, it usally redistributes the traffic load of one flow into different routes, where we directly treat the flow as several independent flows from the source. We use f to denote a single flow that has an initial traffic rate of r_f, and a requested service chain of C_f, which is an ordered set of middleboxes. We reduce the number of forwarding rules by service chain grouping. We need to group several requested service chains into one as a newly grouped

Symbols	Definitions
f, F	a flow and the set of all flows
r_f, C_f	f's traffic rate and its requested service chain
G, Ω	a service chain after grouping and the set of all G
X_f^G	indicator of service chain C_f assigned to G
cost(G)	the cost function of a service chain G

TABLE I: Symbols and Definitions.

service chain G. Here we introduce the definition of service chain grouping.

***Definition** 1: Service chain grouping*: the grouping procedure of multiple service chains forms a new service chain including all distinct middleboxes of these chains, which are sorted in a topological order.

We define G as a service chain after grouping, which is either a newly grouped service chain or an original requested service chain that is not grouped with other chains. The length of a service chain G is denoted as $|G|$, which is the total number of middleboxes in G. (We use $|\cdot|$ to denote the cardinality of a set.) The union set of all service chains after grouping is defined as Ω, i.e., $\Omega = \{G\}$. We introduce an indicator value X_f^G to indicate whether C_f is assigned (grouped) to G. If yes, $X_f^G = 1$; otherwise, $X_f^G = 0$. The cost of a service chain G, cost(G), is defined as follows:

***Definition** 2: Cost function*: is the length of the newly-grouped service chain G times the total traffic rate of each flow f that is assigned to the group chain G, i.e., $X_f^G = 1$, which satisfies $\text{cost}(G) = |G| \cdot \sum_{X_f^G=1} r_f$.

We are given a pre-determined parameter k as the allowed total number of service chains after the service chain grouping. This constraint comes from the capacity of the forwarding table entries in hardware switches. The value of k can be slightly larger than the capacity of forwarding tables because TCAMs can reduce the number of rules by wildcard matching. The value can be adjusted in the real network environment. If the value of k is much higher than the capacity, then because of the overwhelming flow table, the matching takes a longer time and the delay increases dramatically. This parameter limits the total number of service chains after grouping to no more than k, meaning $|\Omega| \leq k$. Here we need to mention that the deployment issue of multiple VNF instances of the same type is out of scope of this paper, which has been studied by a lot of research [21].

C. Problem Hardness

***Theorem** 1:* It is NP-hard to check whether there is a feasible service chain grouping with a constraint that the total number of service chains is no more than k.

Proof: First, we draw the dependency graph, denoted as P, of all requested service chains $\{C_f | \forall f \in F\}$. If there are no cycles or no more than k cycles in the generated directed graph, the service chain grouping is definitely feasible because we can break each cycle and make each into a separate new service chain, resulting in a total number no more than k. If there are more than k cycles or cycles that intertwine with each other, we prove that it is NP-hard to check its feasibility. Simply speaking, it is NP-hard to check whether it is possible to break all cycles by cutting off no more than k edges [22]. Specifically, we can apply the method proposed in [23] to find all Strong Connected Components (SCCs). Suppose there is one SCC in the directed graph P and the general collection of cycles (elements) is U. In total, N service chains involve in P and the i^{th} service chain ($i = 1, ..., n$) is able to break the set $S_i \subseteq U$. The goal is to break all the cycles by removing $k-1$ service chains. We consider each cycle is an element in the set cover problem. The union set is the set of all cycles. Each set in the set cover problem is the set of all the cycles removing one service chain that can break the cycle. So we reduce the original problem to the so-called set cover, an NP-hard problem that covers all elements by selecting $k-1$ sets. ■

Since SDN rules match with packets against specified header fields, forwarding ambiguity arises when a packet goes through a switch more than once, each time toward a different next hop. Flows always ask for network function service, which further enlarges the number of rules dramatically. Network functions (NFs) such as firewalls, deep packet inspection, content caches, WAN load-balancers, etc. are provided by specialized network devices called middleboxes. They are widely deployed in various networking scenarios including broadband access networks, enterprise networks, backbone networks, data centers, and cloud computing environments [24]. Typically, network flows go through several NFs in a specific order to meet the required processing. This forwarding ambiguity is further exacerbated by middleboxes like NAT. Without their modification strategies as a priori, it is hard to pre-configure rules for modified packets. Stateful processing against forwarding ambiguity requires middleboxes to tag packet headers [13]. The tagging scheme is lightweight and augments only tens lines of code to middleboxes. With the limited number of the forwarding table in a switch, we have to efficiently utilize each rule in order to route more flows. We also implement a motivating experiment in a real testbed in order to show that the matching speed increases a lot when the number of rules exceeds the capacity of the forwarding table in a switch.

D. Problem Formulation

Based on the above network model, we formulate our service chain grouping problem in this paper as an optimization problem with an objective of minimizing the total cost of all service chains after grouping as follows:

$$\min \quad \sum\nolimits_{G\in\Omega} \text{cost}(G) \tag{1}$$

$$\text{s.t.} \quad \text{cost}(G) = |G| \cdot \sum\nolimits_{X_f^G=1} r_f \tag{2}$$

$$|\Omega| \leq k \tag{3}$$

$$C_f \subseteq G \qquad \text{if } X_f^G = 1,\ \forall f \in F \tag{4}$$

$$\sum\nolimits_{G\in\Omega} X_f^G = 1 \qquad \forall f \in F \tag{5}$$

$$X_f^G = \{0,1\} \qquad \forall f \in F, G \in \Omega \tag{6}$$

Eq. (1) is our objective: minimizing the total cost of all service chains after grouping. In Eq. (2), the cost of each service chain G equals the chain length times the sum of all assigned flows' traffic rates, whose requested service chain is grouped to G. Eq. (3) ensures the total number of service chains after grouping is no more than the given threshold k. Eq. (4) states that a flow f's requested service chain C_f is a subset of its assigned service chain G when $X_f^G = 1$. Eq. (5) requires that each flow f's requested service chain be assigned to one and only one service chain after grouping. Eq. (6) shows that the indicator value X_f^G can only be 0 or 1, $\forall f \in F$.

III. Solutions

In this section, we propose two different solutions. Although the general service chain grouping problem is proven NP-hard, in the real world, the length of a requested service chain is always less than 10 and the orders among different middleboxes are always fixed [25]. For example, an IPSec decryptor is usually placed before a NAT gateway [26]. It illustrates that the service grouping is usually feasible with an appropriate value of k.

A. Submodular Solution

In this subsection, we first prove the submodular property of our objective function and then we propose one corresponding solution. We prove that this solution has an approximation ratio for the case without the requirement of load balancing. Before proposing our solution, we introduce an extra definition, called *marginal increment*, in order to prove the submodularity of the cost function of a service chain, cost(G).

***Definition** 3 (marginal increment):* The marginal increment, denoted as $\Delta\text{cost}_G(f) = \text{cost}(G \cup C_f) - \text{cost}(G)$, indicates the increased cost when the service chain C_f of flow f is assigned into the service chain G.

