A Dynamic Placement Mechanism of Service Function Chaining Based on Software-defined Networking

Yicen Liu, Yu Lu, Xingkai Chen, Xi Li, Wenxin Qiao and Liyun Chen
Shijiazhuang Campus, Army Engineering University of PLA
Shijiazhuang, Hebei 050003 - China
[e-mail: oecluyu@sina.com]
*Corresponding author: Yu Lu

Received December 25, 2017; revised April 18, 2018; accepted June 5, 2018; published October 31, 2018

Abstract

To cope with the explosive growth of Internet services, Service Function Chaining (SFC) based on Software-defined Networking (SDN) is an emerging and promising technology that has been suggested to meet this challenge. Determining the placement of Virtual Network Functions (VNFs) and routing paths that optimize the network utilization and resource consumption is a challenging problem, particularly without violating service level agreements (SLAs). This problem is called the optimal SFC placement problem and an Integer Linear Programming (ILP) formulation is provided. A greedy heuristic solution is also provided based on an improved two-step mapping algorithm. The obtained experimental results show that the proposed algorithm can automatically place VNFs at the optimal locations and find the optimal routing paths for each online request. This algorithm can increase the average request acceptance rate by about 17.6% and provide more than 20-fold reduction of the computational complexity compared to the Greedy algorithm. The feasibility of this approach is demonstrated via NetFPGA-10G prototype implementation.

Keywords: Software-defined networking, service function chaining placement, integer linear programming, greedy heuristic solution

This research was supported by the National Natural Science Foundation of China [No. 51377170,61271152] and National Youth Science Foundation of China [No. 61602505]. We express our thanks to Dr. Waleed Ejaz who checked our manuscript.

1. Introduction

In recent years, the demand for Internet services has constantly expanded due to the explosive propagation of mobile devices and the emergence of novel networking paradigms such as the Internet of Things (IoTs) [1]. Today's networks ubiquitously deploy vertically integrated proprietary middleboxes (e.g., Firewall, Intrusion Detection System (IDS), and Network Address Translation (NAT)). However, the static service model causes two main problems: Firstly, the traditional middlebox comes at the cost of high Capital Expenditure (CAPEX) and Operating Expenditure (OPEX). Secondly, it is impossible to add new functionality to an existing middlebox, which makes it hard for network operators to place new services. Consequently, the new dynamic service model research has become a hot topic [2-3].

Both the promising Software-defined Networking (SDN) [4] network architecture and Network Function Virtualization (NFV) [5] technology can solve these outlined problems. SDN decouples the control plane from the data plane. The controller is the central administrator of the network, it obtains the global network and can be used to programmatically configure forwarding flow tables in the switches, thus enabling Virtualized Network Function (VNF) orchestration. NFV proposes to move the packet processing from hardware middleboxes toward software, thus providing possibilities for network optimization and cost reduction. The dynamic Service Function Chaining (SFC) is an enabler of the SDN/NFV networking paradigm. It provides a flexible and economical alternative to today's static network environment for application service providers (ASPs) and Internet service providers (ISPs).

At present, dynamic SFC technology is still in its infancy. However, the optimal placement is one of the most challenging problems in the NFV-based network [6-8]. Significant current research focuses on the single aspect of resource utilization. For example, Mijumbi et al. used the Tabu Search algorithm that performs both mapping and scheduling of VNFs by searching for better solutions in its neighborhood [9]. Deval et al. proposed the Affinity-based heuristic algorithm that considers the cache capacity of a node with priority. However, this approach causes a longer VNF processing waiting time [10]. Xiong et al. proposed a service function placement mechanism, which is biased towards favoring the use of nodes that are loaded the least [11]. Zhang et al. proposed a scalable framework (called StEERING) for dynamic traffic routing through the proper VNF sequence. The Greedy algorithm which is biased towards favoring using paths that are least loaded was proposed. However, this approach results in unbalanced network loads [12]. Bari et al. used a Viterbi algorithm to place the service function chaining, which considers the total cost of the entire service path, without considering resource utilization [13]. The same algorithm has also been used by the authors Liu et al. However, this approach has been abandoned due to computational complexity [14].

The above studies are either solely based on node resource utilization or on making only some preliminary explorations regarding link resource utilization. Few studies have designed a specific placement strategy that considers global network utilization, which is the focus in this paper. Here, we denote the optimal SFC placement problem, focusing on how to place the VNFs at the optimal locations and how to find optimal routing paths, with the objective of minimizing resource consumptions while simultaneously balancing the network loads. Specifically, the contributions of this paper are:

1. The service function chaining placement problem is theoretically formulated by the integer linear programing model.
2. A novel greedy heuristic solution is proposed based on the improved two-step mapping algorithm, and the performance is evaluated via Matlab.
3. The effectivity of our placement mechanism is validated on an SDN framework with OpenDaylight [15], NetFPGA-10G [16] and sFlow [17].

The remainder of this paper is organized as follows: we start by introducing the SDN/NFV-based architecture and by formally explaining the network model of our system (Section 2). Then, the integer linear programming (ILP) formulation is presented (Section 3). Next, a novel greedy heuristic algorithm is proposed that obtains the near-optimal solution (Section 4). Our proposed solution is validated via Matlab and a NetFPGA-10G prototype implementation is given (Section 5). Finally, we conclude this work by providing promising future directions (Section 6).

2. Network Model

2.1 SDN/NFV-based architecture

Fig. 1 provides an overview of our architecture. The components of the architecture can be classified into three types including the orchestration plane, the control plane and the data plane. The orchestration plane is the development environment of VNFs, and its main role is to manage and orchestrate the VNFs to control the global network, according to different network demands and application scenarios. The control plane can programmatically configure forwarding flow tables in the switches to enable VNF orchestration. The SDN controller plays a key role in mapping VNFs and virtual links onto the substrate network based on a specific placement strategy, and consequently form the service path. The data plane mainly includes routers/switches and NFV platforms, which forwards the traffic flow and provides service processing.

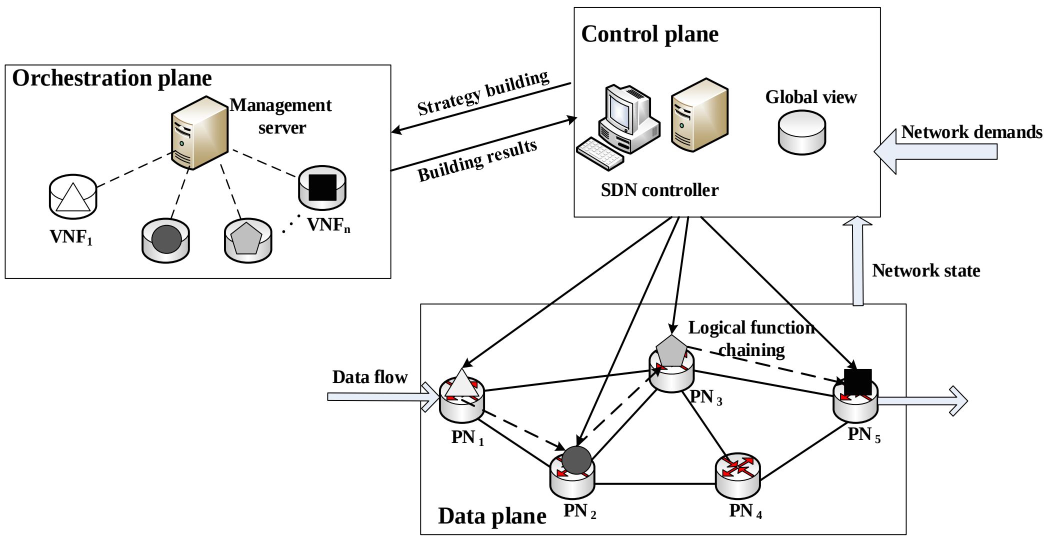


Fig. 1. Overview of the service function chaining placement using SDN

2.2 Placement Model

The placement process of service chaining is formulated as a two-level model, which includes the "SFC Policy—Logical Function Chaining" process and the "Logical Function Chaining—Service Path" process. Fig. 2 provides an overview of this two-level model. The "SFC Policy—Logical Function Chaining" process is a dynamic choreography of VNFs implemented by applications through northbound interfaces. The "Logical Function Chaining—Service Path" process allocates the physical resources reasonably through southbound interfaces.

