This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/JIOT.2019.2937110, IEEE Internet of Things Journal

Low-latency and Resource-efficient Service Function Chaining Orchestration in Network Function Virtualization

Gang Sun, Zhu Xu, Hongfang Yu, Xi Chen, Victor Chang, Athanasios V. Vasilakos

***Abstract*—Recently, network function virtualization (NFV) has been proposed to solve the dilemma faced by traditional networks and to improve network performance through hardware and software decoupling. The deployment of the service function chain (SFC) is a key technology that affects the performance of virtual network function (VNF). The key issue in the deployment of SFCs is proposing effective algorithms to achieve efficient use of resources. In this paper, we propose a service function chain deployment optimization (SFCDO) algorithm based on a breadth-first search (BFS). The algorithm first uses a BFS based algorithm to find the shortest path between the source node and the destination node. Then, based on the shortest path, the path with the fewest hops is preferentially chosen to implement the SFC deployment. Finally, we compare the performances with the greedy and simulated annealing (G-SA) algorithm. The experiment results show that the proposed algorithm is optimized in terms of end-to-end delay and bandwidth resource consumption. In addition, we also consider the load rate of the nodes to achieve network load balancing.**

***Index Terms*— Network function virtualization; Service function chain; End-to-end delay; Resource consumption**

I. INTRODUCTION

With the increase in network users and the development of services, today's telecommunication industry needs to store and transmit large amounts of data. Hardware-based networks cannot withstand the impact of these applications. In most traditional networks, each network function required separate and expensive hardware, which caused the network to become rigid and increased network capital and operating expenses. Network function virtualization (NFV) [1,2] technology was proposed to solve the dilemma faced by traditional networks. NFV is a promising and critical technology for future network service providers [3]. Through software and hardware decoupling and functional abstraction, network device functions no longer rely on dedicated hardware. The hardware resources in the network can be fully and flexibly shared. In addition, operators can realize the rapid development and deployment of new services. Based on actual business needs, multiple virtual network functions (VNFs) are grouped into service function chains (SFCs) [4-8] in a predefined order and then deployed to the network to serve users. By running a virtual machine (VM) that performs various functions, the service providers can automatically start a VM whenever a user needs a new network function, which can reduce deployment time, capital costs and operating expenses.

An SFC is defined as a sequence of middleboxes that is traversed by given flows in a predefined order [9]. An SFC request can be abstracted into a directed topology. An example of an SFC request is shown in Figure 1. The SFC consists of a service terminal, a user, and a set of VNFs in a predefined order connected by virtual network links. Usually, VNFs refer to middlebox services in the network, such as deep packet inspection (DPI), firewalls, and gateways. In Figure 1, the two ovals represent the service terminal and the user. The hexagons represent the VNFs. VNFs are connected by directed virtual network links.

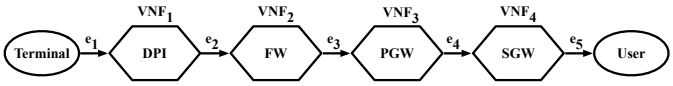

Fig. 1. Example of an SFC request.

SFC deployment is one of the key technologies affecting the performance of NFV. We need to find a path between the service terminal and the user that satisfies the requested resource constraints to deploy VNFs and virtual network links in the underlying physical network. Deploying a VNF requires a certain amount of CPU resources, and deploying a virtual network link consumes a certain amount of bandwidth resources. Different path selections will cause different end-to-end delays. Therefore, the deployment path choice affects the performance of the SFC. In the process of SFC deployment, many indicators need to be considered simultaneously, such as resource consumption, end-to-end delay, and load balancing. With the expansion of the network scale and the increase in SFC requests, ensuring successful SFC deployment is a considerable challenge. Many studies have shown that the SFC deployment problem is an NP-hard problem [9-11]. There is no polynomial time algorithm to solve the problem. Usually, an efficient heuristic algorithm is used to obtain an approximate solution.

Recently, there have been many academic studies on how to deploy the SFC. Liu et al. [9] proposed a two-step deployment approach, first deploying VNFs and then finding paths to deploy virtual network links between deployed nodes. They used a greedy algorithm to find the initial deployment scheme, and a simulated annealing algorithm was used to optimize the deployment scheme based on the greedy algorithm. The author of [12] proposed the middlebox placement optimization (MPO) algorithm, which used the ordering of the underlying topology and the SFC to optimize the end-to-end delay of SFC deployment.

Gang Sun, Zhu Xu and Hongfang Yu are with Key Lab of Optical Fiber Sensing and Communications (Ministry of Education), University of Electronic Science and Technology of China, Chengdu, China (e-mail: gangsun@uestc.edu.cn; 2215766944@qq.com; yuhf@uestc.edu.cn).

Xi Chen is with School of Computer Science and Technology, Southwest Minzu University, Chengdu, China (e-mail: chullsea@163.com).

Victor Chang is with School of Computing & Digital Technologies, Teesside University, Middlesbrough, UK (e-mail: victorchang.research@gmail.com).

Athanasios V. Vasilakos is with Department of Computer Science, Electrical and Space Engineering, Lulea University of Technology, Sweden (e-mail: athanasios.vasilakos@ltu.se).

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/JIOT.2019.2937110, IEEE Internet of Things Journal

A. Motivation

Few previous researches have simultaneously considered resource consumption and end-to-end delay in SFC deployment. The increase in resource consumption can cause network congestion and increase operating costs, and the increase in end-to-end delay can seriously affect network performance and user experiences. In this paper, we focus on the SFC deployment problem. More specifically, we propose an SFC deployment optimization (SFCDO) algorithm to optimize both resource consumption and end-to-end delay. The algorithm first uses a BFS to find the shortest path between the terminal and the user. Then, based on the shortest path, the path with the fewest hops is preferentially chosen to implement the SFC deployment. In addition, we also consider the load balancing problem to improve the reliability of the network. As a result, the proposed algorithm can improve the performance of the network, effectively reducing the deployment cost and the end-to-end delay.

B. Research Contributions

Our main contributions are described as follows:

- We build a mathematical model of the SFC deployment problem and propose an SFCDO algorithm that optimizes resource consumption and end-to-end delay of the deployment path.
- In our algorithm design process, the load balancing problem of network nodes is considered. When selecting the physical node to deploy the VNF, we design the optimal selection factor (OSF) to select the node with the lowest current load rate, distribute the load evenly on all nodes, and achieve load balancing.
- We implement our proposed approach and compare it with the existing algorithms through experiment. Then, we analyze our algorithm's performance with the compared algorithms.

C. Structure of this Paper

The remainder of this paper is organized as follows. In Section II, we review the related work. In Section Ⅲ, we describe the problem in this research with some formulations. In Section IV, we propose our heuristic algorithm. A performance evaluation of our proposed algorithm is presented in Section V, and Section VI concludes this work.

II. Related Work

A. SFC deployment for optimizing resource consumption

Resource consumption is an important indicator for measuring the benefits and drawbacks of the SFC deployment algorithm. The optimization of resource consumption can reduce network congestion and operating costs. To reduce resource consumption and the cost of SFC deployment, Huang et al. [13] studied service chain deployment by exploiting two types of correlations between network functions and devised an approximation algorithm based on the Markov approximation technique to decrease the implementation cost. Liu et al. [14] studied how to adjust the SFC deployment when the user requests dynamic changes, especially when the user moves. They established an integer linear programming model and a column generation model to optimize the node resources and bandwidth resources consumed by the deployment. Sun et al. [15] proposed a reliability-aware SFC deployment algorithm to select a less reliable deployment solution to achieve smaller deployment costs and obtain greater benefits while ensuring user reliability requirements. Liang et al. [16] proposed a dynamic orchestration mechanism for the SFC in hybrid NFV networks. They constructed a dynamic model SFC-D by considering selection changes and proposed an algorithm based on the Markov renewal process (MRP) to reduce the computing time.

Sun et al. [17] designed an SFC deployment for cloud-edge computing. They proposed that by combining cloud computing and edge computing, the consumption of network resources can be effectively reduced. The reorganization of the SFC was studied, effectively solving network congestion. The author of [18] modeled SFC deployment as a set coverage problem and proposed two logarithmic factor approximation algorithms. They designed an optimization algorithm specifically for a tree topology. Feng et al. [19] designed a fast approximation algorithm to minimize deployment costs and modeled the multicommodity chain flow problem on a cloud augmented graph. They proposed a queue-length-based algorithm that provides an $O(\varepsilon)$ approximation in time $O(1/\varepsilon)$. The author of [11] presented distributed service function chaining that coordinated these operations, distributed VNF instances of the same function, and selected appropriate instances from typical VNF offerings. They formulated this deployment as a mixed integer programming (MIP) model and developed a local search heuristic called Kariz. Extensive experiments demonstrated that Kariz achieved an additional cost of less than 24 percent compared with that of the MIP model.