Next, we analyze the property of the cost function cost(G). A function f is *submodular* if and only if $\forall S \subseteq T \subseteq N, \forall e \in N \backslash T$, $f_T(e) \leq f_S(e)$. Then we prove that $\forall f \in F$, if $X_f^G = 0$ and $G \subseteq G'$, the submodular property holds, i.e., cost$(G \cup C_f) - \text{cost}(G) \geq \text{cost}(G' \cup C_f) - \text{cost}(G')$.

***Theorem** 2:* cost(G) is a submodular function.

Proof: It is intuitive that cost(G) is a non-decreasing function, which is monotone. Suppose there are two deployments G and G' with $G \subseteq G'$. It is intuitive that G' includes all middleboxes in G and it also satisfies the dependency relations of all middleboxes in G. As long as the service chain length increases or remains in G', it will process no less than the total traffic rate in G. Then we have cost$(G \cup C_f) - \text{cost}(G) \geq \text{cost}(G' \cup C_f) - \text{cost}(G')$. Thus, cost$(G)$ is submodular. ■

Then we solve our grouping problem by proposing our algorithm in Alg. 1, called Submodular Greedy Solution (SGS). Line 1 calculates the cost function value of each service chain. In line 2, we select the k largest values and assign the corresponding service chain into Ω as a single service chain G after grouping. Line 3-5 merge the service chain of an unassigned flow f into one service chain $G \in \Omega$ with the smallest marginal increment $\Delta\text{cost}_G(f)$ in every loop, until

Algorithm 1 Submodular Greedy Solution (SGS)

```
In: The set of flows $F$ and threshold $k$;
Out: The service chain set $\Omega$ after grouping;
1: Calculate all flows' cost function value;
2: Select top $k$ original requested service chains with the largest cost into $\Omega$;
3: while not all flows have been assigned do
4:     Assign one unassigned flow $f$ into one service chain $G \in \Omega$ with the smallest marginal increment $\Delta\text{cost}_G(f)$;
5:     Update $G = G \cup C_f$ and $X_f^G = 1$.
6: return The service chain set $\Omega$ after grouping.
```

all flows are assigned. Line 7 returns the service chain set Ω after grouping. The insight of Alg. 1 is to solve the solution based on the submodularity of our objective function.

For better understanding, we use the same setting in Fig. 2(a) to apply Alg. 1 as an example. Suppose $k = 3$. The cost function value of the requested service chain of f_1 is $|C_1|\cdot r_1 = 3\cdot 3 = 9$. Similarly, the cost function values of all service chain are 9, 6, 4, 18, and 8, respectively. The top 3 largest values are 9, 18, and 8. So the requested service chains of f_1, f_4, and f_5 are assigned as a single service chain in $\Omega = \{G_1 = C_1, G_2 = C_4, G_3 = C_5\}$. Then we calculate the marginal increment of the unassigned flows f_2 and f_3. Here we take the f_2 as an example. If its service chain is grouped with the service chain of f_1, the service chain is updated as $\{A \rightarrow B \rightarrow C \rightarrow D\}$. The marginal increment is calculated as $4\cdot(2+3)-3\cdot 3 = 11$. Similarly, the marginal increments of f_2 with each service chain in Ω are 11, 30, and 16, respectively. 11 is the smallest, so we assign C_2 into C_1 and update $G_1 = \{A \rightarrow B \rightarrow C \rightarrow D\}$. Similarly, the smallest marginal increment of C_3 is $6\cdot(5+1)-4\cdot 5 = 16$ with G_1, so we assign C_3 to G_1 and update $G_1 = \{A \rightarrow B \rightarrow C \rightarrow D \rightarrow E \rightarrow F\}$. Then the returned service chain set is the same as that in Fig. 2(b). Luckily, the returned set is optimal.

Theorem 3: The total cost of Alg. 1 is at most $1 + 1/e$ times of that of the optimal solution.

Proof: Based on the classic set cover submodular problem [27], the approximation ratio is $1+1/e$. Detailed explanations are omitted here. ■

Time complexity: The time complexity of our proposed Alg. 1 is $O(|F|\cdot\max\{k, \log|F|\})$. Initially, it takes $O(|F|)$ time to calculate the cost function values of all flows in line 1. Line 2 takes $O(|F|\log|F|)$ to sort all the values and constant time to select the top k largest values. Then there are $|F|-k$ loops in lines 3-5. In each loop, it takes $O(k)$ to calculate the marginal increment with each service chain in Ω and $O(k)$ time to find the smallest marginal increment among k service chains in Ω. Then the total time complexity is $O(|F| + |F|\log|F| + (|F|-k)\cdot k)$. As $k \le |F|$, the time complexity is $O(|F|\cdot\max\{k, \log|F|\})$.

Algorithm 2 Greedy k-means Method (GKM)

```
In: The set of flows $F$ and threshold $k$;
Out: The service chain set $\Omega$ after grouping;
1: Randomly select $k$ service chains into $\Omega$;
2: for every flow $f$ with $\sum_{G\in\Omega} X_f^G = 0$ do
3:     Assign the flow $f$ into the service chain $G \in \Omega$ with the smallest length increment of the service chain after grouping;
4:     Update $G = G \cup C_f$ and $X_f^G = 1$.
5: return The service chain set $\Omega$ after grouping.
```

B. Greedy Solution

In this subsection, we apply the k-means method [28] to solve our grouping problem and propose the algorithm in Alg. 2, called Greedy k-means Method (GKM). In line 1, we initiate the grouped service chain set Ω by randomly selecting k service chains of flows from F. Lines 2-3 merge one service chain of a flow f with the smallest length increment every loop, until the total number of chains in Ω is no more than k. Line 4 updates the service chain information. Line 5 returns the grouped chain set Ω. The insight of Alg. 2 is to apply the flow management technology in SDN to group flows based on the similarity among their requested service chains.

For better understanding, we also use the same setting in Fig. 2(a) to apply Alg. 2 as an example. Suppose $k = 3$. Suppose $\Omega = \{G_1 = C_1, G_2 = C_2, G_3 = C_3\}$ after random selection. Then for flow f_4, the length increments with G_1, G_2, and G_3 are 2, 3, and 2. Then we can assign C_4 to G_1. Similarly, we assign C_5 to G_2. Then the returned service chain set is $\{G_1 = \{A \rightarrow B \rightarrow C \rightarrow F \rightarrow G\}, G_2 = \{A \rightarrow C \rightarrow D \rightarrow E\}, G_3 = C_3\}$. The total cost of the set is $5\cdot(3+6)+4\cdot(2+4)+4\cdot 1 = 73$, which is much larger than that of the optimal solution. If $\Omega = \{G_1 = C_1, G_2 = C_4, G_3 = C_5\}$ after random selection, the returned service chain set is the same as that in Fig. 2(b), which is optimal. It illustrates that the initial random selection of Ω matters a lot.