Definition 1. SFC Policy. The SFC policy is represented by a 4-tuple $P = \{v_s, v_t, (c_1, c_2, ...c_m), \tau\}$, where v_s and v_t denote the ingress and egress switches, respectively. $C = \{c_1, c_2, ...c_m\}$ represents the ordered VNF sequence the traffic flow must pass (e.g., $NAT \mapsto Firewall \mapsto IDS$). τ denotes the expected propagation delay according to the SLAs and m is the number of service functions.

Definition 2. Logical Function Chaining. Logical function chaining is formed by the VNF function modules based on the SFC policy. Here, we assume that all deployable nodes and their contextual relationships have been defined as a directed graph $G^v = (N^v, L^v)$, where $N^v = \{n_1^v ..., n_{i-1}^v, n_i^v, n_{i+1}^v, ...n_m^v \mid 1 \le i \le m\}$ represents the set of traffic nodes (switches and VNFs) and $L^v = \{(n_i^v, n_j^v), ... \mid 1 \le i < j \le m\}$ denotes the links between them.

Definition 3. Service Path. We place the VNFs at the optimal locations and obtain the optimal routing path based on the placement strategy, then forming a service path. Here, we assume that the substrate network is defined as an undirected graph $G^s = (N^s, L^s)$, with the node set $N^s = \{n_1^s, ..., n_{j-1}^s, n_j^s, n_{j+1}^s ...n_k^s \mid 1 \le j \le n\}$ representing substrate nodes and the edge set $L^s = \{(n_i^s, n_j^s), ... \mid 1 \le i < j \le n\}$ representing substrate links.

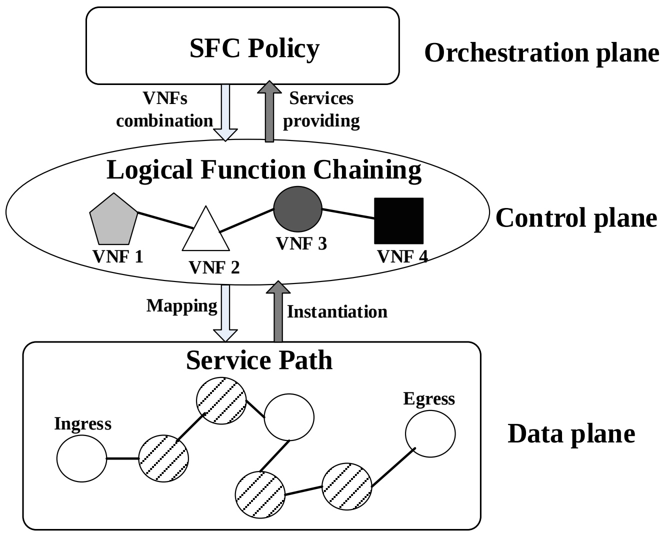


Fig. 2. The "SFC Policy—Logical Function Chaining—Service Path" process

The procedure of our placement is as follows: Firstly, the orchestration plane can orchestrate the VNFs based on the SFC policy, which satisfies the requirements of the tenant. Secondly, the SDN controller runs the service placement decision module, which solves the optimization problem of minimizing the resource consumption. Finally, the results of the decision module are output of the configuration to map a logical function chaining to the substrate network via the uniform APIs. If the mapping is successful, the corresponding instantiation resource is allocated, and then the service path is formed; otherwise, the mapping fails. As shown in **Fig. 2**, the shaded nodes represent the substrate nodes where the VNFs are placed, while the unshaded substrate nodes only forward data flow.

3. Integer Linear Programming (ILP) Formulation

In this section, we introduce the problem of minimizing the total resource consumption for each SFC mapping in the NFV-based networks as an ILP optimization problem. The goal of the presented optimization model is to minimize the total physical resource consumption to the ISPs, thus satisfying other constraints such as the site capacity constraint, the link capacity constraint, and the placement constraint (due to SLAs, which will be explained later). We formulated the optimization model to deploy workflows on the VNFs and assign client requests to these workflows to meet the service demands. The resource consumption can be divided into two categories: VNF resources consumption and virtual link resources consumption. Consequently, an optimization model with applicable constraints is formulated and the SFC placement problem can be solved via an Integer Linear Programming (ILP) methodology. The list of variables used in the ILP is provided in **Table 1**.

Table 1. Variables used in the ILP

Variable	Explanation
$x_{n_i^v}^{n_k^s}$	If the VNF node $n_i^v \in N^v$ is located in the substrate node $n_k^s \in N^s$
$y_{n_i^v, n_j^v}^{n_k^s, n_l^s}$	If the virtual link $(n_i^v, n_j^v) \in L^v$ is located in the substrate link $(n_k^s, n_l^s) \in L^s$
$c(n_i^v)$	Amount of requested CPU resources for the VNF node $n_i^v \in N^v$
$r(n_k^s)$	Amount of remaining CPU resources for the substrate node $n_k^s \in N^s$
$c(n_i^v, n_j^v)$	Amount of requested bandwidth resources for the virtual link $(n_i^v, n_j^v) \in L^v$
$r(n_k^s, n_l^s)$	Amount of remaining bandwidth resources for the substrate link $(n_k^s, n_l^s) \in L^s$
$D(n_k^s)$	Maximum VNF processing delay on the substrate node $n_k^s \in N^s$
$D(n_k^s, n_l^s)$	Maximum communication delay along the substrate link $(n_k^s, n_l^s) \in L^s$

Consider the problem of optimal SFC mapping in a NFV-enabled network. $x_{n_i^v}^{n_k^s}$ is introduced to indicate which of the VNFs $n_i^v \in N^v$ can be provisioned on a substrate node $n_k^s \in N^s$. That is:

$$x_{n_i^v}^{n_k^s} = \begin{cases} 1 & \text{a VNF } n_i^v \in N^v \text{ is provisioned on } n_k^s \in N^s, \\ 0 & \text{otherwise.} \end{cases}$$

The binary variable $y_{n_i^v, n_j^v}^{n_k^s, n_l^s}$ is introduced to formulate the routing of the service chaining. Therefore:

$$y_{n_i^v,n_j^v}^{n_k^s,n_l^s} = \begin{cases} 1 & \text{a virtual link } (n_i^v,n_j^v) \in L^v \text{ is provisioned on } (n_k^s,n_l^s) \in L^s, \\ 0 & \text{otherwise.} \end{cases}$$

The following lists the constraints of the optimization model:

1) Site capacity constraint: the total load of the CPU resource across all SFC requests and all VNFs at each substrate node $n_k^s \in N^s$ should be less than or equal to its CPU capacity. We express this constraint as follows:

$$x_{n_i^v}^{n_k^s} \times c(n_i^v) \le r(n_k^s), \forall n_i^v \in N^v, n_k^s \in N^s \tag{1}$$

2) Link capacity constraint: the total load of the bandwidth resource across all SFC requests and all logical links at each substrate link $(n_k^s, n_l^s) \in L^s$ should be less than or equal to its bandwidth capacity. We represent this constraint as follows:

$$y_{n_i^v,n_j^v}^{n_k^s,n_l^s} \times c(n_i^v, n_j^v) \le r(n_k^s, n_l^s),\ \forall (n_k^s, n_l^s) \in L^s, (n_i^v, n_j^v) \in L^v \tag{2}$$

3) Delay constraint: for wide area service chaining, the end to end delay of each SFC request should be less than or equal to D^P . This delay includes both the VNF processing delay at the datacenter sites and the communication delay along the links. We present the delay constraints for the service path as follows:

$$\sum_{(n_k^s,n_l^s)\in L^s} \sum_{(n_i^v,n_j^v)\in L^v} y_{n_i^v,n_j^v}^{n_k^s,n_l^s} \times D(n_k^s,n_l^s) + \sum_{n_i^v\in N^v} \sum_{n_k^s\in N^s} x_{n_i^v}^{n_k^s} \times D(n_k^s) \le D^P, \forall n_i^v \in N^v, n_k^s \in N^s, (n_k^s,n_l^s) \in L^s, (n_i^v,n_j^v) \in L^v \tag{3}$$