Many researchers are studying SFC deployment in datacenter networks. The author of [20] addressed the problem of mapping SFCs across different datacenters with the objective of reducing the flow processing costs. They developed an integer linear programming formulation to optimally deploy SFCs to multiple datacenters while adhering to the datacenter's capacity constraints. A novel application-aware flow reduction (AAFR) algorithm was proposed to reduce the cost of SFC deployment. Jia et al. [21] investigated the dynamic placement of SFCs across geodistributed datacenters to serve flows between the dispersed source and destination pairs for operational cost minimization of the service chain provider over the entire system span. An efficient online algorithm that consists of two components was proposed. The author of [22] found that traffic fluctuations in large-scale datacenters (LDCs) could result in overload and underload phenomena in SFCs. They proposed a distributed approach based on the alternating direction method of multipliers (ADMM) to jointly load balance the traffic and horizontally scale up and down VNFs in LDCs with minimum deployment and forwarding costs.

Zhong et al. [23] orchestrated SFCs across multiple datacenters, with a goal to minimize the overall cost. An integer linear programming model was formulated and solved with a metaheuristic algorithm named GBAO that contained three modules. The author of [24] proposed a multiobjective genetic algorithm (GA) to dynamically forecast resource utilization and energy consumption in cloud datacenters. They formulated a multiobjective optimization problem of resource

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/JIOT.2019.2937110, IEEE Internet of Things Journal

allocation that considers the CPU and memory utilization of VMs and physical machines (PMs) and the energy consumption of the datacenter. The proposed GA forecasted the resource requirement of the next time slot according to the historical data in previous time slots. They further proposed a VM placement algorithm to allocate VMs for the next time slot based on the prediction results of the GA. The author of [25,26] also studied the deployment of SFC in data center network, and proposed corresponding algorithms to reduce the cost of deployment.

B. SFC deployment for optimizing the end-to-end delay

Many researchers are working to reduce the end-to-end delay of service chain deployments. Reducing the end-to-end delay of deployment paths can improve network performance and user experience. Qu et al. [27] established a reliability-aware and delay-constrained (READ) routing optimization framework for NFV-enabled datacenter networks. First, a mixed integer linear program model was proposed to reduce the end-to-end delay. Then, a heuristic algorithm was proposed to reduce the complexity of the algorithm. Cheng et al. [28] established the SFC deployment problem as a mixed integer nonlinear programming model. Based on the model, a heuristic algorithm was designed to reduce the complexity of the algorithm to ensure the delay constraints. Can et al. [12] proposed an MPO algorithm that utilizes the flexibility and dynamics provided by a software-defined network (SDN) and NFV. It can dynamically deploy a sequence of services in the SFC to adapt real-time changing service characters. Li et al. [29] were motivated to investigate applying the SFC in the small satellite-based software-defined satellite networks (SDSN) for service delivery. They introduced the structure of the multilayer constellation-based SDSN. In addition, they described two deployment patterns for the SFC in SDSN: the multidomain (MD) pattern and the satellite formation (SF) pattern. Two algorithms were proposed to reduce the delay and packet loss rate.

Cai et al. [30] aimed to achieve a flexible service orchestration for satellite networks with minimal end-to-end service delays. Based on the general NFV-enabled architecture, they built a time-varying satellite communication network model and novel forms of SFC requests. An algorithm for effectively deploying an SFC in a satellite network was proposed. Lei et al. [31] proposed a stochastic prediction model for VNF latency using random forest technology to predict the processing time and queuing time of VNFs and finally optimized the end-to-end delay.

The author of [32] studied SFC deployments in the cloud network infrastructure using the multiaccess edge computing (MEC) standard for accommodating mission critical and delay sensitive traffic. They aimed to minimize the end-to-end communication delay while keeping the overall deployment cost minimal. Yang et al. [33] considered SFC deployment based on realistic topology sensing in a fifth-generation cloud-radio access network (C-RAN). The partial observation Markov decision process (POMDP) was used to estimate the whole real topology condition. They proposed a POMDP-based SFC deployment scheme and a point-based mingled heuristic value iteration algorithm to maximize the utility associated with the total delay.

C. SFC deployment for load balancing

Implementing load balancing can effectively prevent bottleneck links or bottleneck nodes from appearing in the network. The author of [34] considered network load balancing and server load balancing when researching SFC deployment. They proposed a two-phase algorithm, nearest first and local-global transformation (NF-LGT) in the datacenter network environment. Fei et al. [35] proposed deploying VNFs in geodistributed central offices (COs). They first selected a set of central offices that minimized the communication cost among the selected COs. Then, they employed a shadow-routing-based approach, which minimized the maximum of appropriately defined CO utilizations, to jointly solve the VNF-CO and VNF-server assignment problem.

Hu et al. [36] proposed an SFC runtime framework NFCompass that uses SFC reorganization technology and task scheduling technology based on graph partitioning. They ultimately reduced the length and complexity of the processing SFC and achieved better load balancing. The author of [37] considered SFC deployment in a self-organizing SDN-NFV network. They introduced a new dynamic fine-grained function placement and migration mechanism. The designed algorithm considers load balancing and optimized fault tolerance and avoids network congestion.

III. PROBLEM STATEMENT AND FORMULATION

A. Physical network model

The physical network is the underlying network responsible for mapping the SFC. A physical network is usually composed of a set of servers connected to the switches and physical network links. The server has a certain computing resource, and the link has a certain bandwidth resource. In the process of modeling, we abstract servers as nodes and physical links as links in the topology.

We model the physical network as $G_P = (N_P, E_P)$, where $N_P = \{n_1, n_2, \dots, n_{|NP|}\}$ is the set of network nodes and $E_P = \{l_1, l_2, \dots, l_{|EP|}\}$ is the set of network links. $|NP|$ refers to the number of network nodes in the network, and $|EP|$ refers to the number of physical links. A network node n_i usually refers to a server with a certain computing resource $a(n_i)$. We use $c(n_i)$ to represent the remaining computing resources of the node, and $b(n_i)$ denotes the load rate of the node. The formula for calculating the load rate of node $b(n_i)$ is:

$$b(n_i) = \frac{a(n_i) - c(n_i)}{a(n_i)} \quad \forall\, n_i \in N_P \tag{1}$$

For a physical link l_i, we use $a(l_i)$ to represent all bandwidth resources, and $c(l_i)$ to represent the remaining bandwidth resources. $b(l_i)$ is used to indicate the load rate of the link. The formula for calculating the link load rate $b(l_i)$ is:

$$b(l_i) = \frac{a(l_i) - c(l_i)}{a(l_i)} \quad \forall\, l_i \in E_P \tag{2}$$

In addition, we use $p(n_i, n_j)$ to represent a path from node n_i to node n_j, where $p(n_i, n_j)$ is a subset of E_P that contains all the links on a path from node n_i to n_j. These are shown in Formula (3). We assume that the nodes connected at both ends

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/JIOT.2019.2937110, IEEE Internet of Things Journal

of the link l_i are denoted as n_{li1}, n_{li2}. Therefore, the transmission delay of the link l_i is denoted as $d(l_i)$ or $d(n_{li1}, n_{li2})$. The end-to-end delay from n_i to n_j is equal to the sum of the delay of all physical links on this path and is denoted as $d(n_i, n_j)$. These are shown in Formulas (4) and (5). For example, if $p(n_1, n_4) = \{l_1, l_2, l_3\}, l_1, l_2, l_3 \in E_P$, then $d(n_1, n_4) = d(l_1) + d(l_2) + d(l_3)$, where $d(l_1), d(l_2)$ and $d(l_3)$ represent the delay of link l_1, l_2 and l_3, respectively.

$$p(n_i, n_j) = \{l_{m,} \dots, l_n\} \subseteq E_P \ \forall n_i, n_j \in N_P \tag{3}$$

$$d(l_i) = d(n_{li1}, n_{li2}) \ \forall l_i \in E_P \tag{4}$$

$$d(n_i, n_j) = \sum_{l_k \in p(n_i, n_j)} d(l_k) \ \forall n_i, n_j \in N_P \tag{5}$$

B. SFC request model

The SFC request is composed of a set of VNFs and links according to the actual needs of users. The VNF requests a certain computing resource, and the virtual network link requests a certain bandwidth resource. The remaining resources of the deployed node or link must be greater than the requested resources. In addition, the VNF has strict order requirements. The traffic flow must traverse from the terminal to the user in the predefined order. The SFC can be regarded as a singly linked list.