Time complexity: The time complexity of Alg. 2 is $O(|F|k)$. Line 1 takes $O(k)$ time to initialize Ω. Lines 2-4 have $O(|F|-k)$ loops. In each loop, it takes $O(k)$ time to calculate the length increment and $O(k)$ time to find the smallest length. Then the total time complexity of our proposed Alg. 2 is $O(k+(|F|-k)\cdot k) = O(|F|k)$.

IV. Some Discussions

In this section, we include some discussion to make our work more comprehensive. First, we consider the dynamic network situations including insertion, deletion and update of service chains. Next, we consider an alternative objective of load balancing among service chains and propose an efficient solution. Last but not least, we conduct real experiments on our testbed to show the practicality of our objective function.

Algorithm 3 Updating Grouping Strategy (UGS)

```
In: Sets of vertices $V$, edges $E$, original flows $F$, current chains $\Omega$, threshold $k$, and changed flow $f$;
Out: The grouping chain set $\Omega$;
1: if $f$ is a newly-arrived flow then
2:     Assign $f$ to the chain with the minimum $\Delta\text{cost}_G(f)$;
3: else if $f$ finishes its transmission then
4:     Delete $f$'s private middleboxes and update the grouped chain set;
5:     if $|\Omega| \leq k$ then
6:         Select the service chain with the largest cost value and assign it as a new grouped chain;
7: else if $C_f$ has been changed then
8:     Delete the previous $C_f$ from $\Omega$ and reassign the new $C_f$ to the chain with the minimum $\Delta\text{cost}_G(f)$;
9: Update $\Omega$;
10: return The updated service chain set $\Omega$.
```

A. Insertion, Deletion and Update

To maximize the data center network utilization, the SDN control plane needs to frequently update the data plane via flow changing as the network conditions change dynamically [29]. With the development of SDNs [30], there are many causes for a network update: (1) changes in security policies [31] (e.g., traffic from one sub-network may have to be rerouted via a firewall before entering another sub-network); (2) traffic engineering in the network [32] (to minimize the maximal link load, an operator may decide to reroute parts of the traffic along different links); (3) network maintenance works [33] (e.g., in order to replace a faulty router, it may be necessary to temporarily reroute traffic); and (4) reactions to link failures [34] (e.g., fast network update mechanisms are required to react quickly to link failures and determine a failover path). When network update happens, there is always rule changing. In order to handle flows in an online manner, we also generate solutions for inserting, deleting, and updating the service chains in the grouped set Ω. Here we need to mention that the insertion and deletion of service chains can be treated as two special cases of the update case, which are also included in the solution for service chain update. All the following proposed algorithms are extended from the above Alg. 1 based on the submodularity of our objective function.

For the service chain update case, we propose our algorithm in Alg. 3, called Updating Grouping Strategy (UGS). Lines 1-2 handle the service chain insertion case if there is a newly-arrived flow. We assign the new flow f to the service chain with the minimum marginal increment. Lines 3-6 handle the service chain deletion case if a flow finishes its transmission. We delete the unused middleboxes in the grouped service chain in order to shorten the chain. If the deleted service chain serves as one grouped chain in Ω, then after deletion, we are allowed to delete one grouped chain with the largest cost value and assign it to one new chain in Ω alone. Lines 7-8 handle the service chain update case if the requested service chain of a

(a) Updated service chains.	(b) Optimal grouping to 3 chains.
$f_1(3)$: A→B→C	$f_1+f_3(4)$: A→B→C→D→E→F
$f_2(6)$: A→C→E	
$f_3(1)$: A→D→E→F	$f_4(6)$: B→F→G
$f_4(6)$: B→F→G	
$f_5(4)$: C→E	$f_2+f_5(10)$: A→C→E

Fig. 3: Update procedure.

(a) Inserted service chains.	(b) Optimal grouping to 3 chains.
$f_1(3)$: A→B→C	$f_1+f_2+f_3(6)$: A→B→C→D→E→F
$f_2(2)$: A→C→D	
$f_3(1)$: A→D→E→F	
$f_4(6)$: B→F→G	$f_4+f_6(10)$: B→C→F→G
$f_5(4)$: C→E	
$f_6(4)$: B→C→F→G	$f_5(4)$: C→E

Fig. 4: Insertion procedure.

flow is changed. Line 9 updates the grouped service chain set Ω. The updated Ω is returned in line 10.

For better understanding, we first include an update example, shown in Fig. 3, to apply the Alg. 3. The previous service chain setting is the same in Fig. 2(a) and the current Ω is as shown in Fig. 2(b). If C_2 changes from $\{A \to C \to D\}$ to $\{A \to C \to E\}$ in Fig. 3(a), we first delete all of f_2's private middleboxes (for this example, there are no private middleboxes). Next, we reassign the new C_2 to $G_3 = \{C \to E\}$ with the minimum marginal increment of 24 while the values with G_1 and G_2 are 36 and 31. The updated service chain set is shown in Fig. 3(b), which is also optimal after the update.

Then we include one example of insertion in Fig. 4. Compared to Fig. 2(a), there is one new flow f_6, with a traffic rate of 4 and the requested service chain $C_6 = \{B \to C \to F \to G\}$, shown in Fig. 4(a). The values of the marginal increment with each grouped service chain in Fig. 2(b) are 34, 22, and 32, respectively. The minimum value is 22, so we assign f_6 to G_2. The returned service chain set is shown in Fig. 4(b). We also find out that the returned set is optimal.

Next, we delete the flow f_4. As $G_2 = C_4$, if f_4 is deleted, we are allowed to assign one service chain as G_2. The largest cost value among all the remaining grouped flows is $3 \cdot 3 = 9$ of f_1 that is assigned to G_1 with f_2 and f_3 together (f_5 is not grouped). So we update $G_1 = \{A \to C \to D \to E \to F\}$ and $G_2 = \{A \to B \to C\}$. The returned grouped service chain set is shown in Fig. 5(a).

Time complexity: The time complexity of Alg. 3 is $O(|F|)$. For the insertion, the time complexity is $O(k)$ in order to calculate the marginal increment and select the minimum one. For the deletion, it takes a constant time to delete a service chain and takes $O(|F|)$ to calculate the cost values of all flows and select the largest one. For the update, it takes a constant time to delete the old service chain and $O(k)$ to reassign, which is exactly the same procedure of the insertion. To sum up, the time complexity of Alg. 3 is $O(|F|)$.

B. Grouping for Load Balance

Load balancing refers to balancing the total traffic rate among all service chains. Here we need to mention that

f_2+f_3(3):
A→C→D→E→F

f_1(3): A→B→C

f_5(4): C→E

(a) Deleted service chains.

f_1+f_2(5):
A→B→C→D

f_4(6): B→F→G

f_3+f_5(5):
A→C→D→E→F

(b) Load balancing result.

Fig. 5: Deletion procedure and load balancing.