4) Connectivity constraint: we present the connectivity constraint that ensures both the in-flow and out-flow of each switch in the substrate network are equal except at the ingress and egress switches. This constraint is represented as follows:

$$\sum_{(n_i^v,n_j^v)\in L^s} \sum_{(n_k^s,n_l^s)\in L^v} (y_{n_i^v,n_j^v}^{n_k^s,n_l^s} - y_{n_j^v,n_i^v}^{n_k^s,n_l^s}) = x_{n_i^v}^{n_k^s} - x_{n_i^v}^{n_l^s}, \forall n_i^v \in N^v, n_k^s \in N^s, (n_k^s,n_l^s) \in L^s, (n_i^v,n_j^v) \in L^v \tag{4}$$

5) Placement constraint: to ensure that every traffic node is provisioned onto exactly one VNF; we also need to ensure that every link in a traffic request is provisioned on one or more substrate links within the networks. We present the placement constraint as follows:

$$\sum_{n_i^v\in N^v} \sum_{n_k^s\in N^s} x_{n_i^v}^{n_k^s} = 1,\ \forall n_i^v \in N^v, n_k^s \in N^s \tag{5}$$

$$\sum_{(n_i^v,n_j^v)\in L^s} \sum_{(n_k^s,n_l^s)\in L^v} (y_{n_i^v,n_j^v}^{n_k^s,n_l^s} + y_{n_i^v,n_j^v}^{n_k^s,n_l^s}) \ge 0,\ \forall (n_k^s,n_l^s) \in L^s, (n_i^v,n_j^v) \in L^v \tag{6}$$

6) Variable constraint: to ensure that the values of $x_{n_i^v}^{n_k^s}$ and $y_{n_i^v,n_j^v}^{n_k^s,n_l^s}$ equal 0 or 1, we must have:

$$x_{n_i^v}^{n_k^s} \in \{0,1\}, \forall n_i^v \in N^v, n_k^s \in N^s \tag{7}$$

$$y_{n_i^v,n_j^v}^{n_k^s,n_l^s} \in \{0,1\},\ \ \forall (n_k^s,n_l^s) \in L^s, (n_i^v,n_j^v) \in L^v \tag{8}$$

Now, given a set of network services, each of which requiring a strictly ordered chaining of network functions as corresponding traffic traverses, the fundamental objective is to efficiently place VNFs to the datacenter sites and map virtual links to substrate links that can satisfy all constraints mentioned above while minimizing resource consumption across the network. This objective can be mathematically described as:

$$Minimize \quad \sum_{n_i^v \in N^v} \frac{\alpha}{r(n_k^s)+\delta} x_{n_i^v}^{n_k^s} + \sum_{(n_i^v,n_j^v)\in L^v} \frac{\beta}{r(n_k^s,n_l^s)+\delta} y_{n_i^v,n_j^v}^{n_k^s,n_l^s} \tag{9}$$

Here, α and β are weighting factors that are used to adjust the relative importance of the resource consumption components. δ represents as a minimum value, which ensures the nonzero property of the denominator. To improve the utilization of the physical resources, $r(n_k^s)$ and $r(n_k^s,n_l^s)$ are taken as denominators so that the placement strategy can be more inclined to nodes or links, which contain the largest amount of the remaining resources.

The SFC placement problem is NP-Hard since we can reduce this problem to the *Multi-Commodity*, *Multi-Plant*, and *Capacitated Facility Location Problem* [18], which is more commonly known as the *Trans-Shipment Problem* [19] by imposing a constraint on the appropriate number of VNFs that can be deployed in the network. Both of these problems are known to be NP-hard. Therefore, the SFC placement problem is also NP-Hard. The method to solve this problem can be divided into two categories: an exact algorithm and a heuristic algorithm [20]. The exact algorithm has limited applicability due to its computational complexity. However, the heuristic algorithm can be used to obtain the near-optimal solution of the placement problem within a polynomial time. Therefore, we propose a heuristic algorithm to solve this problem in the next section.

4. Greedy Heuristic Solution

In this section, we present a greedy heuristic to solve the SFC placement problem. Rost et al. [21], Schmid et al. [22] and Moens et al. [23] have reported that the heuristic algorithm achieves good performance in dealing with integer linear programming. In this study, we extend the baseline two-step mapping algorithm [24] with several improvements, such as greedy strategy, sorting mechanism, and k-Dijkstra algorithm. Then, the *Greedy node mapping with k-Shortest Path link mapping* algorithm (called G-kSP) is proposed to obtain the optimal SFC placement for the ILP model.

4.1 Introduction of the G-kSP algorithm

Existing schemes either focus exclusively on the resource utilization of nodes or on the resource utilization of links. To overcome this narrow focus, an improved two-step mapping algorithm is presented that efficiently assigns both VNFs and virtual links onto the substrate network for each online SFC request, with the objective of minimizing the resource consumption to the ISPs. In our G-kSP algorithm, the requested resource, the remaining resource and the QoS are considered. The requested resources can be decomposed into the requested CPU resources for the VNFs and the requested bandwidth resources for the virtual links. The remaining physical resource includes the remaining CPU resources for the substrate nodes and the remaining bandwidth resources for the substrate links.

This algorithm keeps track of the available node/link resources of the substrate network. Note that for the substrate node $n_k^s \in N^s$, we do not use $r(n_k^s)$ alone as the metric of the available resource, because we not only want to ensure sufficient available CPU capacity, but also consider bandwidth capacity to prepare the subsequent virtual link mapping stage. Therefore, we define the amount of available resources for the substrate node n_k^s via:

$$R(n_k^s) = r(n_k^s) \sum_{e_k^s \in E(n_k^s)} r(e_k^s) \tag{10}$$

where $E(n_k^s)$ represents the set of all adjacent substrate links of n_k^s, $r(n_k^s)$ represents the remaining CPU resource of n_k^s, and $r(e_k^s)$ represents the unoccupied bandwidth resource for the substrate link e_k^s.

Thus, for a VNF node $n_i^v \in N^v$, the requested resource $C(n_i^v)$ is computed as follows:

$$C(n_i^v) = c(n_i^v) \sum_{e_i^v \in E(n_i^v)} c(e_i^v) \tag{11}$$

where $E(n_i^v)$ represents the set of all adjacent virtual links for n_i^v, $c(n_i^v)$ represents the CPU requirements for the VNF node n_i^v, and $c(e_i^v)$ represents the bandwidth requirements for the virtual link e_i^v.

The greedy heuristic algorithm mainly runs in two steps. Firstly, the VNF nodes are sorted in a descending manner based on their resource requirements, and a near-optimal VNF placement is obtained by running the greedy strategy, which prioritizes the substrate node with maximum available resources. Then, the k-shortest path algorithm is used to produce an approximation approach to minimize the bandwidth consumption via virtual link mapping. For the former approach, the sorting mechanism can obtain the optimal service path without requiring global searching, and the proposed greedy strategy can optimize operational network utilization. For the latter approach, the k-shortest path algorithm can find optimal mapping from a virtual link between VNFs to specific substrate links.

4.2 Formulation of the G-kSP algorithm

The main steps of the G-kSP algorithm can be described as follows:

Input: Substrate network topology G^s, SFC requests P

Output: VNF mapping set $\left\{ M_N(n_i^s) \middle| \forall n_i^s \in N^s \right\}$, virtual link mapping set $\left\{ M_L(n_i^s, n_j^s) \middle| \forall (n_i^s, n_j^s) \in L^s \right\}$

Step 1. Acquisition of the VNF queue Q. Q contains the mapping order of VNFs. Sort the VNFs in descending order based on their requested resources (defined in Equation (11)), then store this VNFs in queue Q.