Now, we present a formal model to describe the SFC. Let $List_{SFC} = \{G_{S_1}, G_{S_2}, \dots, G_{S_{|List_{SFC}|}}\}$ denote the set of SFCs, where $|List_{SFC}|$ represents the number of SFCs. We model one SFC request as a directed weight graph $G_S = (N_S, E_S)$, where $N_S = \{vnf_1, vnf_2, \dots, vnf_{|NS|}\}$ is represented as the set of VNFs in the SFC, and $E_S = \{e_1, e_2, \dots, e_{|ES|}\}$ represents the set of virtual network links. $|NS|$ and $|ES|$ represent the number of VNFs and links in the SFC, respectively. Deploying a VNF vnf_i requires a certain amount of computing resources $r(vnf_i)$. Similarly, deploying a virtual network link e_i requires a certain amount of bandwidth resources $r(e_i)$. Each SFC has a known source node and destination node, denoted by S and D, respectively. The source node and the destination node represent the terminal and the user, respectively. Besides, VNFs should be traversed in the predefined order. We denote it as $C_{OR} = \{vnf_1 \to vnf_2 \to \cdots \to vnf_{|NS|}\}$.

C. SFC deployment

In our problem setting, the underlying physical network and SFC request information are given as inputs. Through the heuristic algorithm, a physical path that satisfies the resource constraint is outputted to deploy the SFC. In other words, SFC deployment finds some physical nodes to deploy the VNFs and some links to map the virtual network links between the known source node and the destination node. However, the quality of the deployment path has a great impact on deployment costs and the end-to-end delay. Our goal is to find an optimal deployment scheme that minimizes total end-to-end delay and total bandwidth consumption.

For an SFC $G_S = (N_S, E_S)$, we denote $\mathrm{DS} = \{\mathrm{DS_N}, \mathrm{DS_E}\}$ as the scheme of SFC deployment, where $\mathrm{DS_N} = \{\mathrm{DS_N}(vnf_1), DS_N(vnf_2), \dots, DS_N(vnf_{|NS|})\}$ records the deployment scheme of VNFs and $\mathrm{DS_E} = \{\mathrm{DS_E}(e_1), \mathrm{DS_E}(e_2), \dots, \mathrm{DS_E}(e_{|ES|})\}$ records the deployment scheme of virtual network links. The SFC deployment procedure can be formulated as follows.

(1) *VNF deployment*

The deployment process of VNFs ca be formulated as follows:

$$DS_N: N_S \xrightarrow{DS_N} N' \tag{6}$$

$$DS_N(vnf_i) \in N' \quad \forall vnf_i \in N_S \tag{7}$$

$$c(DS_N(vnf_i)) \geq r(vnf_i) \quad \forall vnf_i \in N_S \tag{8}$$

In Formula (6), $N' \subset N_P$ denotes the set of physical network nodes which host all VNFs. As shown in Formula (7), $DS_N(vnf_i)$ records the physical node that host the VNF vnf_i. In the process of deployment, since the physical network resources are limited, some resource constraints must be met. For the deployment of VNFs, Formula (8) is a constraint to ensure that the computing resources requested by the VNF cannot be greater than the remaining computing resources of the physical node.

$$Z(vnf_i, n_j) \in \{0,1\} \ \forall vnf_i \in N_S, \forall n_j \in N_P \tag{9}$$

$$\sum_{j=1}^{|NP|} Z(vnf_i, n_j) = 1 \ \forall vnf_i \in N_s \tag{10}$$

$$\sum_{i=1}^{|NS|} Z(vnf_i, n_j) = 1 \ \forall n_j \in N_P \tag{11}$$

$$\sum_{G_S \in List_{SFC}} \sum_{vnf_i \in N_S} Z(vnf_i, n_j) \times r(vnf_i) \leq a(n_j) \ \forall n_j \in N_S \tag{12}$$

Formula (9) indicates that $Z(vnf_i, n_j)$ is a binary variable that can be equal to only 0 or 1. If $Z(vnf_i, n_j) = 1$, the i-th VNF vnf_i is deployed on the physical node n_j, otherwise $Z(vnf_i, n_j) = 0$. Formulas (10) and (11) guarantee that a VNF can be deployed on only one physical network node and that a physical network node only hosts one VNF for an SFC. Formula (12) ensures that the request resources of all VNFs deployed on the node n_j can not exceed all resources of this node.

(2) *Virtual network link deployment*

$$DS_E: E_S \xrightarrow{DS_E} E' \tag{13}$$

$$DS_E(e_i) \in E' \ \forall e_i \in E_S \tag{14}$$

$$\min_{l_j \in DS_E(e_i)} \{c(l_j)\} \geq r(e_i) \ \forall e_i \in E_S \tag{15}$$

In Formula (13), E' denotes the set of physical paths for hosting all virtual network links and each physical path is a subset of E_S. As shown in Formula (14), $DS_E(e_i)$ records the physical path that host the virtual network link e_i. For the deployment of the virtual network link, the bandwidth resource requested by the virtual network link cannot be greater than the remaining bandwidth resources of the physical link, which is described in Formula (15).

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/JIOT.2019.2937110, IEEE Internet of Things Journal

$$Y(e_i, l_j) \in \{0,1\} \ \forall e_i \in E_S, \forall l_j \in E_P \quad (16)$$

$$\sum_{G_S \in List_{SFC}} \sum_{e_i \in E_S} Y(e_i, l_j) \times r(e_i) \le a(l_j) \ \forall l_j \in E_P \quad (17)$$

$$P_{succ} = NUM_{succ} / |List_{succ}| \quad (18)$$

Formula (16) indicates that $Y(e_i, l_j)$ is a binary variable that can be equal to only 0 or 1. If $Y(e_i, l_j) = 1$, the i-th virtual network link e_i is deployed on the physical link l_j, otherwise $Y(e_i, l_j) = 0$. However, different from node deployment, since a virtual network link can map multiple physical links, the sum $Y(e_i, l_j)$ of is not required to be 1. Formula (17) ensures that the request bandwidth resources of all virtual network links deployed on the physical link l_j can not exceed all bandwidth resources of this link. Finally, we define the success rate of the SFC deployment, which is equal to the number of successfully deployed SFCs divided by the number of SFCs in a set. As shown in Formula (18), NUM_{succ} represents the number of successfully deployed SFCs. P_{succ} is the success rate of the SFC deployment.

(3) *SFC deployment example*

We show an example of SFC deployment in Figure 2. An SFC request is shown in Figure 2(a); the source node is A, and the destination node is G. The SFC contains two VNFs. Each of them has a certain computing resource request. The VNFs are connected by a directed virtual network link, and each virtual network link has a certain bandwidth resource request.

In Figure 2(b), we assume that the underlying network resources meet the resource constraints, and show two different deployment schemes marked with red dashed lines and blue dashed lines. The bandwidth consumption of the red path is $5 + 7 + 8 + 8 = 28$, and the bandwidth consumption of the blue path is $5 + 7 + 8 = 20$. In addition, since the red path has more hops than those of the blue path, the end-to-end delay of the red path is greater than that of the blue path. This example explains that deploying different paths, and especially the hop count of the path, has a large impact on the cost and the end-to-end delay. Therefore, it is important to investigate this problem and design an algorithm that outputs an optimized SFC deployment scheme.

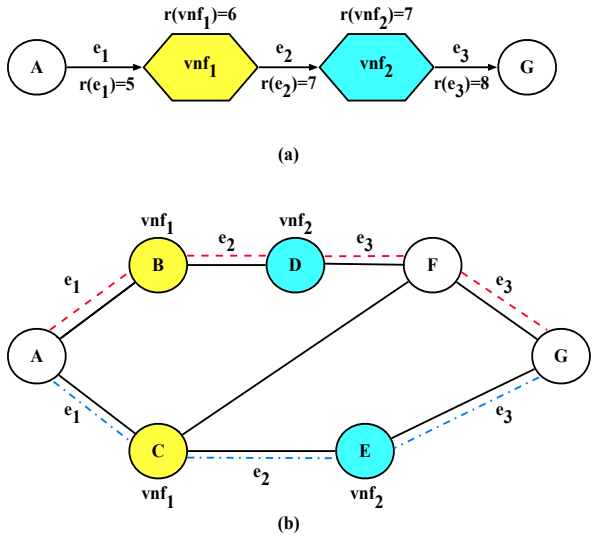

Fig. 2. Example of an SFC deployment.