Algorithm 4 Load Balancing Method (LBM)

```
In: Sets of vertices $V$, edges $E$, flows $F$ and threshold $k$;
Out: The grouping chain set $\Omega$;
1: Select top $k$ flows with the largest traffic rate into $\Omega$;
2: for every flow $f$ with $\sum_{G\in\Omega} X_f^G = 0$ do
3:     Assign the flow $f$ into the grouped service chain $G \in \Omega$
       with smallest increment of the total traffic rate.
4: return The service chain set $\Omega$ after grouping.
```

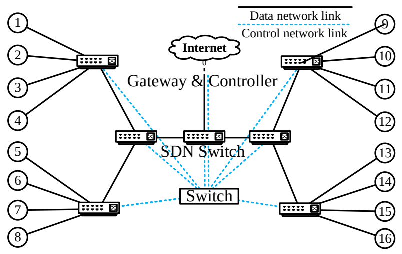
the load balancing can also serve as the objective of our problem, which can be represented as $\min\max_G \sum_{X_f^G=1} r_f$. Load balancing mainly limits the largest traffic load of a single service chain. It is also an extremely measurement index in today's network. This is because when the traffic is heavily unbalanced, the bandwidth resource of links is more likely to be not enough, resulting in an unexpected transmission delay because of link congestion. So we take this important index into consideration and propose the solution, called Load Balancing Method (LBM), in Alg. 4. In line 1, we initiate the grouped service chain set Ω with flows having the top k largest traffic rate. Lines 2-3 arrange an unassigned flow f to the grouped service chain with the smallest total traffic rate in each loop until all flows are assigned. Line 4 returns the grouped chain set Ω.

With the objective of load balancing, we continue to use the same setting in Fig. 2(a). The top 3 flows with the largest traffic rate are f_1, f_4, and f_5. So the initialized grouped service chain set is $\Omega = \{G_1 = C_1, G_2, C_4, G_3 = C_5\}$. For the unassigned flow f_2, we assign it to G_1, which has the minimum total traffic rate as 3. Then its total traffic rate is $3+2=5$. Next for the unassigned flow f_3, we assign it to G_3, which has the minimum total traffic rate as 4. Then its total traffic rate is $4+1=5$. The returned grouped service chain is shown in Fig. 5(b). The largest total traffic rate is 6 for G_2.

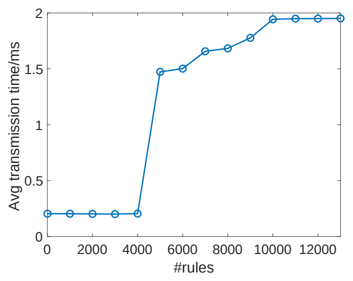
Time complexity: The time complexity of our proposed Alg. 4 is $O(\max\{|F|\log|F|, |F|k\})$. It takes $O(|F|\log|F|)$ to sort the traffic rate of all flows and select the top k. We need to keep a min-heap for k elements, which takes $O(k\log k)$. Then we have $O(|F|-k)$ loops. In each loop, it takes O(k) time to select the minimum value from the min-heap. So the total time complexity of Alg. 4 is $O(\max\{|F|\log|F|, |F|k\})$.

C. Converting Cost Function to Actual End-to-end Delay

We conduct real experiments to show the practicability of our motivation as well as our objective in our testbed. First we test the estimated time of single link transmission and rule



(a) Testbed topology.



(b) Average transmission time.

Fig. 6: Matching speed on the number of rules.

matching in a real system. Then we show that our objective function in this paper is an abstract model of actual end-to-end network delay.

Fig. 6 shows the average transmission time of a 64-byte packet with different number of rules in a switch in our SDN testbed. Our testbed is a four-layered perfect tree topology with 16 Dell servers and seven Pica8 P-3297 SDN switches, shown in Fig. 6(a). The capacity of the forwarding table in TCAM in a P-3297 SDN switch is 5000 (rules). We have a Dell 3248 PowerEdge server as our controller, which is running a OpenDaylight platform. We constantly send background traffic among all servers, of which the source and destination are randomly generated. Each packet has an identical size of 64 bytes. The number of forwarding rules in the root switch (the one connected to the controller) increases with a stride of 1000. We record the average transmission time (in milliseconds) of a single packet, whose path is 6-hop length (pass 6 links), on the increment of forwarding rules. As the transfer time on the links of a packet is too short (in a light speed), we use the average transmission time as the matching time inside a TCAM switch. The result is shown in Fig. 6(b). The time has a sharp increment when the number of rules is 5000. It indicates that when the number of rules is more than the capacity of TCAM, the matching time increases enormously. The results of less than 5000 rules verify the nearly-constant matching time of the TCAM forwarding table, which is around 0.2 milliseconds. When the number of rules is more than 5000, the matching time increases in a relatively steady way. The result demonstrates that the system performance will be seriously degraded when the total number of rules is more than the capacity of TCAM in a switch. This illustrates that we have to ensure the number of rules to be within the capacity of the forwarding tables.

We use $delay_1$ and $delay_2$ to denote the average delays

Total delay (ms)	TCAM matching	Software matching
Before grouping	0.68	3.04
After grouping	1.12	5.32

TABLE II: Time for transmission and processing.

of flows, whose matching rules are placed in slow software and fast TCAM switches, respectively. Additionally, $delay'$ is denoted as the average uniform delay of all flows after their service chains are grouped and all rules can be placed in TCAMs. Then the decrement of the delay can be calculated as $\#exceed\ rules \cdot delay_1 + capacity \cdot delay_2 - capacity \cdot delay'$. Here we use a detailed calculation to show that the grouping of rules will decrease the average end-to-end delay. There are usually less than 10 hops in a routing path of a flow. [25] mentions that the total number of middleboxes in a service chain is usually 3 to 5. The average processing time of a packet by a middlebox is around 0.1 ms [35] and the average transmission delay (including the rule matching and link transmission time) is 0.03ms within the TCAM capacity of the forwarding table and 0.33 ms beyond that capacity, obtained from Fig. 6.

We assume the average length of service chains is 4 before grouping, and 7 after grouping. The length of flow routing paths is usually proportional to the length of its required service chain, which is set as 2 here. The total estimated end-to-end delay of different switch matching is calculated in Tab. II. Here we take two results in the table as an examples. We assume the average length of service chains is 4 before grouping, and 7 after grouping. The total delay for TCAM before grouping is calculated as the sum of link transmission delay, $4 \cdot 2 \cdot 0.03 = 0.24$ms (the service chain length is 4, path length is $4 * 2$), and middlebox processing delay, $4 \cdot 0.1 = 0.4$ms, whose sum is 0.68ms. The total delay for software matching after grouping is $7 \cdot 2 \cdot 0.33 + 7 \cdot 0.1 = 5.32$ms. If we have 15000 rules before grouping and 5000 after grouping, the total decrement of the overall delay can be estimated as $(10000 \cdot 3.04 + 5000 \cdot 0.68 - 15000 \cdot 1.12)/(10000 \cdot 3.04 + 5000 \cdot 0.68) = 50.2\%$, which is a huge benefit. It demonstrates that the total delay is decreased by service chain grouping though flows may have a longer merged service chain to get processed. If the increment of path length is proportional to the increment of the service chain length, we can convert our cost objective function into the overall delay by adding a multiplier.