Step 2. Acquisition of the mappable set M. M contains the substrate nodes that satisfy the constraints mentioned above. Find the mappable node set M of the substrate node that satisfies both the restrictions and available resources. If this is found, store substrate nodes in set M; otherwise, open new VMs for VNF mapping.

Step 3. Placing the VNF at optimal locations. Take out the first VNF in queue Q, and map the VNF to the substrate node with "maximum available resources" (as defined in Equation (12)). If the VNF n_i^v is successfully mapped to the substrate node n_k^s , $x_{n_i^v}^{n_k^s} = 1$, store it in set M_N ; otherwise, trace back to the suboptimal solution in the mappable node set M . Update queue Q .

$$n_k^s = \left\{ \hat{n}_k^s \middle| \hat{n}_k^s = \arg\max R(n_k^s), \forall n_k^s \in N^s \right\} \tag{12}$$

Step 4. Acquisition of the VNF mapping set M_N . M_N contains the placement information of the VNFs. Determine whether there is the remaining VNF in queue Q. If feasible, go to Step 3 and execute the next VNF in queue Q ; otherwise, obtain the set M_N .
Step 5. Acquisition of the k-shortest path set R . R contains k paths with minimum transport bandwidth between ingress and egress. Find the k-shortest paths from ingress node v_s to egress node v_t by using the K-Dijkstra algorithm, while preserving the set M_N and the ordering of VNFs φ_P. Denote the k-shortest paths as $R = \{r_1, r_2, ..., r_k\}$.
Step 6. Finding the optimal routing path M_L . M_L contains the placement information of the virtual links. Sort R in descending order based on the occupied bandwidth resource $c(n_i^v, n_j^v)$, and take the path with the minimum occupied bandwidth resource. If the virtual link (n_i^v, n_j^v) is successfully mapped to the substrate link (n_k^s, n_l^s) , and $y_{n_i^v, n_j^v}^{n_k^s, n_l^s} = 1$, store it in set M_L ; otherwise, trace back to the suboptimal solution in the set R .
Step 7. Monitor the service time and allocate physical resources. Determine whether the service path should preserve QoS (defined in Equation (3)). If feasible, the corresponding instantiation resource is assigned; otherwise, reclaim allocated resources. Then, update the network state information.

Based on the provided description, we input all algorithm parameters and call the G-kSP algorithm to obtain the SFC placement scheme. The process flow is shown in **Fig. 3**. Upon the controller, the decision module runs the G-kSP algorithm, and achieves the results $M_N(n_i^s)$ and $M_L(n_i^s, n_j^s)$, which represent the locations of the VNFs and virtual links, respectively.

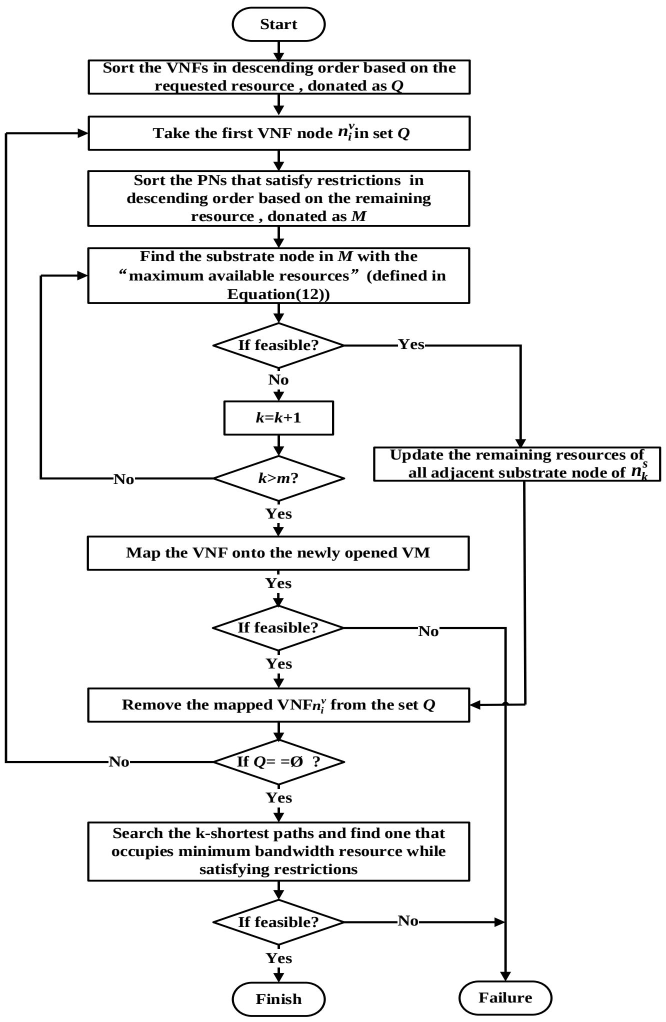

Start
Sort the VNFs in descending order based on the requested resource , donated as Q
Take the first VNF node n_i^v in set Q
Sort the PNs that satisfy restrictions in descending order based on the remaining resource , donated as M
Find the substrate node in M with the "maximum available resources" (defined in Equation(12))
If feasible?
Yes
No
k=k+1
Update the remaining resources of all adjacent substrate node of n_k^s
No
k>m?
Yes
Map the VNF onto the newly opened VM
Yes
If feasible?
No
Yes
Remove the mapped VNFn_i^v from the set Q
No
If Q= =Ø ?
Yes
Search the k-shortest paths and find one that occupies minimum bandwidth resource while satisfying restrictions
If feasible?
No
Yes
Finish
Failure

Fig. 3. Process flow of G-kSP algorithm

4.3 Algorithm Complexity

The G-kSP placement algorithm proposed here aims to work around the complexity of the formulated ILP. Here, we assume that a network supports a constant number of network services. Let the number of the substrate nodes and the maximum length of a service chaining be m and n, respectively. For each network service, the main processes of the proposed G-kSP are: *i*) placing the VNF at the optimal locations and *ii*) finding the optimal routing path. The first process is based on both the sorted mechanism and greedy strategy. It is well known that this process has a certain complexity, which follows the order of $O(n\log n)$. In the second part of the proposed algorithm, the K-Dijkstra algorithm has a complexity of $O(K+n\log n+M)$. In conclusion, the complexity is $O(K+n\log n+M)$ for the process of one SFC request.

5. Performance Evaluation

Our performance evaluation focuses on the following aspects: (i) analyze the performance of the proposed G-kSP algorithm on the 6-nodes network topology adopted by SNDlib library.

Compare the performance of our solution to that of the Greedy, Tabu Search, GFP, and GLL based optimal solutions, which have already been studied in the literature for the SFC placement problem; (ii) perform trace driven simulations on an SDN framework with OpenDaylight, NetFPGA-10G, and sFlow, verify the effectivity of the proposed G-kSP algorithm, and compare the performance of our algorithm to that of Shortest-path, round-robin, and Random, which has already been developed in the OpenDaylight SFC project.

5.1 Algorithm evaluation

5.1.1 Experimental workloads

To evaluate the performance of the proposed algorithm, the experimental environment is set up on a PC with a 3.6 GHz two core Intel® Core™ i7 and 8GB RAM. The GT-ITM tool [25] is used to generate different network topologies and the G-kSP algorithm is implemented with MATLAB. For the topology dataset, we use the datacenter network (6 nodes, 14 links) provided by the SNDlib library [26]. As shown in **Fig. 4**, node 6 is used as egress, and node 1 is used as ingress. In the substrate network, we assume that each node can be used as the service-providing node. Each node of the network represents a single data center, which can carry the physical resource capacity (e.g., CPU, memory, and storage).

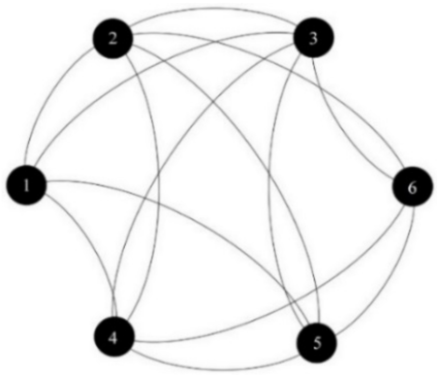


Fig. 4. Example network topology

The choice of these simulation parameters as well as their distribution are motivated by simulations and evaluations of a well-studied and related virtual network embedding problem [27]. The main parameters that were used to create the virtual links and VNFs in these simulations are chosen randomly following a uniform distribution with minimum and maximum values shown in **Table 2**.