D. Optimization goals

In this paper, we explore three performance metrics: the end-to-end delay, bandwidth consumption and the load rate of nodes. The end-to-end delay reflects the performance of the network application and affects the user's experience, so we expect it to be minimized. Network operators, except the total bandwidth consumption, are minimized to prevent network congestion and reduce operating costs. When the network load is concentrated on some nodes, bottleneck nodes occur in the network, which affects the deployment of the following SFC. Therefore, we distribute the load across all nodes as much as possible for load balancing.

(1) *The end-to-end delay*

The end-to-end delay of the deployment path can be formulated as follows:

$$Delay_{G_S} = \sum_{e_i \in E_S} \sum_{l_j \in DS(e_i)} d(l_j) \ \forall G_S \in List_{SFC} \quad (19)$$

$$Delay_{tot} = \sum_{G_S \in List_{SFC}} \sum_{e_i \in E_S} \sum_{l_j \in DS(e_i)} d(l_j) \quad (20)$$

In Formula (19), on the left side of the equal sign, $Delay_{G_S}$ represents the end-to-end delay of the entire SFC. On the right side of the equal sign, the inner summation represents the end-to-end delay of the deployment path on which the virtual network link e_i is deployed. The end-to-end delay of the entire chain is equal to the sum of the delay of all deployment paths. In Formula (20), $Delay_{tot}$ represents the total end-to-end delay of a set of SFCs. On the right side of the equal sign is the sum of the end-to-end delay of every SFC in the set $List_{SFC}$. One of our goals is to minimize $Delay_{tot}$.

(2) *Bandwidth resource consumption*

The calculation expression for bandwidth resource consumption is similar to the end-to-end delay:

$$Band_{G_S} = \sum_{e_i \in E_S} \sum_{l_j \in DS(ei)} r(e_i) \ \forall G_S \in List_{SFC} \quad (21)$$

$$Band_{tot} = \sum_{G_S \in List_{SFC}} \sum_{e_i \in E_S} \sum_{l_j \in DS(e_i)} r(ei) \quad (22)$$

In Formula (21), on the left side of the equal sign, $Band_{G_S}$ represents the whole bandwidth consumption of one SFC. On the right side of the equal sign, the inner summation represents the bandwidth consumption of the deployment path on which one virtual network link is deployed. The bandwidth consumption of the entire chain equals the sum of the bandwidth consumption of all deployment paths. In Formula (22), $Band_{tot}$ represents the total bandwidth consumption of a set of SFCs. On the right side of the equal sign, it is the sum of the bandwidth consumption of every SFC in the set $List_{SFC}$. One of our goals is to minimize $Band_{tot}$.

(3) *The load rate of the nodes*

Here, we mainly consider the load rate of nodes in the physical network.

$$Load_{node} = \max_{n_i \in N_P} \{b(n_i)\} \quad (23)$$

Formula (23) represents the maximum load rate of the nodes in the physical network. As shown in the formula, the maximum node load rate is equal to the maximum load rate of all nodes in the physical network.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/JIOT.2019.2937110, IEEE Internet of Things Journal

In summary, our optimization goal is to minimize end-to-end delay and bandwidth consumption and to minimize the maximum load rate of nodes. In addition, the symbols used in the problem statement and formulation are summarized in Table 1.

TABLE 1
The Symbols Used in the Problem Statement and Formulation

Symbol	Description
G_P	The physical network
N_P	The set of the physical nodes in the network
E_P	The set of the physical links in the network
$\|NP\|$	The number of physical nodes
$\|EP\|$	The number of physical links
$a(n_i)$	All computing resources of the node n_i
$c(n_i)$	The remaining resources of the node n_i
$b(n_i)$	The load rate of the node n_i
$a(l_i)$	All bandwidth resources of the link l_i
$c(l_i)$	The remaining resources of the link l_i
$b(l_i)$	The load rate of the link l_i
$p(n_i, n_j)$	A path from node n_i to node n_j
$d(n_i, n_j)$	The delay from node n_i to node n_j
$d(l_i)$	The delay of the link l_i
$d(n_{li1}, n_{li2})$	The delay of the link (n_{li1}, n_{li2})
$List_{SFC}$	The set of SFCs
G_S	The SFC request
N_S	The set of VNFs
E_S	The set of virtual network links
$\|NS\|$	The number of VNFs in the SFC
$\|ES\|$	The number of virtual links in the SFC
$\mathrm{r}(vnf_i)$	The requested computing resources for vnf_i
$\mathrm{r}(e_i)$	The requested bandwidth resources for e_i
S	The source node for an SFC
D	The destination node for an SFC
C_{OR}	The order constraints of VNFs
DS	The scheme of SFC deployment
DS_N	The scheme of VNF deployment
DS_E	The scheme of virtual network link deployment
$DS_N(vnf_i)$	The physical node on which vnf_i is deployed
$DS_E(e_i)$	The set of physical links on which e_i is deployed
$Z(vnf_i, n_j)$	A binary variable, if vnf_i is deployed on node n_j, $\mathrm{Z}(vnf_i, n_j) = 1$, otherwise $\mathrm{Z}(vnf_i, n_j) = 0$
$Y(e_i, l_j)$	A binary variable, if e_i is deployed on the link l_j, $Y(e_i, l_j) = 1$, otherwise $Y(e_i, l_j) = 0$
NUM_{succ}	The number of successfully deployed SFCs
P_{succ}	The success rate of the SFC deployment
$Delay_{tot}$	The end-to-end delay for an SFC
$Band_{tot}$	The bandwidth consumption for an SFC
$Load_{node}$	The maximum node rate for all nodes

IV. ALGORITHM DESIGN

As seen from the previous examples, the performance of SFC deployment is largely determined by the length of the deployment path. Therefore, we designed an algorithm to deploy the SFC based on the shortest path between the source node and the destination node. The algorithm is divided into two phases. The first phase is the sequence traversal of the network topology based on a BFS. The length of the shortest path between the source node and the destination node is found. Then, we compare the length of the shortest path with the length of the SFC. Three different comparison results are obtained. Different deployment strategies are adopted based on the comparison results.

A. Breadth-first search

The algorithm introduced in this section calls the BFS algorithm between the source node where the service terminal is located and the destination node where the user is located. It realizes the sequence traversal between the source node and the destination node, generates a breadth-first tree, and finds the length of the shortest path between two nodes. The details are shown in *Algorithm 1*.

Algorithm 1 requires the input of information of the physical network topology, which is usually stored in an adjacency list. In addition, the source and destination nodes of an SFC are known. This algorithm outputs the length of the shortest path between the source node and the destination node. The two-dimensional node distribution of different hops between the two nodes is also outputted.

During the initialization process, we set up $queue$ to follow the first-in, first-out criteria and set it as empty. In addition, we initialize a two-dimensional list $list_1$, a one-dimensional list $list_2$, and two count variables con_1, con_2. con_1 is used to record the number of nodes in the current hop, and con_2 records the number of nodes in the next hop. $length$ is used to record the length of the shortest path between the source and destination nodes.

Line 2 pushes the source node S into $queue$. When $queue$ is not empty, we iteratively search for the next physical node to deploy the VNF in the network topology. Line 4 obtains a node from the head of $queue$ and marks it as T. After this, add T into $list_2$ and mark the node T as already visited. con_1 is decremented by 1. If T is the destination node, it represents that we found the shortest path between the source node and the destination node. Then, $list_2$ is added into $list_1$, and $length$ is decremented by 1. After doing this, *Algorithm 1* completes the task and returns $length$ and $list_1$. If T is not the destination node, we need to traverse the adjacency nodes of T. If the adjacency node V of T has not been visited, V is put into the head of $queue$, and then con_2 is incremented by 1.

If con_1 is equal to 0, it means that all the nodes in the current hop have been traversed. $list_2$ is added to $list_1$, and then $list_2$ is set as empty to record the nodes in the next hop. Let con_1 be equal to con_2 and con_2 be set to 0. $length$ is incremented by 1.

***Algorithm* 1:** Breadth-First Search based Algorithm (BFS)

Input: (1) Physical network $G_P = (N_P, E_P)$.
(2) The source node S.
(3) The destination node D.

Output: The length of the shortest path between two nodes; the two-dimensional list $List < List < n_i >>$ that records the distribution of nodes with different hops.