V. Evaluation

Simulations are conducted and numerical evaluations are presented to demonstrate the efficiency of our solutions. After we present the network and flow settings, the results are shown from different perspectives to provide insightful conclusions.

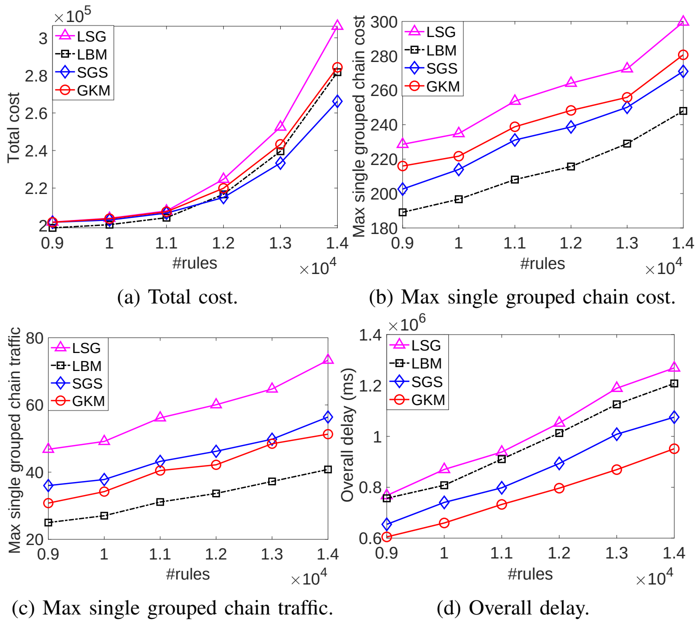
A. Settings

We use real data from either trustworthy sources or our testbed mentioned above and implement our simulations in MATLAB. The requested service chain and traffic rate of every flow are known in advance. We adopt the flow size distribution of the Facebook data center. The dataset contains flows of 10-minute packet traces of three different node types: a web-server rack, a single cache follower, and a Hadoop node [36]. More than 92% of flows are less than 12 Mbps in the dataset. Therefore, we set the traffic rates ranging from 0.1 to 12 Mbps with a granularity of 0.5 Mbps in this paper. We generate 20 different kinds of VNFs, each of which has a distinct label. The lengths of the requested service chains of flows are taken randomly between 3 and 10. The types of the VNFs are randomly selected from the generated 20 kinds. When the number of flows is the variable, it ranges from 9000 to 14000 with a granularity of 1000. As for the range of group number k, we choose from 2500 to 5000 with a granularity of 500. The default values of the number of flows and k are 10000 and 5000, respectively. We keep the default value of k as 5000, because it's slightly larger than the capacity of Pica8 P-3297 switch. The value of k even larger than 5000 may increase the delay because of overwhelming the TCAM capacity. We use 0.1 ms as the processing time for a middlebox. This processing time is used in [35] and verified in our testbed. The average processing and transmission (1-hop) time for the Firewall in our testbed is around 0.15 ms. We set 0.03 ms and 0.33 ms as the single-hop transmission time within and beyond the rule capacity limit k, which is obtained from our testbed evaluation result (see in Fig. 6). In our testbed, we have a small amount of flows for two reasons. Firstly, when the number of flows and data rate is higher, the delay increases due to queuing of the packets in SDN switches and the NICs. We wanted to avoid queuing delays as much as possible. Secondly, each of the links are 1Gbps, and it's hard to generate a heavily used link with a lot of flows as the number of servers in our small datacenter is limited. We agree with the reviewer that the performance will be low if we use the metrics of a testbed with a large number of flows. The runtime of the algorithm is dominated by k-means clustering algorithm which runs pretty fast in our machines. That is why we do evaluate the actual runtime of our algorithm.

B. Comparison Algorithms and Performance Metrics

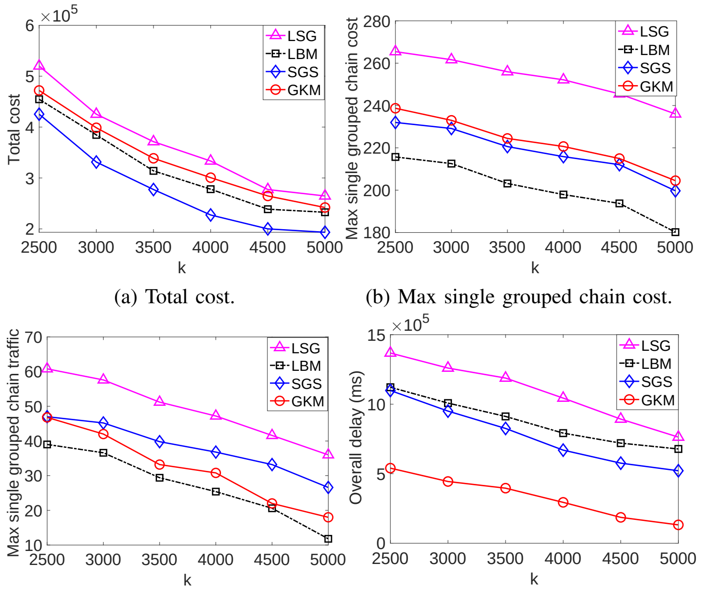
We include one benchmark scheme in our simulations, called *Largest Similarity Grouping* (LSG). In LSG two service chains having the largest number of similar VNFs are merged together and a new service chain is formed. This merging continues until the total number of chains is no more than k. We proposed four algorithms in this paper. Alg. 4 LBM is for the objective of load balancing. For the objective of minimizing the total cost, Alg. 1 SGS is the greedy solution using submodularity and Alg. 2 GKM is the solution based on k-means method. Alg. UGS is for updating service chains.

The variables include the number of flows and the group number k. The performance metrics include the total cost (our objective), the largest cost (our alternative objective of load balancing), the largest total traffic of a single grouped chain among all chains and the overall delay. The overall delay is calculated as the sum of delay among all flows. Our proposed


(a) Total cost. (b) Max single grouped chain cost.

(c) Max single grouped chain traffic. (d) Overall delay.

Fig. 7: Changing number of rules.


(a) Total cost. (b) Max single grouped chain cost.

(c) Max single grouped chain traffic. (d) Overall delay.

Fig. 8: Changing k.