Table 2. Simulation parameter ranges

Parameter	Minimum	Maximum
Number of substrate nodes	6	6
Number of VNFs per service	2	5
Function processed by each node	1	5
Function processing times	15	30
VNF or virtual link resource demand	0.5	0.8
Substrate CPU or bandwidth capacity	100	150

5.1.2 Evaluation results

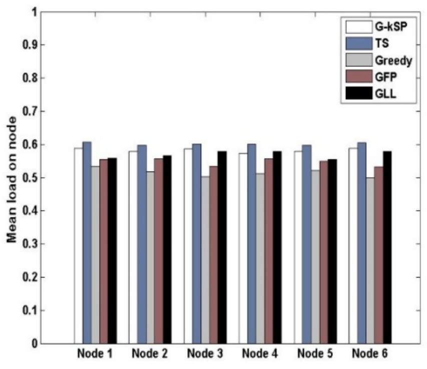
We use a synthetic policy to evaluate our algorithm in terms of node load balancing, RSP load balancing, SFC request ratio, and running time. Our proposed algorithm is compared to four other SFC placement algorithms that have been widely used in the literature. The notations and description of different algorithms are listed in **Table 3**. The results of these simulations are shown in **Figs. 5-8**. In the following, we present these results.

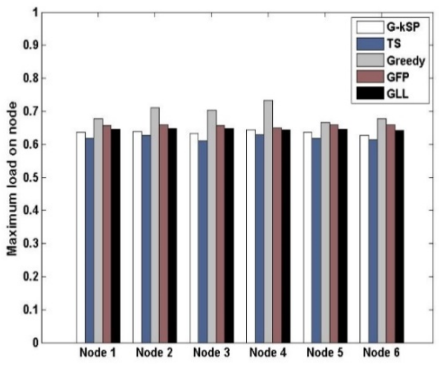
Table 3. Comparison of algorithms

Algorithm	Description
TS [9]	Tabu Search-based Network Function Mapping and Scheduling
Greedy [12]	The simple Greedy-based Network Function Mapping and Scheduling
GFP [27]	Greedy mapping with bias towards Fast Processing
GLL [28]	Greedy mapping with bias towards Least Loaded
G-kSP	Greedy node mapping with k-Shortest Path link mapping

Metric 1 Node Load Balancing: This is the load accumulation on the substrate node. The node load balancing can be computed as follows:

$$LB_{n_i^s} = \sum_{n_k^s \in M_N} \sum_{n_i^v \in N^v} \frac{c(n_i^v)}{r(n_k^s)} \tag{13}$$





(a) Mean node load (b) Maximum node load

Fig. 5. Performance characteristic of the node load

Fig. 5 shows the mean and maximum loads on each node (about 100 SFC requests). The results indicate that TS can achieve the largest mean node load, while the maximum node load is below that of the other four algorithms, which shows that TS finds better solutions in balancing the node. In contrast, Greedy is less effective and the other improved greedy algorithms G-kSP, GLL, and GFP lie in the middle of the compared algorithms. Among these, TS performs only slightly better than G-kSP. This is because TS has a chance to iteratively improve the solution; therefore, the node resource can be effectively utilized. The reason why G-kSP performs well could be attributed to the proposed greedy strategy, which improves the global network utilization. The reason why Greedy performs worse than the other schemes could be due to the fact that Greedy achieves higher node load by routing the traffic through a shorter path, and hence, Greedy falls a little short on optimal node placement. Ultimately, GLL performs better than GFP. This is because the least loaded nodes are likely to have shorter queues, which implies that services mapped onto such nodes are processed earlier, and hence they do not occupy the node resources for extended periods.

Metric 2 RSP Load Balancing: This is the load accumulation on the Service Function Path (RSP). Now, we can compute the RSP load balancing as follows:

$$LB_{RSP} = \omega_N \sum_{n_k^s \in M_N} \sum_{n_i^v \in N^v} \frac{c(n_i^v)}{r(n_k^s)} + \omega_L \sum_{(n_k^s, n_l^s) \in M_L} \sum_{(n_i^v, n_j^v) \in L^v} \frac{c(n_i^v, n_j^v)}{r(n_k^s, n_l^s)} \tag{14}$$

Here, ω_N and ω_L represent the impact factors of the node load and the link load, respectively.

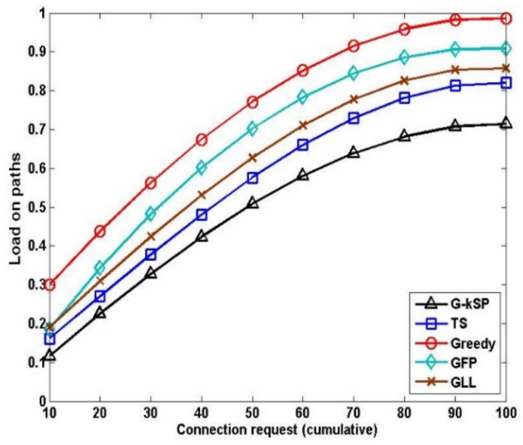


Fig. 6. RSP load comparison

Fig. 6 shows the load on service function paths. As shown in **Fig. 6**, the curve of each algorithm gradually becomes gentler, with increasing number of arriving requests. This is because the decreasing of available resources leads to the rejection of service requests. As indicated in **Fig. 6**, our proposed G-kSP preforms significantly better than the other tested algorithms, while Greedy has the highest RSP load. This is because G-kSP focuses on allocating resources from the global perspective, while Greedy focuses on optimizing utilization of bandwidth resources, and the other algorithms TS, GLL, and GFP only focus on optimizing the utilization of CPU resources. The reason why TS performs only slightly better than GLL and GFP is because TS is specifically formulated with the objective of minimizing the flow time. This gives TS an advantage in utilizing the physical resource while balancing the load. Finally, GFP has a considerably higher value of RSP load compared to GLL. This is because this algorithm always tries to map a given VNF to the node that processes it faster. This means that a node that has the least processing time for any given VNF is likely to always be over loaded, thus extending the queue at such a path.

Metric 3 Request Acceptance Rate: This is the ratio of the number of SFCs successfully deployed to the total number of SFC requests. Now, we can compute the request acceptance rate as follows:

$$\eta = \lim_{T \to \infty} \frac{\sum_{t=0}^{T} SFC_{accepted}}{\sum_{t=0}^{T} SFC_{total}} \tag{15}$$

where $\sum_{t=0}^{T} SFC_{accepted}$ represents the number of SFCs successfully deployed to the substrate network from $t=0$ to $t=T$, $\sum_{t=0}^{T} SFC_{total}$ represents the total number of SFC requests that arrive from $t=0$ to $t=T$.

Fig. 7 shows the variation of request acceptance ratio with service arrivals. The mean request acceptance rate of the Greedy algorithm is 74.9%, the mean request acceptance rate of TS is 89.3%, the mean request acceptance rate of GLL is 83.3%, the mean request acceptance rate of GFP is 78.6%; however, when G-kSP is used, the mean request acceptance rate is 93.1%. This is a measure of how efficiently the algorithm uses network resources to accept service requests. As shown in **Fig. 7**, G-kSP has the highest request acceptance rate, while Greedy has the lowest request acceptance rate. This is because G-kSP reduces the service path load and occupies less physical resources for longer periods, which could lead to receiving more SFC requests. Therefore, lower service path load results in a higher request acceptance rate. The fact that the Greedy algorithm performs worst because it prioritizes the service path with the lowest delay, and hence reuses the same path frequently, which could possibly lead to the rejection of service requests. Finally, we observe that the improved greedy scheme, which is biased towards favoring using nodes that are least loaded (GLL), performs better than GFP, which is based on favoring nodes with the best processing capacities. This is because least loaded nodes are likely to have shorter queues, which implies that VNFs mapped onto such nodes get processed earlier, and hence they do not occupy the node resources for extended periods, which could possibly lead to the rejection of service requests.