1: **Initialization:** $queue = \emptyset, List < List < n_i >> list_1 =$

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/JIOT.2019.2937110, IEEE Internet of Things Journal

$\emptyset, List < n_i > list_2 = \emptyset, con_1 = 1, con_2 = 0, length = 0$;
2: push S into the $queue$;
3: **while** $queue \neq \emptyset$, **do**
4: Remove a node from the queue and mark it as T;
5: Add T into $list_2$;
6: Mark T as already visited;
7: $con_1 = con_1 - 1$;
8: **if** T is D, **do**
9: Add $list_2$ into $list_1$;
10: $length = length + 1$;
11: return $length$ and $list_1$.
12: **end if**
13: **for** the node $V \in$ T neighbor nodes, **do**
14: **if** V is not visited, **do**
15: Push V into the $queue$;
16: $con_2 = con_2 + 1$;
17: **end if**
18: **end for**
19: **if** $con1$ equals 0, **do**
20: Add $list_2$ into $list_1$;
21: $list_2 = \emptyset$;
22: $con_1 = con_2$;
23: $con_2 = 0$;
24: $length = length + 1$;
25: **end if**
26: **end while**

To facilitate a better understanding of the algorithm, we demonstrate an example of a BFS based algorithm in Figure 3. The example shows the state of the underlying network topology and the key parameters in the algorithm after each iteration. The simple network topology consists of 6 physical nodes and 7 physical links. We perform a BFS based algorithm between the source node s and the destination node d. Figure 3(a) shows that after the initialization process, the source node s is added into $queue$, and all parameters are set to their initial values. In Figure 3(b), the node s is removed from $queue$ and marked as already visited in the topology (whenever a node is removed from $queue$, we mark it as already visited and mark its hop count in the topology. For example, we mark s as 1 in the topology). We can discover the nodes r and w by traversing the adjacent nodes of s and adding them into $queue$. Since con_1 is equal to 0, the algorithm performs lines 19 to 25 to update the state of these parameters.

In Figure 3(c), node r is removed from $queue$ and added to $list_2$. No new nodes are found by traversing the adjacent nodes of node r. In Figure 3(d), the current node is w; we can find nodes t and x by traversing the adjacent nodes of node w. Since con_1 is equal to 0, the algorithm performs lines 19 to 25 to update the state of these parameters. Next, node t is removed from $queue$. After adding t into $list_2$, we scan its adjacent nodes and discover node d. Node d is added into $queue$ and con_2 is incremented by 1. These are all shown in Figure 3(e). In Figure 3(f), x is removed from $queue$. Since con_1 is equal to 0, the algorithm performs lines 19 to 25 to update the state of these parameters. In the last figure, node d is removed from $queue$. Because d is the destination node, we add $list_2$ into $list_1$ and $length$ is incremented by 1. Finally, $list_1$ and $length$ are returned.

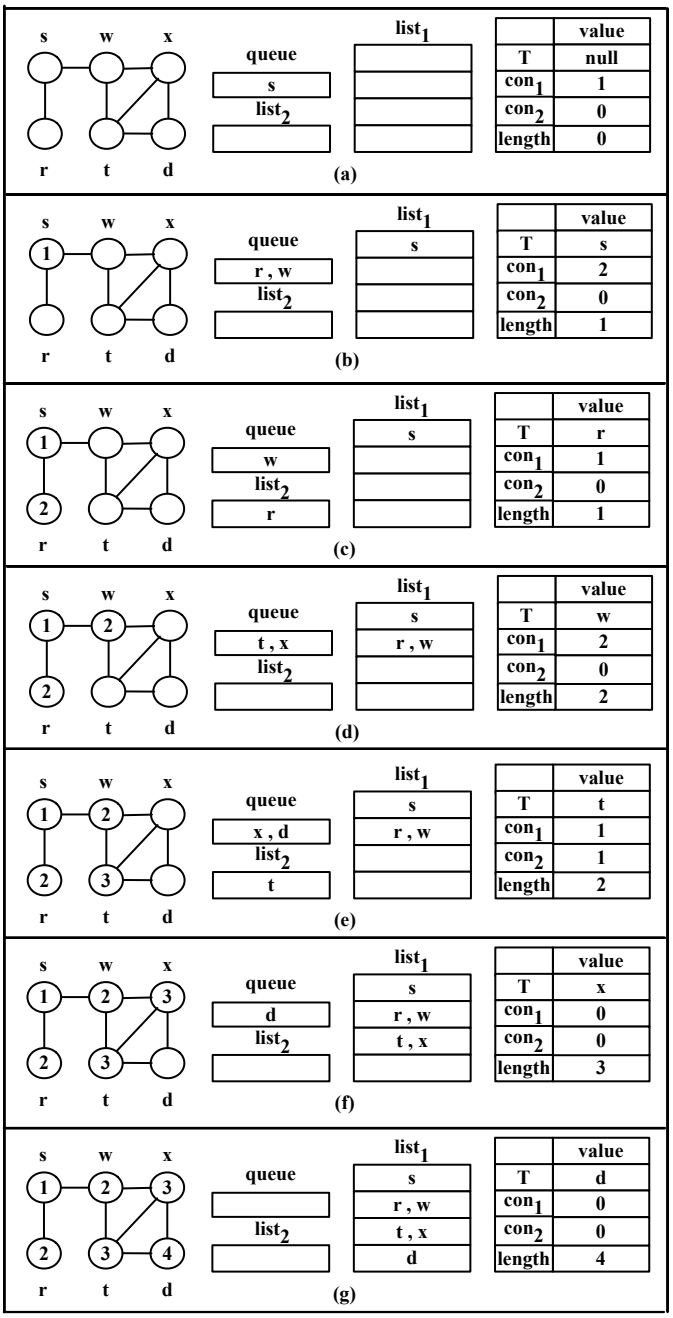

Fig. 3. An example of BFS.

After ensuring that the physical network topology is connected, by calling the BFS algorithm between the source node and the destination node, we can easily find the length of the shortest path between the source node and the destination node. The node distribution of different hops between two nodes can be obtained. We use these output results to introduce the strategy of SFC deployment in *Algorithm 2*.

B. SFC deployment based on BFS

Through *Algorithm 1*, we can obtain the length of the shortest path between the source node and the destination node and the node distribution of different hops between the two nodes. By comparing the length of the shortest path with the

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/JIOT.2019.2937110, IEEE Internet of Things Journal

length of the SFC, we can obtain three different results; namely, the length of the shortest path is equal to, less than or greater than the length of the SFC. Based on these three different results, three different deployment scenarios can be generated. In the process of deployment, we start from the layer where the destination node is located and iteratively search for the next physical network node in the upper layer or the current layer to deploy the necessary VNF. While the VNF is being deployed, the connected virtual network link is also deployed.

$$OSF\ (n_k) = \lambda \times delay_{rate}(n_k) + (1-\lambda) \times \left(b(n_k) + b\ (n_c, n_k)\right) \forall n_{k \in SOC_{node}} \quad (24)$$

$$delay_{rate}(n_k) = \frac{d(n_c, n_k)}{max\{d(n_c, n_m)\}, \forall n_m \in SOC_{node}} \quad (25)$$

When selecting a physical node to deploy the current VNF that needs to be deployed, we design an optimal selection factor $OSF(n_k)$ for a physical node n_k. The calculation method of $OSF(n_k)$ is shown in Formula (24), where n_c is the last physical node that has been selected. At the beginning of the algorithm, n_c is the destination node. λ is a weight parameter. The end-to-end delay and the load rate are weighted together to obtain an optimal selection factor OSF. Increasing the value of λ can increase the weight of the end-to-end delay at the time of selection to achieve a smaller end-to-end delay. In the following experiments, we change the value of λ to observe its effect on performance.

The formula for calculating $delay_{rate}(n_k)$ is shown in Formula (25). n_m is a node in the set of candidate nodes, which is represented by SOC_{node}. The candidate nodes are the nodes that connect with node n_c in the upper layer or the current layer. $d(n_c, n_k)$ is the delay of the link that is connected by the nodes n_c and n_k, and the denominator is the maximum delay among all candidate links. $b(n_k)$ is the load rate of the node n_k, and $b(n_c, n_k)$ is the load rate of the link connected by the nodes n_c and n_k. Dividing $d(n_c, n_k)$ by the maximum delay allows the load rate to have greater weight when selecting nodes. We select the node with the minimum optimal selection factor OSF that satisfies the resource constraint in Formulas (8) and (15) to deploy the current VNF (link). More details are shown in *Algorithm 2*.