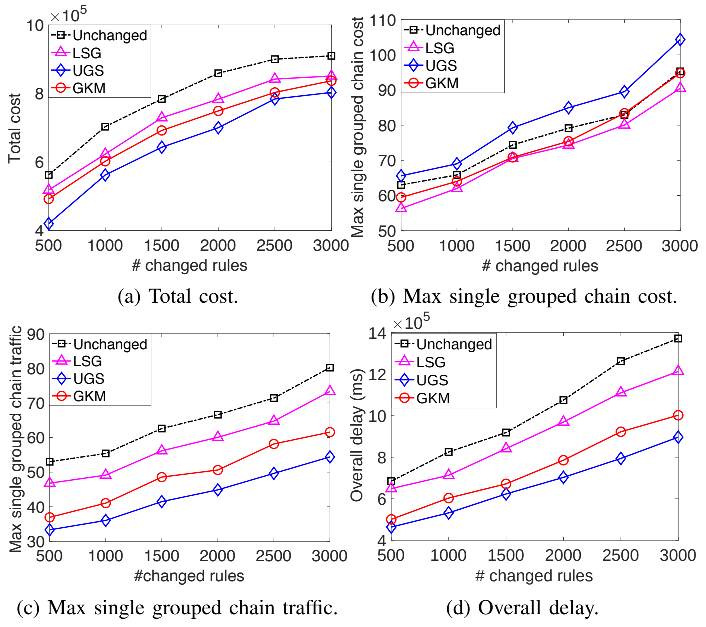
four algorithms have excellent performances on all metrics, especially their designed objective.

C. Results of Initialization

In this subsection, we show the simulation results of grouping initialization. We conduct simulations on two variables: the number of flows and the value of k, respectively.

Changing the number of flows: Fig. 7 shows the results of changing the number of rules from 2000 to 7000. From Fig. 7 (a), we can see that our Alg. SGS has the lowest total cost for all the number of rules while our Alg. GKM has the second lowest total cost. Here it is worth mentioning that the execution time of Alg. GKM is less than half of that of running Alg. SGS. This is because Alg. SGS needs to find the minimum value in each round, which is a little bit time-consuming. It demonstrates the trade-off between the performance and the efficiency of our proposed algorithms. We also test the estimated end-to-end delay using the constants of proportion and single hop delay. It shows that the overall delay is reduced by at least 36.7% compared with that before grouping. Fig. 7(b) shows the cost of the maximum single grouped chain, which is also the objective of load balancing. Alg. LBM has the best performance with the minimum cost, which illustrates its effectiveness. In Fig. 7(c), on average, the maximum length of our Alg. LBM is 16.5% less than that of Alg. SGS and 19.1% less than that of Alg. GKM. As for the maximum cost of a single grouped chain, shown in Fig. 7(d), Alg. LSG still has the worst performance because of its largest overall delay. This indicates that it is not enough to consider only the similarity of chains.

Changing k: Fig. 8 shows the results of changing k from 1000 to 3500. When the number of k is 5000, the performance of the algorithms is similar to that in Fig. 7 when the number of flows is 10000. Fig. 8(a) shows the total cost, which is our objective in this paper. Alg. SGS has the best performance with the minimum total cost, which illustrates its effectiveness on the total cost. Additionally, it also performs outstandingly on the other three metrics. As for the maximum cost of a single grouped chain, shown in Fig. 8(b), Alg. LBM still has the best performance over all k values while Alg. GKM and Alg. LSG have worse performances than in Fig. 7(b). It indicates that the influence of k is much larger than that of the number of flows. From Fig. 8(c), on average, the largest traffic of our Alg. LBM is 16.5% less than that of Alg. SGS and 19.1% less than that of Alg. LSG. Fig. 8(d) shows that Alg. LSG has the largest delay among all others.


(a) Total cost. (b) Max single grouped chain cost.

(c) Max single grouped chain traffic. (d) Overall delay.

Fig. 9: Update rules.

D. Results of Update

In this subsection, we conduct simulations on updating flows, including their traffic rate and requested service chains. We update the flows randomly. As the algorithm handles updates of flows one by one, we do not need to update the flows in burst. We include another benchmark algorithm, called Unchanged. It retains all allocated groups of original flows but updates the newly-grouped service chains. Fig. 9 shows the results of changing the number of updated rules from 2500 to 5000. Fig. 9(a) shows the result of the total cost. When the number of changed rules increases, the increment of the total cost of all algorithms becomes smaller. This is because all grouped chains become longer and include more middleboxes. Changed flows have less impact on the total cost. In Fig. 9(b), the maximum cost of a single grouped service chain becomes larger because more flows are changed and bigger changes of the grouped service chains are made. As for the maximum cost of a single grouped chain, shown in Fig. 9(c), Alg. Unchangeable still has the worst performance. This indicates that it is not enough to consider only the similarity of chains instead of both similarity and traffic. Fig. 9(d) shows the overall delay for all flows. Alg. UGS has the best performance with the minimum total delay, which illustrates its effectiveness. However, Alg. UGS needs to find the minimum value in each round, which is a little bit more time-consuming than Alg. GKM. It demonstrates the trade-off between the performance and the efficiency of our proposed algorithms.

In summary, our proposed four algorithms have excellent performances on all metrics, especially their designed objectives. The simulated results illustrate the importance of not only the traffic of each grouped service chain, but also the length of the service chain. We demonstrate the necessity of grouping service chains in order to save the number of rules in switches. Additionally, we demonstrate the trade-off between the performance and the time-efficiency of our proposed algorithms.

VI. Related Work

Service Chain: The research related to service chains have become more and more popular in recent years. Rost et al. [37] prove the NP-completeness and inapproximability of the service chain deployment under different constraint settings, extended from the virtual network embedding problem. [38] initiates the study of approximation algorithms and proposes a performance-guaranteed solution under the offline setting (given multiple flows), based on randomized rounding of Linear Programming, to maximize the total profit of satisfied flows. [39] aims to maximize the number of requests for each service chain and proposes a systematic way of VNF deployment to tune the proper link and server resource usages. [40] proposes a context-free language to formalize the chaining of middleboxes and describes the VNF resource allocation problem as a mixed integer quadratically constrained program. Rami et al. [41] locate VNFs in a way that minimizes both new VNF setup costs and the distance cost between VNF instances and the flows' paths; they provide near optimal approximation algorithms to guarantee a theoretically proven performance.

Flow Routing and Service Chain Deployment: The main challenges of our deployment problem lie in the selection of middleboxes locations and the allocation of each deployed middlebox of each service chain [11]. The vertex capacity constraint complicates the deployment, since flows have to be fully processed before reaching their destinations. Intuitively, if the VNF instances are deployed at most overlapping nodes along their routing paths, the processing volume is more likely to be used up, while flows with destinations far from the service chain may not be processed; if too far from their overlapping node the opportunity of sharing the processing volume of an instance is scarce so that some will be wasted and more middleboxes are needed 8. The service chain with multiple middleboxes makes the flow processing order problem even more complicated. This is because different flows on each middlebox of the service chain have various processing times. A flow cannot start (finish) being processed in a middlebox of a service chain, before its starting (finishing) time of its previous middlebox plus the transmission delay between these two middleboxes. Additionally, the link transmission delay between the locations of deployed middleboxes is necessarily taken into consideration, which further complicates the deployment problem.