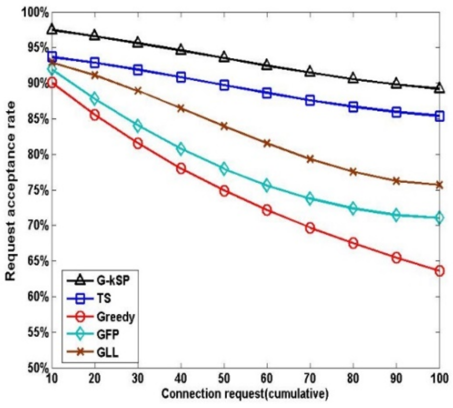


Fig. 7. Request rate comparison

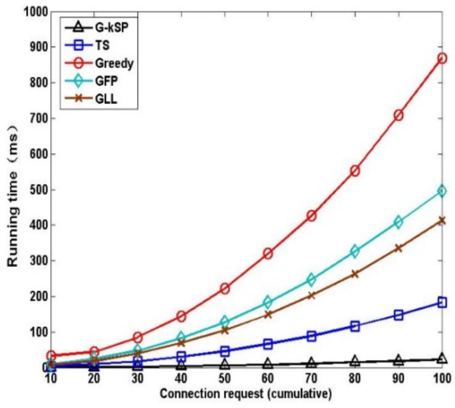


Fig. 8. Running time comparison

Metric 4 Running Time: This is the time required to find the service path placement for a given SFC request and network topology. **Fig. 8** shows the running time with service arrivals. Towards the end of the simulation (at about 100 service requests), the running time of the Greedy algorithm is 881.7 milliseconds, the running time of TS is 192.6 milliseconds, the running time of GLL is 398.2 milliseconds, and the running time of GFP is 508.7 milliseconds, while the running time of G-kSP is 40.1 milliseconds. The results show that the running time of Greedy is about 20 times longer than that of G-kSP. This is because Greedy searches for all the service paths based on the exhaustive method. Thus, Greedy performs $O(m^n)$ computations (refer to [12]). However, the running time of TS is affected by the length of the Tabu list λ, TS performs $O(m^2 + m\lambda)$ computations (refer to [9]). Finally, we found that GLL performs better than GFP; GLL specifically ensures that VNFs are mapped to those substrate nodes with the lowest loading, which ultimately reduces the waiting times of their executions. Thus, the complexity of GLL is $O(kmn+m^2)$ (refer to [28]), where k represents the number of iterations. This is in contrast to GFP, which always tries to map a given VNF to the substrate node that processes it faster, which implies that other VNFs for the same service should wait for such a VNF. Thus, the complexity of GFP is $O(km^3n^2)$ (refer to [27]). Combined with the analysis presented in Section 4.3, the trend of the curves in **Fig. 8** is basically consistent with the theoretical analysis.

5.2 Prototype implementation

To demonstrate the effectivity of our proposed placement mechanism, its operation is shown on the NetFPGA-10G prototype. In this section, we follow the implementation method that was proposed by Gibb et al [29]. We used the 6-node network topology provided by [30]. **Fig. 9** shows an overview of the NetFPGA-10G simulation environment, where six OpenFlow switches are managed by the OpenDaylight controller. The OpenDaylight controller provides an interface module of service chaining placement, which allows a third party to provide custom algorithms. Therefore, our G-kSP algorithm in this subsection is implemented in Java and added to the OpenDaylight controller via Eclipse.

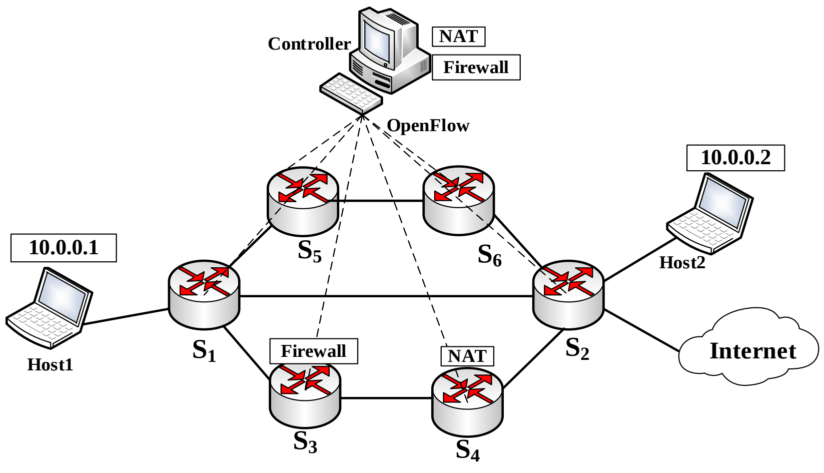


Fig. 9. NetFPGA-10G Simulation Environment

For the given network, OpenDaylight Helium release is used as the SDN controller and the OpenFlow 1.3 protocol is used as the south interface between the controller and switches. For the topology dataset, we use the traces available from [30] and replay the traffic for between random source-destination pairs. Real VNF modules (Firewall and, NAT) [31] are used to validate the effectivity of SFC placement. Each switch connects one single server, which provides VNF module in the network. **Table 4** lists the parameters used for both servers and VNFs.

Table 4. Server and VNF data used in the simulation

Server Data		
Physical CPU Cores	Idle Energy	Peak Energy
16	80.5W	2735
VNF Data		
Network Function	CPU Required	Processing Capacity
Firewall	4	600Mbps
NAT	4	900Mbps

5.2.1 Placement validation

OpenDaylight is the central administrator of the network, it obtains the global network, and can be used to programmatically configure forwarding flow tables in the switches, thus enabling VNF orchestration. The Firewall module can only be available on switch S_3 and the NAT module can be available on the switch S_4. Different SFC strategies can be configured by the OpenDaylight controller to satisfy the service demands of the tenant. The experimental results are shown in **Table 5** and the execution of the experiment can be decomposed into five stages as follows:

- **Test 1**. The policy $P_1 = \{$Host 1, Host 2$\}$ is added, namely, no VNF modules need to be placed. OpenDaylight automatically configures the forwarding flow table “dst = Host 2; actions = output: S_2” onto S_1 based on the shortest path algorithm. The data flow is driven to pass through switch S_3 and S_4, thus achieving communication between Host 1 and Host 2.
- **Test 2**. The policy is changed to $P_2 = \{$Host1, Host2, (Firewall)$\}$, namely, only the Firewall module needs to be placed. The forwarding flow tables “dst=Host2; actions = output: Firewall”, “dst = Host2; actions = output: S_2” are automatically configured to the S_3 and S_4, respectively. The data flow firstly enters S_3 for function processing, and then is forwarded to Host2 through S_4. The service path can be described as “$S_1 \rightarrow S_3 \rightarrow$ Firewall $\rightarrow S_4 \rightarrow S_2$”.
- **Test 3**. The policy is changed to $P_3 = \{$Host1, Host2, (NAT)$\}$, namely, only the NAT module needs to be placed. The corresponding forwarding flow tables “dst = Host2; actions = output: S_4”, “dst = Host2; actions = output: NAT” are automatically configured to the switch S_3 and S_4, respectively. Therefore, S_3 serves as the transition node, and S_4 serves as the service node. The service path can be described as “$S_1 \rightarrow S_3 \rightarrow S_4 \rightarrow$ NAT $\rightarrow S_2$”.
- **Test 4**. The policy is changed to $P_4 = \{$Host1, Host2, (Firewall, NAT)$\}$, namely, the Firewall and NAT modules need to be placed simultaneously. The forwarding flow tables “dst = Host2; actions = output: Firewall”, “dst = Host2; actions = output: NAT” are automatically configured to S_3 and S_4, and the service path can be described as “$S_1 \rightarrow S_3 \rightarrow$ Firewall $\rightarrow S_4 \rightarrow$ NAT $\rightarrow S_2$”.