***Algorithm 2*:** SFC deployment based on a BFS

Input: (1) Physical network $G_P = (N_P, E_P)$;
(2) An SFC request $G_S = (N_S, E_S)$, the source node S and the destination node D;
(3) $length$ that records the length of the shortest path between S and D and the two-dimensional list $list1$ that records the distribution of nodes with different hops.

Output: The deployment solution DS.

Initialization: $DS = \{D\}$,$L_{path} = length$, $L_{SFC} = |NS|$, $n_s = D$;
while $n_s \neq S$, **do**
  if $L_{path} = L_{SFC}$,**do**
    Remove the last VNF in the set of VNFs N_S;
    Find a physical node n_k in the node distribution set of the previous hop that meets both node and link resource constraints and has minimal OSF;
    if find a physical node n_k, **do**
      $n_s = n_k$;
      $DS = DS \cup n_k$;
      $L_{path} = L_{path} - 1$;
      $L_{SFC} = L_{SFC} - 1$;
    else
      return DS.
    end if
  else if $L_{path} > L_{SFC}$,**do**
    Find the link e with the smallest bandwidth resource request in the E_S, and the bandwidth resource consumption is $r(e)$;
    Expand the $|L_{path} - L_{SFC}|$ links that request $r(e)$ bandwidth resources in the SFC. The node request resource between the links is set to 0. After doing this, L_{path} is equal to L_{SFC};
  else if $L_{path} < L_{SFC}$,**do**
    Remove the last VNF in the set of VNFs N_S;
    Find a physical node n_k in the node distribution set of the current hop, which meets both node and link resource constraints and has minimal OSF;
    if fine a physical node n_k,**do**
      $n_s = n_k$;
      $DS = DS \cup n_k$;
      $L_{SFC} = L_{SFC} - 1$;
    else
      Find a physical node n_k in the node distribution set of the previous hop that meets both node and link resource constraints and has minimal OSF;
      if find a physical node n_k, **do**
        $n_s = n_k$; $DS = DS \cup n_k$;
        $L_{path} = L_{path} - 1$;
        $L_{SFC} = L_{SFC} - 1$;
      else
        return DS.
      end if
    end if
  end if
end while
return DS

Algorithm 2 requires the inputs of the underlying physical network topology and an SFC request containing the source node and the destination node. The results derived by *Algorithm 1* containing the length of the shortest path and the node distribution with different hops between the source node and the destination node is also needed. Finally, the deployment scheme of this SFC is outputted. *Algorithm 2* finds a physical node to deploy the current VNF each iteration until the source node is found, or the algorithm ends because there are not enough underlying resources.

We compare the size of L_{path} and L_{SFC} each time. According to three different comparison results, three different deployment scenarios are obtained. When L_{path} is equal to L_{SFC}, we look for a physical node in the node distribution of the previous hop to deploy the current VNF. The node is required to satisfy node and link resource constraints and has a

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/JIOT.2019.2937110, IEEE Internet of Things Journal

minimum OSF.

If L_{path} is greater than L_{SFC}, the length of the shortest path will be greater than the length of the SFC. In this case, there are redundant nodes and links. Therefore, we extend the length of the SFC, adding nodes and links so that L_{path} is equal to L_{SFC}. The added virtual links need to consume a certain amount of bandwidth resources, so we choose to expand the virtual network link requesting the least bandwidth resources in the SFC. The added virtual network nodes do not need to consume redundant computing resources.

When L_{path} is smaller than L_{SFC}, the current SFC cannot be deployed in the shortest path. We need to find other nodes between the nodes in the shortest paths. In each iteration, we first try to find the physical node in the node distribution of the current hop and then find the physical node in the node distribution of the previous hop. DS is used to record the deployment scheme. After the algorithm ends, we can compare the SFC and the deployment scheme to determine whether the deployment is successful, and the resource consumption caused by the deployment is calculated.

To facilitate a better understanding of the algorithm, we demonstrate an example of SFC deployment in Figure 4. Figure 4(a) shows the outputs of *Algorithm 1*, including information about the underlying physical topology, the length of the shortest path between nodes s and d, and the node distribution of different hop counts $list_1$. Figure 4(b) shows an SFC request that contains two VNFs and three virtual network links. We show the end-to-end delay and load rate of all physical links in Figure 4(c). In Figure 4(d), the current load rate of all nodes is given.

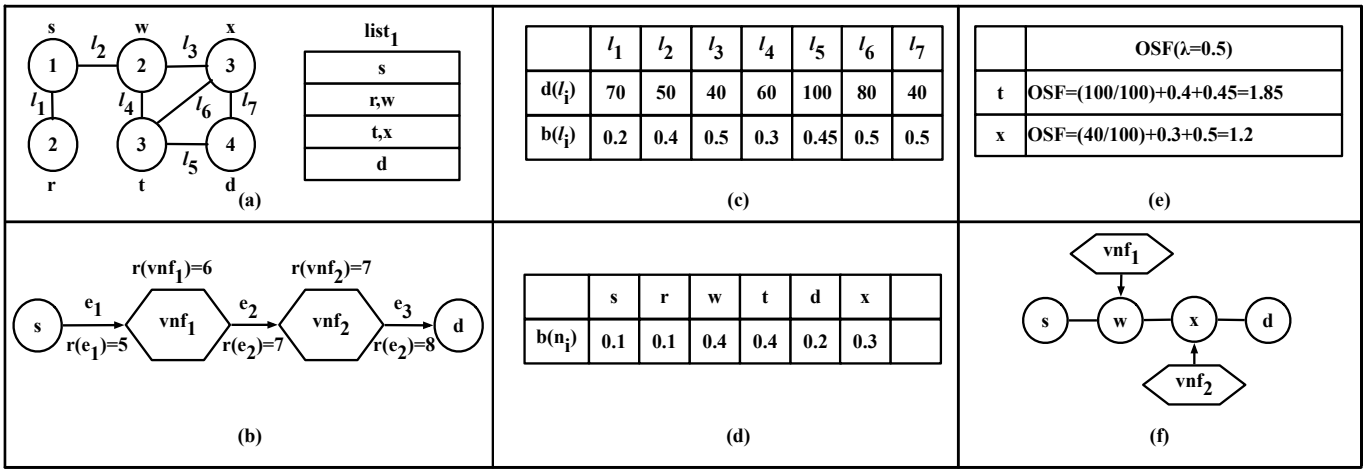

Fig. 4. An example of SFC deployment based on BFS.

To find a path between node s and d to deploy the SFC, we start from the destination node d. Here, we assume that both the underlying node resources and link resources satisfy the resource constraints. Since the length of the SFC is equal to the length of the shortest path between two nodes, we search the nodes from the upper layer of d. As you can see in $list_1$, there are two candidate nodes, t and x. The optimal selection factor OSF for the two nodes is given in Figure 4(e). The optimal selection factor of node x is smaller than that of node t, so we choose node x to deploy the current VNF vnf_2. While deploying VNFs, virtual network links are also deployed.

Next, starting from node x, we look for the next physical node to deploy $vnf1$. The candidate nodes are r and w. Since the node r is not a neighbor of node x, we can choose only node w to deploy vnf_1. Finally, we find the source node s. This SFC is deployed successfully. The deployment path of the SFC is $s \rightarrow w \rightarrow x \rightarrow d$. Figure 4(f) shows the scheme of the SFC deployment.

Algorithm 2 deploys the SFC based on a BFS. The algorithm first considers the shortest path between the source node and the destination node and preferentially selects the path with fewer hops to deploy the SFC. Because the length of the deployed path is closely related to the end-to-end delay and bandwidth consumption, the algorithm can optimize these two performance metrics. In addition, the load balancing of the network is also considered in the process of designing the algorithm. The associated algorithm experiments are described in the next section.

C. Complexity analysis

The proposed SFCDO algorithm consists of *Algorithm 1* and *Algorithm 2*. In a physical network topology, we assume that there are $|NP|$ nodes and $|EP|$ links. We analysis the time complexity of our proposed SFCDO algorithm as follows:

- *Algorithm 1* uses BFS search method between the source node and the destination node. The algorithm ensures that each node enters and pops a queue at most one time. Therefore, the total time to operate on the queue is $O(|NP|)$. We use an adjacent list to store the network topology. Because the algorithm only scans the adjacency list of the node when it is dequeued, each adjacency list is scanned at most once. The sum of the lengths of all adjacent lists is $\theta(|EP|)$, and the total time for scanning the adjacent lists is $O(|EP|)$. Therefore, the complexity of *Algorithm 1* is $O(|NP| + |EP|)$.
- For *Algorithm 2*, when $L_{path} \geq L_{SFC}$, each node and link are scanned at most once, so the algorithm complexity is $O(|NP| + |EP|)$. When $L_{path} < L_{SFC}$, the algorithm complexity is $O(2 * (|NP| + |EP|))$. Therefore, the time complexity of *Algorithm 2* is $O(|NP| + |EP|)$.