Forwarding Table: In the flow table at each switch, forwarding rules define how to process the received flows based on requested policies. These rules are stored in TCAMs. TCAM enables fast lookups in switches with flexible wildcard rule patterns. However, the cost and power requirements limit the number of rules the switches can support to several thousands or tens of thousands [16]. In a network that has a large number of active flows, flow tables at switches can be easily overflowed, which could cause blocking of new flows or eviction of entries of some active flows [42]. The eviction of active rules, however, can severely degrade the network performance and overload the SDN controller, which is the last thing we want to happen. As a result, efficiently utilizing the forwarding table entries becomes extremely important. Instead of evicting rules, we can group flows by using tag. However, high performance network requirements are becoming more and more intense [43]. For example, data centers usually claim that packet loss rates are around 2% [44], while the requirements for wide area networks (WANs) and carrier-grade networks are higher [45].

Rule Coding: Most works on saving forwarding entries pay attention to the rule coding policies in TCAMs [46–48]. [46] formulates the problem as an abstract optimization problem based on two-level logic minimization, and proposes an exact solution and a number of heuristics. Rottenstreich et al. in [47] find a limited-size classifier that can correctly classify a high portion of the traffic so that it can be implemented in commodity switches with classification modules of a given size. [48] introduces an optimal algorithm that minimizes the number of rules needed to represent a weighted traffic distribution and proposes more efficient solutions to approach the exact distribution. However, there is little work related to enforcing service chains with limited capacities of forwarding tables in switches. Since SDN rules match with packets against specified header fields, forwarding ambiguity arises when a packet goes through a switch more than once, each time toward a different next hop [13]. This forwarding ambiguity is further exacerbated by middleboxes like NAT. Without their modification strategies as a priori, it is hard to pre-configure rules for modified packets. Bu et al. in [49] initially propose a simple tag-augmented forwarding rule routing strategy, but they focus on avoiding attacks instead of reducing the total number of forwarding rules.

VII. Conclusion

Forwarding rules are preferred to be inserted in high-speed matching TCAMs instead of slow-matching software switches. However, the capacity of TCAM is only thousands due to its high requirements for cost and power. In order to reduce rule matching delay, we need to effectively utilize rule entries in TCAM by grouping flows, where all flows in each group serve as a new flow to get processed by a longer merged service chain. Though more flows can have a faster rule matching time, it results in grouped flows detouring on a longer path and being processed by more middleboxes. In this paper, we aim at efficiently grouping flows with the minimum total cost while satisfying the capacity constraint of the forwarding tables. We first prove the submodularity of our objective function and propose corresponding solutions with an approximation ratio. Additionally, we also apply the k-means method to solve our problem with intuitive insights. We include extra discussion on several aspects. Extensive simulations are conducted to evaluate the performance of our proposed algorithms in various scenarios. For future work, we will scale up our testbed and conduct more large-scale and general experiments.

References

[1] F. Schneider, T. Egawa, S. Schaller, S.-i. Hayano, M. Schöller, and F. Zdarsky, "Standardizations of sdn and its practical implementation," *NEC Technical Journal, Special Issue on SDN and Its Impact on Advanced ICT Systems*, vol. 8, no. 2, 2014.

[2] M. Ghaznavi, E. Jalalpour, B. Wong, and R. Boutaba, "Fault tolerance for service function chains," *arXiv preprint arXiv:2001.03321*, 2020.

[3] P. Quinn and T. Nadeau, "Service function chaining problem statement," *draft-ietf-sfc-problem-statement-07 (work in progress)*, 2014.

[4] D. A. Joseph, A. Tavakoli, and I. Stoica, "A policy-aware switching layer for data centers," in *Proceedings of the ACM SIGCOMM 2008 conference on Data communication*.

[5] S. Kumar, M. Tufail, S. Majee, C. Captari, and S. Homma, "Service function chaining use cases in data centers," *IETF SFC WG*, p. 10, 2015.

[6] W. Haeffner, J. Napper, M. Stiemerling, D. Lopez, and J. Uttaro, "Service function chaining use cases in mobile networks," *Internet Engineering Task Force*, 2015.

[7] V. Sekar, N. Egi, S. Ratnasamy, M. K. Reiter, and G. Shi, "Design and implementation of a consolidated middlebox architecture," in *NSDI 2012*.

[8] Z. A. Qazi, C.-C. Tu, L. Chiang, R. Miao, V. Sekar, and M. Yu, "Simple-fying middlebox policy enforcement using sdn," in *SIGCOMM 2013*.

[9] A. Bremler-Barr, Y. Harchol, and D. Hay, "Openbox: a software-defined framework for developing, deploying, and managing network functions," in *Proceedings of the 2016 ACM SIGCOMM Conference*.

[10] J. Halpern, C. Pignataro *et al.*, "Service function chaining (sfc) architecture," in *RFC 7665*, 2015.

[11] Y. Chen, J. Wu, and B. Ji, "Virtual network function deployment in tree-structured networks," in *ICNP 2018*.

[12] ——, "Deploying virtual network functions with non-uniform models in tree-structured networks," *TNSM 2020*.

[13] S. K. Fayazbakhsh, L. Chiang, V. Sekar, M. Yu, and J. C. Mogul, "Enforcing network-wide policies in the presence of dynamic middlebox actions using flowtags," in *NSDI 2014*.

[14] G. Shen, Q. Li, S. Ai, Y. Jiang, M. Xu, and X. Jia, "How powerful switches should be deployed: A precise estimation based on queuing theory," in *INFOCOM 2019*.

[15] J. Wu, Y. Chen, and H. Zheng, "Approximation algorithms for dependency-aware rule-caching in software-defined networks," in *GLOBECOM 2018*.

[16] B. Stephens, A. Cox, W. Felter, C. Dixon, and J. Carter, "Past: Scalable ethernet for data centers," in *CoNEXT 2012*.

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

[18] S. Bera, S. Misra, and A. Jamalipour, "Flowstat: Adaptive flow-rule placement for per-flow statistics in sdn," *IEEE Journal on Selected Areas in Communications*, vol. 37, no. 3, pp. 530–539, 2019.

[19] A. B. Kahn, "Topological sorting of large networks," *Communications of the ACM*, vol. 5, no. 11, pp. 558–562, 1962.

[20] N. Handigol, B. Heller, V. Jeyakumar, D. Mazières, and N. McKeown, "I know what your packet did last hop: Using packet histories to troubleshoot networks," in *NSDI 2014*.

[21] X. Li and C. Qian, "A survey of network function placement," in *2016 13th IEEE Annual Consumer Communications Networking Conference (CCNC)*.

[22] X. .Jin, H. Liu, R. Gandhi, S. Kandula, R. Mahajan, M. Zhang, J. Rexford, and R. Wattenhofer, "Dynamic scheduling of network updates," in *SIGCOMM 2014*.

[23] D. B. Johnson, "Finding all the elementary circuits of a directed graph," *SIAM Journal on Computing*, vol. 4, no. 1, pp. 77–84, 1975.

[24] Y. Zhang, C. Truchan, M. Tatipamula, N. Beheshti, L. Beliveau, G. Lefebvre, R. Manghirmalani, R. Mishra, R. Patneyt, M. Shirazipour *et al.*, "Steering: A software-defined networking for inline service chaining," in *ICNP 2013*.