- **Test 5**. The policy is changed to $P_5 = \{$Host1, Host2, (NAT, Firewall)$\}$, namely, S_3 and S_4 fail to be configured with the corresponding function modules; therefore, the service path cannot be formed and Host1 and Host2 cannot communicate with each other.

Table 5. Experimental test results

Test	SFC policy	Service path	Delay
1	P_1 = {Host1, Host2}	S_1.output:$S_2 \rightarrow S_2$.output:Host2	0.2ms
2	P_2 = {Host1, Host2, (Firewall)}	S_1.output:$S_3 \rightarrow S_3$.Firewall $\rightarrow S_3$.output:$S_4 \rightarrow S_4$.output:$S_2 \rightarrow S_2$.output:Host2	20.4ms
3	P_3 = {Host1, Host2, (NAT)}	S_1.output:$S_3 \rightarrow S_3$.output:$S_4 \rightarrow S_4$.NAT $\rightarrow S_4$.output:$S_2 \rightarrow S_2$. output: Host2	30.6ms
4	P_4= {Host1, Host2, (Firewall, NAT)}	S_1.output:$S_3 \rightarrow S_3$.Firewall $\rightarrow S_3$.output:$S_4 \rightarrow S_4$.NAT $\rightarrow S_4$.output:$S_2 \rightarrow S_2$.output:Host2	50.9ms
5	P_5 = {Host1, Host2, (NAT, Firewall)}	—	∞

As shown in **Table 5**, the communication delay between Host1 and Host2 is consistent with the theoretical analysis, and the service path is the identical to the one delivered.

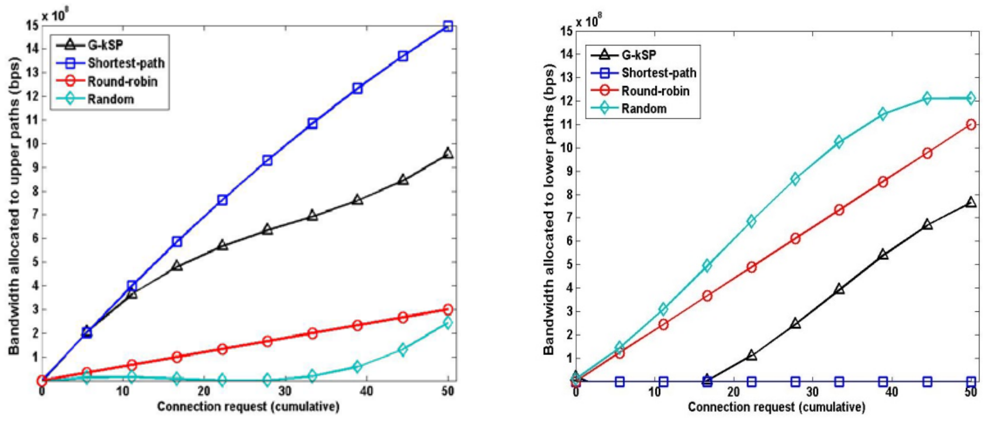
5.2.2 Resource utilization analysis

We have configured that the bandwidth capacities of the upper path ($S_1 — S_5 — S_6 — S_2$) and the lower path ($S_1 — S_3 — S_4 — S_2$) are 1.5 Gbps and 1 Gbps, respectively. The connections need to be routed from Host1 to Host2, the real VNF module needs to be used (Firewall available on nodes S_3 and S_5, NAT available on nodes S_2 and S_4). The 30Mbps SFC request is repeatedly mapped between Host1 to Host2 onto the example topology. The sFlow-RT and host-sFlow are used to obtain the utilization of nodes and links.

We compare the obtained results to three other SFC placement schemes, which have been developed in the OpenDaylight SFC project [32]. The notations and description of different algorithms are list in **Table 6**. The results of the simulations are shown in **Fig. 10** and **Fig. 11**. In the following, these results are discussed.

Table 6. Comparison of algorithms

Algorithm	Description
Shortest-path	Shortest-path selects the candidate VNFs that minimizes the service path
Round-robin	Round-robin assigns the candidate VNFs using cyclic selection
Random	Candidate VNFs are selected randomly
G-kSP	G-kSP selects the candidate VNFs that optimizes network utilization



(a) Upper path allocation (b) Lower path allocation

Fig. 10. Comparison of path allocation

Fig. 10 shows the bandwidth allocation on both upper and lower path. The first 20 connection requests of the G-kSP algorithm are assigned to the upper path due to its higher resource capacity. Note that the processing resource allocation is split between nodes S_5 and S_6 proportionally to their capacity. Since the network loads on the upper path, the lower path is preferred and connection requests 20 through 50 are both service paths, with the majority going to the lower path. For the shortest-path algorithm, the entire 50 connection requests are always assigned to the upper path due to the higher resource capacity. For the random algorithm, the entire 50 connection requests are randomly assigned to the upper path. For the round-robin algorithm, the entire 50 connection requests are assigned to the upper path and the lower path via cyclic selection. The reason that G-kSP performs better than the other tested schemes is because our proposed scheme has a chance to iteratively improve network utilization.

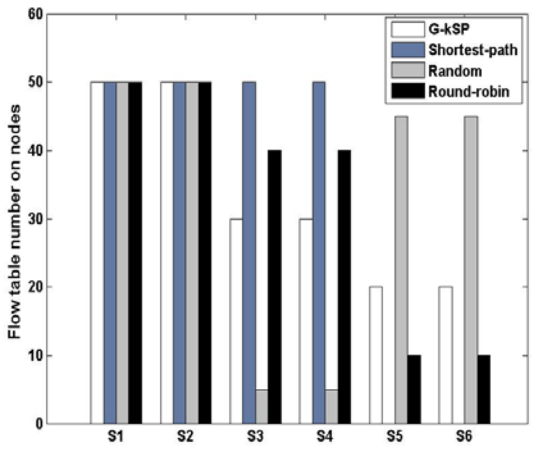


Fig. 11. Comparison of node allocation

Fig. 11 shows the flow table number on nodes. As the network reaches its capacity, G-kSP can balance the node significantly better than the tested other schemes. The reason why G-kSP outperforms the others could be attributed to ensuring that it consistently achieves higher resource utilization and dynamic resource allocation across all mappings. This ensures that the number of used nodes remains large, thus balancing the loads on the nodes and achieving higher resource utilization. However, Random, Round-robin, and Shortest-path achieve lower resource utilization, because during all mappings, resources are used on clearly suboptimal paths in a static manner.

6. Conclusion

To solve the SFC placement problem in the SDN/NFV environment, an optimal model with applicable constraints is formulated. This problem has been solved via the Integer Linear Programming (ILP). The ILP model has limited applicability due to both resource consumption and computational complexity. To resolve this limitation, a G-kSP placement approach is proposed based on the two-step algorithm. Built on the top of SDN and the advantages of the Greedy heuristic algorithm, our placement strategy can simultaneously obtain the optimal placement of virtual network functions and routing optimal paths.

Simulation showed that the algorithm can optimize the network resource utilization, request acceptance rate, and running time, compared to Greedy, Tabu, GLL, and GFP algorithms, which have already been studied for the SFC placement problem. Our trace driven on the NetFPGA-10G demonstrates that the G-kSP can improve network utilization, compared to Round-robin, Shortest-path, and Random schemes, which have already been developed in the OpenDaylight SFC project. In our future work, it will be important to explore the possibility of introducing failure-resilience by placing backup VNFs and virtual links that can take over the traffic processing tasks from failed SFC.