In summary, for the deployment of an SFC, the time complexity of our SFCDO algorithm is $O(|NP| + |EP|)$.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/JIOT.2019.2937110, IEEE Internet of Things Journal

V. Experiment Results and Analysis

In this section, we first introduce the experiment environment and some key parameter settings. Then, we compare the proposed algorithm with an algorithm in an existing paper. The author of [9] proposed a G-SA algorithm. The algorithm first used the greedy algorithm to initially deploy the VNFs and then used a simulated annealing algorithm to optimize the results obtained by the greedy algorithm. The algorithm in [9] is a typical two-step algorithm that first deploys VNFs and then finds paths between deployed nodes. This algorithm wasted the underlying network resources and introduced a relatively high end-to-end delay. We will show and analyze the results of the performance comparison.

A. Experiment environment and settings

In the experiment, we use OpenStack to build testbeds to evaluate different deployment algorithms. OpenStack is a cloud operating system that controls large pools of compute, storage, and networking resources. Through openstack, we can configure the resources in the underlying physical network and the request resources of the SFC. Similar with Ref. [38], we conduct the experiment on a typical Chinese network topology. An example of a physical network topology is shown in Figure 5. The network topology has 55 nodes, and the average degree of each node is about 4.

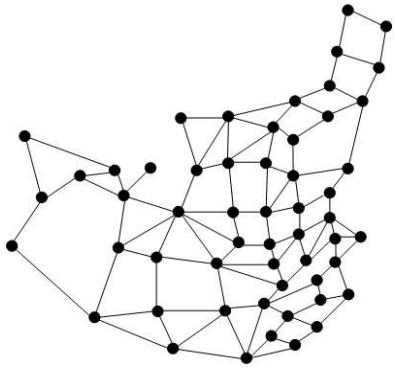

Fig. 5. An example of a physical network topology.

Similar with Ref. [17], we assume that each node has 2000 computing resources, each link has 2000 Mbps bandwidth resources, and the end-to-end delay of each link follows a uniform distribution, U (30, 130). In the experiment, we generate 500 SFC requests per group. The length of each SFC is uniformly distributed, U (4, 6). The computing resources requested by each VNF follow a uniform distribution, U (10, 20). The bandwidth resources requested by each virtual network link are subject to a uniform distribution, U (10, 20). For fairness, we use the same network topology and parameters when comparing the two algorithms.

B. Experiment results and analysis

In the process of comparing the performance of the algorithms, the main performance indicators we focus on are end-to-end delay, bandwidth consumption of deployed links, and network load balancing. Based on the environment introduced in *Part A*, we analyze the three indicators separately.

To further investigate the influence of different algorithms on the end-to-end delay, we study the distribution of the end-to-end delay of each SFC. We plot the result in Figure 6. The abscissa is the end-to-end delay, and the ordinate is the sum of the proportions of the end-to-end delay, which is less than the abscissa. As shown in the figure, the performance gap between the SFCDO algorithm and the G-SA algorithm is large. The end-to-end delay of the SFCDO algorithm is mainly concentrated between 160-690 ms, while the G-SA algorithm is mainly concentrated between 200-1130 ms. Compared to the G-SA algorithm, SFCDO has a shorter end-to-end delay, mainly because SFCDO deploys the SFC based on the shortest path between the source node and the destination node. Our proposed algorithm preferentially selects the physical path with fewer hops, so the end-to-end delay is optimized.

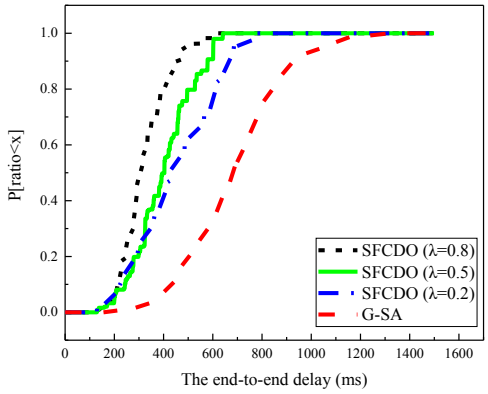

Fig. 6. The distribution of end-to-end delay.

In addition, we compare the end-to-end delay with a changing λ parameter. As the λ increases, the end-to-end delay is continuously optimized because the influence of the end-to-end delay in the optimal selection factor is increasing, so it tends to choose a link with a smaller end-to-end delay.

Figure 7 shows the distribution of bandwidth consumption. The abscissa represents the bandwidth consumption of the deployment, and the ordinate is the sum of the proportions of bandwidth consumption, which is less than the abscissa. For bandwidth consumption, the SFCDO algorithm is mainly concentrated between 35 and 120 Mbps, while the G-SA algorithm is concentrated between 40 and 302 Mbps. Because both bandwidth consumption and end-to-end delay are determined by the number of hops of the deployment path, bandwidth consumption is optimized as end-to-end delay when we prefer the path with the fewest hops to deploy the SFC.

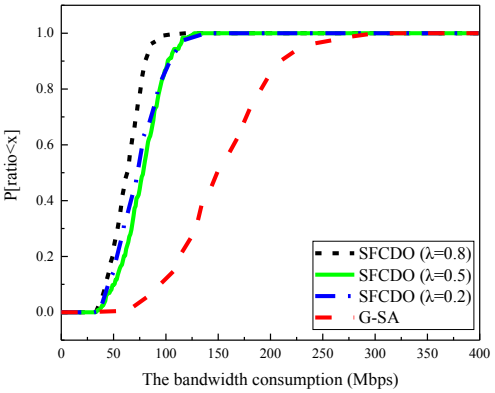

Fig. 7. The distribution of bandwidth consumption.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/JIOT.2019.2937110, IEEE Internet of Things Journal

We also observe changes in bandwidth consumption while changing the λ parameter. As shown in Figure 7, increasing the value of λ is not significant for changes in bandwidth consumption. This is because bandwidth consumption is mainly determined by the length of the deployment path and the bandwidth resources requested by the virtual network link, do not interact with λ.

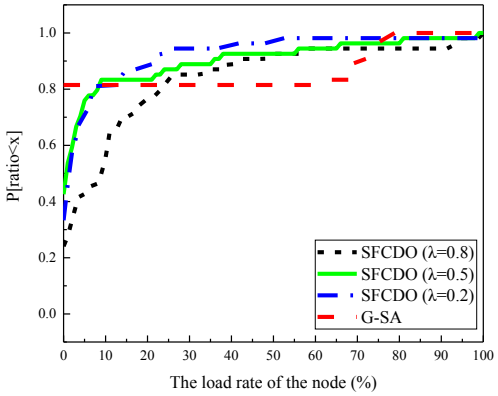

Fig. 8. The distribution of load rate of the node.

To further investigate the influence of different algorithms on the load rate, we studied the distribution of the load rate of the nodes. The result is shown in Figure 8. In the figure, the abscissa represents the load rate of the node, and the ordinate is the sum of the proportions of the load rate, which is less than the abscissa. It can be seen that 80 percent of the nodes have no load in the G-SA algorithm, and the network load is mainly concentrated on 20 percent of nodes. In the SFCDO algorithm, 95 percent of the load rate of the nodes is concentrated below 60 percent. Nodes with high load rates rarely occur because in the design process of the SFCDO algorithm, we consider the load rate of the node and use it as one of the factors for the selection of nodes. The G-SA algorithm deploys the VNF on a certain node as much as possible, resulting in excessive load concentration.

In addition, we compare the load rate with changing λ parameters. As λ decreases, the load rate is continuously optimized because the influence of the load rate in the optimal selection factor is increasing, so it tends to choose a node with a smaller load rate.

In addition to performing the experiment in the environment described in section A, we also attempt experiments under different network parameters. To investigate the adaptability of the two algorithms to different network parameters, we change the number of SFCs and the length of the SFCs. We compare and analyze the performances of the two algorithms. The results show that the proposed algorithm is still optimized in terms of end-to-end delay and bandwidth resource consumption. The rest of the experiment results are shown.