[25] "Service function chaining (SFC) architecture," 2014. [Online]. Available: https://tools.ietf.org/html/draft-quinn-sfc-arch-04

[26] "Cisco: Nat order of operation," 2019. [Online]. Available: http://www.cisco.com/c/en/us/support/docs/ip/network- address-translation- nat/6209- 5.html

[27] S. Fujishige, *Submodular functions and optimization*. Elsevier, 2005, vol. 58.

[28] J. A. Hartigan and M. A. Wong, "Algorithm as 136: A k-means clustering algorithm," *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, vol. 28, no. 1, pp. 100–108, 1979.

[29] K. Qiu, J. Yuan, J. Zhao, X. Wang, S. Secci, and X. Fu, "Fastrule: Efficient flow entry updates for tcam-based openflow switches," *IEEE Journal on Selected Areas in Communications*, vol. 37, no. 3, pp. 484–498, 2019.

[30] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: Enabling innovation in campus networks," *SIGCOMM 2008*.

[31] A. Ludwig, S. Dudycz, M. Rost, and S. Schmid, "Transiently secure network updates," in *SIGMETRICS 2016*.

[32] V. Jalaparti, I. Bliznets, S. Kandula, B. Lucier, and I. Menache, "Dynamic pricing and traffic engineering for timely inter-datacenter transfers," in *SIGCOMM 2016*.

[33] H. H. Liu, S. Kandula, R. Mahajan, M. Zhang, and D. Gelernter, "Traffic engineering with forward fault correction," in *SIGCOMM 2014*.

[34] J. Zheng, H. Xu, X. Zhu, G. Chen, and Y. Geng, "We've got you covered: Failure recovery with backup tunnels in traffic engineering," in *ICNP 2016*.

[35] "Middleboxes: Taxonomy and issues," 2002. [Online]. Available: https://tools.ietf.org/html/rfc3234

[36] A. Roy, H. Zeng, J. Bagga, G. Porter, and A. C. Snoeren, "Inside the social network's (datacenter) network," in *SIGCOMM 2015*.

[37] M. Rost and S. Schmid, "Np-completeness and inapproximability of the virtual network embedding problem and its variants," Technical Report,

Tech. Rep., 2018.

[38] "Virtual network embedding approximations: Leveraging randomized rounding," *arXiv preprint arXiv:1803.03622*, 2018.

[39] T.-W. Kuo, B.-H. Liou, K. C.-J. Lin, and M.-J. Tsai, "Deploying chains of virtual network functions: On the relation between link and server usage," in *INFOCOM 2016*.

[40] S. Mehraghdam, M. Keller, and H. Karl, "Specifying and placing chains of virtual network functions," in *CloudNet 2014*.

[41] R. Cohen, L. Lewin-Eytan, J. S. Naor, and D. Raz, "Near optimal placement of virtual network functions," in *INFOCOM 2015*.

[42] Z. Guo, R. Liu, Y. Xu, A. Gushchin, A. Walid, and H. J. Chao, "Star: Preventing flow-table overflow in software-defined networks," *Computer Networks*, vol. 125, pp. 15–25, 2017.

[43] H. Zheng, W. Chang, and J. Wu, "Coverage and distinguishability requirements for traffic flow monitoring systems," in *IWQoS 2016*.

[44] Y. Zhu, N. Kang, J. Cao, A. Greenberg, G. Lu, R. Mahajan, D. Maltz, L. Yuan, M. Zhang, B. Zhao, and H. Zheng, "Packet-level telemetry in large datacenter networks," in *SIGCOMM 2015*.

[45] T. Mizrahi and Y. Moses, "Time4: Time for sdn," *IEEE Transactions on Network and Service Management*, vol. 13, no. 3, 2016.

[46] R. McGeer and P. Yalagandula, "Minimizing rulesets for tcam implementation," in *INFOCOM 2009*.

[47] O. Rottenstreich and J. Tapolcai, "Lossy compression of packet classifiers," in *ANCS 2015*.

[48] O. Rottenstreich, Y. Kanizo, H. Kaplan, and J. Rexford, "Accurate traffic splitting on commodity switches," in *SPAA 2018*.

[49] K. Bu, Y. Yang, Z. Guo, Y. Yang, X. Li, and S. Zhang, "Flowcloak: Defeating middlebox-bypass attacks in software-defined networking," in *INFOCOM 2018*.

VIII. Acknowledgment

This research was supported in part by NSF grants CNS 1824440, CNS 1828363, CNS 1757533, CNS 1629746, CNS-1651947, and CNS 1564128.

Jie Wu is the Director of the Center for Networked Computing and Laura H. Carnell professor at Temple University. He also serves as the Director of International Affairs at College of Science and Technology. He served as Chair of Department of Computer and Information Sciences from the summer of 2009 to the summer of 2016 and Associate Vice Provost for International Affairs from the fall of 2015 to the summer of 2017. Prior to joining Temple University, he was a program director at the National Science Foundation and was a distinguished professor at Florida Atlantic University. His current research interests include mobile computing and wireless networks, routing protocols, cloud and green computing, network trust and security, and social network applications. Dr. Wu regularly publishes in scholarly journals, conference proceedings, and books. He serves on several editorial boards, including IEEE Transactions on Mobile Computing, IEEE Transactions on Service Computing, Journal of Parallel and Distributed Computing, and Journal of Computer Science and Technology. Dr. Wu was general co-chair for IEEE MASS 2006, IEEE IPDPS 2008, IEEE ICDCS 2013, ACM MobiHoc 2014, ICPP 2016, and IEEE CNS 2016, as well as program co-chair for IEEE INFOCOM 2011 and CCF CNCC 2013. He was an IEEE Computer Society Distinguished Visitor, ACM Distinguished Speaker, and chair for the IEEE Technical Committee on Distributed Processing (TCDP). Dr. Wu is a Fellow of the AAAS and a Fellow of the IEEE. He is the recipient of the 2011 China Computer Federation (CCF) Overseas Outstanding Achievement Award.

Rajorshi Biswas is a PhD student of Computer and Information Sciences at Temple University, Philadelphia. He achieved his bachelor's degree from Bangladesh University of Engineering and Technology, Bangladesh. He is cur- rently doing research at the Center for Networked Computing (CNC) which is focused on network technology and its applications. His research areas include Wireless Networks, Wireless Sensor Networks, Wireless Security, Cryptography, Cognitive Radio Networks etc. He published his research in many conferences and journals including IEEE ICPADS 2018, IEEE GLOBECOM 2019, IEEE ICC 2019, IEEE Sarnoff 2019, and Resilience Week 2019.

Yang Chen received her B.Eng. degree in Electronic Engineering and Information Science from University of Science and Technology of China in 2015. She is currently a Ph.D. candidate in the Department of Computer and Information Sciences, Temple University, Philadelphia, Pennsylvania, USA. Her current research focuses on Software Defined Networks, especially resource allocation, flow scheduling and network updates.