References

[1] Cisco, "Cisco visual networking index: forecast and methodology, 2016-2021," September, 2017. Article (CrossRef Link)

[2] Bhamare D, Jain R and Samaka M, "A survey on service function chaining," *Journal of Network & Computer Applications*, vol. 75, no. 3, pp. 138-155, 2016. Article (CrossRef Link)

[3] Mechtri M, Ghribi C and Zeghlache D, "A scalable algorithm for the placement of service function chains," *IEEE Transactions on Network & Service Management*, vol. 13, no. 3, pp. 533-546, 2016. Article (CrossRef Link)

[4] McKeown N, Anderson T and Balakrishnan H, "Openflow: enabling innovation in campus networks," in *Proc. of ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69-75, 2008. Article (CrossRef Link)

[5] Mijumbi R, Serrat J and Gorricho J L, "Management and orchestration challenges in network functions virtualization," *IEEE Communications Magazine*, vol. 54, no. 1, pp. 98-105, 2016. Article (CrossRef Link)

[6] Hmaity A, Savi M and Musumeci F, "Virtual network function placement for resilient service chain provisioning," in *Proc. of International Workshop on Resilient Networks Design and Modeling IEEE*, pp.325-328, 2016. Article (CrossRef Link)

[7] Kim S, Park S and Kim Y, "VNF-EQ: dynamic placement of virtual network functions for energy efficiency and QoS guarantee in NFV," *Cluster Computing*, vol. 20, no. 3, pp. 1-11, 2017. Article (CrossRef Link)

[8] Kuo T W, Liou B H and Lin C J, "Deploying chains of virtual network functions: on the relation between link and server usage," in *Proc. of IEEE INFOCOM 2016 - the, IEEE International Conference on Computer Communications*, pp. 1-9, 2016. Article (CrossRef Link)

[9] Mijumbi R, Serrat J, and Gorricho J L, "Design and evaluation of algorithms for mapping and scheduling of virtual network functions," *Network Softwarization*, pp. 1-9, 2015. Article (CrossRef Link)

[10] Bhamare D, Samaka M and Erbad A, "Optimal virtual network function placement in multi-cloud service function chaining architecture," *Computer Communications*, vol. 102, no. 3, pp. 1-16, 2017. Article (CrossRef Link)

[11] Xiong G, Hu Y X and Tian L, "A virtual service placement approach based on improved quantum genetic algorithm," *Information and Electrical Engineering Frontier*, vol. 17, no. 7, pp. 661-671, 2016. Article (CrossRef Link)

[12] Zhang Y, Beheshti N and Beliveau L, "StEERING: A software-defined networking for inline service chaining," in *Proc. of IEEE International Conference on Network Protocols*, pp. 1-10, 2014. Article (CrossRef Link)

[13] Bari M F, Chowdhury S R and Ahmed R, "On orchestrating virtual network functions," in *Proc. of International Conference on Network and Service Management*, pp. 50-56, 2015. Article (CrossRef Link)

[14] Liu C X, Lu G Q and Tang H B, "A virtual network function viterbi algorithm adaptive deployment method," *Journal of electronics and information technology*, vol. 38, no. 11, pp. 2922-2930, 2016. Article (CrossRef Link)

[15] Medved J, Varga R and Tkacik A, "OpenDaylight: Towards a model-driven SDN controller architecture," in *Proc. of IEEE, International Symposium on A World of Wireless, Mobile and Multimedia Networks*, pp. 1-6, 2014. Article (CrossRef Link)
[16] Gibb G, "NetFPGA-10G Project," March, 2015. Article (CrossRef Link)
[17] McCloghrie K, "sFlow Standard v5," July, 2004. Article (CrossRef Link)
[18] Pirkul H and Jayaraman V, "A multi-commodity, multi-plant, capacitated facility location problem: formulation and efficient heuristic solution," *Computers & Operations Research*, vol. 25, no. 10, pp. 869-878, 1998. Article (CrossRef Link)
[19] Chiou C C, "Transshipment problems in supply chainsystems: review and extensions," *Supply Chain*, InTech, 2008. Article (CrossRef Link)
[20] Cafieri S, Hansen P and Liberti L, "Improving heuristics for network modularity maximization using an exact algorithm," *Discrete Applied Mathematics*, vol. 163, no. 163, pp. 65-72, 2014. Article (CrossRef Link)
[21] Rost M and Schmid S, "Service chain and virtual network embeddings: approximations using randomized rounding," 2016. Article (CrossRef Link)
[22] Schmid S, "Online admission control and embedding of service chains," *Post-Proceedings of the, International Colloquium on Structural Information and Communication Complexity*, Springer-Verlag New York, Inc, pp. 104-118, 2015. Article (CrossRef Link)
[23] Moens H and Turck F D, "VNF-P: A model for efficient placement of virtualized network functions," in *Proc. of International Conference on Network and Service Management*, IEEE, pp. 418-423, 2014. Article (CrossRef Link)
[24] Yu M, Yi Y and Rexford J, "Rethinking virtual network embedding: substrate support for path splitting and migration," in *Proc. of ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 17-29, 2008. Article (CrossRef Link)
[25] Zegura E W, Calvert K L and Bhattacharjee S, "How to model an internetwork," in *Proc. of Proceedings of IEEE Infocom*, pp. 594-596, 1996. Article (CrossRef Link)
[26] Orlowski S, Wessäly R and Pióro M, "SNDlib 1.0—survivable network design library," *Networks*, vol. 55, no. 3, pp. 276-286, 2010. Article (CrossRef Link)
[27] Mijumbi R, Serrat J and Gorricho J L, "Self-managed resources in network virtualisation environments," in *Proc. of Ifip/ieee International Symposium on Integrated Network Management*, pp. 1099-1106, 2015. Article (CrossRef Link)
[28] Duan T, Lan J L and Cheng G Z, "Metafunction-based SDN Function combination mechanism," *Journal on Communications*, vol. 36, no. 5, pp. 156-166, 2015. Article (CrossRef Link)
[29] Gibb G, Zeng H and Mckeown N, "Outsourcing network functionality," in *Proc. of The Workshop on Hot Topics in Software Defined Networks*, pp. 73-78, 2012. Article (CrossRef Link)
[30] T Benson, "Network traffic characteristics of data centers in the wild," in *Proc. of ACM IMC'10*, pp 267-280. Article (CrossRef Link)
[31] Shin S, Porras P and Yegneswaran V, "Fresco: modular composable security services for software-defined networks," in *Proc. of Proceedings of Network & Distributed Security Symposium*, 2013. Article (CrossRef Link)
[32] OpenDaylight Project, "OpenDaylight service function chaining (SFC): beryllium feature integration system test," 2015. Article (CrossRef Link)

Yicen Liu received his Master's degree in communication and information system from Shijiazhuang Campus of Army Engineering University, Shijiazhuang, China, in 2018. He is currently a Ph. D candidate, with Information Engineering Department, Shijiazhuang Campus of Army Engineering University, Shijiazhuang, China. His current interest is software-defined service and VNF intelligent orchestration technology.

Yu Lu received his Ph. D degree from the Beijing University of Aeronautics and Astronautics, Beijing, China. He is currently a full professor, with Information Engineering Department, Shijiazhuang Campus of Army Engineering University. His current interests are in the area of network security control, software-defined security.

Xingkai Chen received his Master's degree in communication and information system from Shijiazhuang Campus of Army Engineering University, Shijiazhuang, China, in 2013. He is currently a Ph. D candidate, with Information Engineering Department, Shijiazhuang campus of Army Engineering University. His current interest is the key technologies of the future network architecture.

Xi Li received his Ph. D degree in computer science technology from Shijiazhuang Campus of Army Engineering University, Shijiazhuang, China, in 2018. He is currently a lecturer, with Information Engineering Department, Shijiazhuang Campus of Army Engineering University. His current interest is network security and information technology of equipment support.

Wenxin Qiao received his Master's degree in software engineering from Shijiazhuang Campus of Army Engineering University, Shijiazhuang, China, in 2016. She is currently a Ph. D student, with Information Engineering Department, Shijiazhuang Campus of Army Engineering University. Her current interest is the key technologies of network virtualization reliability.

Liyun Chen received his Ph. D degree in computer science technology from Shijiazhuang Campus of Army Engineering University, Shijiazhuang, China, in 2011. He is currently a full professor, with Information Engineering Department, Shijiazhuang Campus of Army Engineering University. His current interests are in the area of artificial intelligence, and deep learning.