Figure 9 is a diagram showing the change in the average end-to-end delay of the deployment path in the case of changing the number of SFCs. In the figure, the abscissa represents the number of SFCs, and the ordinate represents the average end-to-end delay. As the number of SFCs increases, the SFCDO algorithm has good stability. The average end-to-end delay fluctuates within only a small range. The average end-to-end delay is maintained at approximately 395 ms. However, for the G-SA algorithm, the average end-to-end delay increases as the number of SFCs increases. The average end-to-end delay of the G-SA algorithm is always greater than that of the SFCDO algorithm. In addition, we used different λ parameters to observe the average end-to-end delay variation. As shown in Figure 9, as the λ parameter increases, the average end-to-end delay decreases, which shows that we can obtain a smaller end-to-end delay by adjusting the λ parameter.

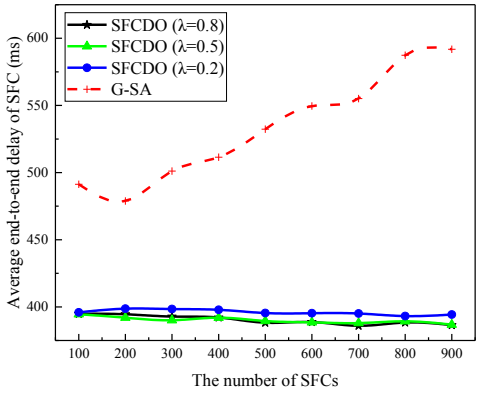

Fig. 9. Average end-to-end delay for different numbers of SFCs.

We show the change in average bandwidth consumption in the case of changing the number of SFCs in Figure 10. In the figure, the abscissa represents the number of SFCs, and the ordinate represents the average bandwidth consumption. It can be seen that with the increase in the number of SFCs, the bandwidth consumption of the SFCDO algorithm remains basically unchanged, and average end-to-end delay is maintained at approximately 72 Mbps. However, for the G-SA algorithm, average bandwidth consumption increases as the number of SFCs increases. The average bandwidth consumption of the G-SA algorithm is always greater than that of the SFCDO algorithm. In addition, as shown in the figure, there is no change in the average bandwidth consumption as the λ parameter changes.

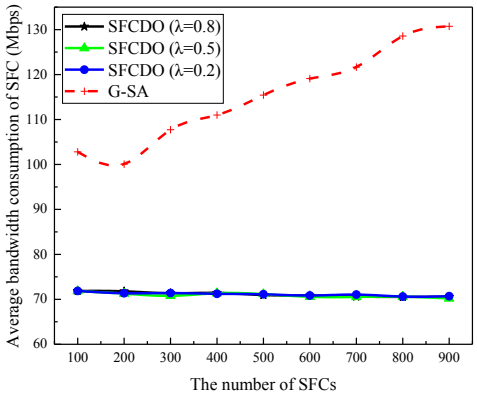

Fig. 10. Average bandwidth consumption for different numbers of SFCs.

To investigate the influence of different parameters on the performance, we study the change in the average end-to-end delay and bandwidth consumption when changing the length of the SFCs. We plot these results in Figure 11 and Figure 12, respectively. In Figure 11, the abscissa represents the length of

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/JIOT.2019.2937110, IEEE Internet of Things Journal

the SFC, and the ordinate represents the average end-to-end delay of the deployment paths. As the length of the SFC increases, the end-to-end delays of the two algorithms both increase, because as the length of the SFC increases, the length of the deployment path also increases, so the average end-to-end delay will also increase. However, as shown in the figure, the growth rate of the SFCDO algorithm is less than that of G-SA, and the end-to-end delay of SFCDO is always smaller than that of the G-SA algorithm. Therefore, the SFCDO algorithm is significantly optimized for end-to-end delay. In addition, as the λ parameter increases, the average end-to-end delay is slightly optimized.

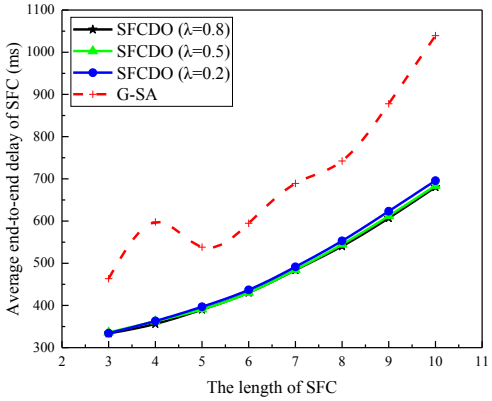

Fig. 11. Average end-to-end delay of SFC.

In Figure 12, the abscissa represents the length of the SFC, and the ordinate represents the average bandwidth consumption. As the length of the SFC increases, the bandwidth consumption of the two algorithms both increase, which is the same as the end-to-end delay. As shown in the figure, the growth rate of the SFCDO algorithm is also less than that of G-SA, and the end-to-end delay of SFCDO is always smaller than that of the G-SA algorithm. Compared with the G-SA algorithm, the SFCDO algorithm also optimizes bandwidth consumption. However, as the λ parameter increases, the average bandwidth consumption remains unchanged.

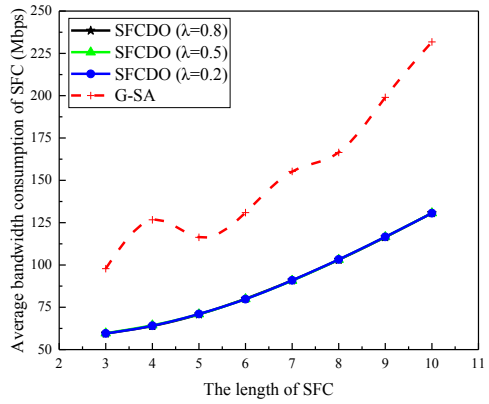

Fig. 12. Average bandwidth consumption of SFC.

VI. Conclusion

In this paper, we study the efficient SFC deployment problem in NFV. The key issue in the SFC deployment problem is how to achieve efficient use of the underlying physical resources and effectively reduce the end-to-end delay of the deployment path. We introduced recent research on service chain deployment issues and proposed mathematical models for SFC deployment. For the proposed mathematical model, we propose an SFC deployment algorithm SFCDO based on a BFS. The algorithm deploys the SFC based on the shortest path between the source node and the destination node and preferentially selects the path with the shortest hops to implement the deployment. In addition, we compare the proposed algorithm with the G-SA algorithm. The experiment results show that the SFCDO algorithm can effectively reduce the end-to-end delay of the deployment path and reduce the bandwidth resource consumption by up to 40% and 49%, respectively. In addition, the algorithm also considers the load rate of the nodes and achieves load balancing.

In the future work, we are going to study the network security problem in the service function chain deployment, and propose corresponding deployment algorithms to ensure the security of the network while further improving the utilization of network resources.

Acknowledgement

This research was partially supported by the National Natural Science Foundation of China (61571098), the 111 Project (B14039).

References

[1] Sun G, Zhu G, Liao D, et al. Cost-Efficient Service Function Chain Orchestration for Low-Latency Applications in NFV Networks. IEEE Systems Journal, Early Access, 2018.

[2] Sun G, Li Y, Liao D, et al. Service function chain orchestration across multiple domains: A full mesh aggregation approach. IEEE Transactions on Network and Service Management, 2018, 15(3): 1175-1191.

[3] Sun G, Chang V, Yang G, et al. The cost-efficient deployment of replica servers in virtual content distribution networks for data fusion. Information Sciences, 2018, 432: 495-515.

[4] Zhang S, Wang Y, Li W, et al. Service failure diagnosis in service function chain. IEEE 19th Asia-Pacific Network Operations and Management Symposium, 2017: 70-75.

[5] Phan T V, Ngo L H, Huong T T, et al. AQA 2: An analytical QoS-assessment approach for service function chaining in cloud environment. IEEE International Conference on Advanced Technologies for Communications, 2017: 21-26.

[6] Imagane K, Kanai K, Katto J, et al. Performance evaluations of multimedia service function chaining in edge clouds. IEEE Annual Consumer Communications & Networking Conference, 2018: 1-4.

[7] Gunleifsen H, Kemmerich T. Security requirements for service function chaining isolation and encryption. IEEE 17th International Conference on Communication Technology, 2017: 1360-1365.

[8] Mirjalily G, Zhiquan L U O. Optimal Network Function Virtualization and Service Function Chaining: A Survey. Chinese Journal of Electronics, 2018, 27(4): 704-717.

[9] Liu J, Li Y, Zhang Y, et al. Improve service chaining performance with optimized middlebox placement. IEEE Transactions on Services Computing, 2017, 10(4): 560-573.

[10] Sun G, Chen Z, Yu H, et al. Online Parallelized Service Function Chain Orchestration in Data Center Networks. IEEE Access, 2019,7(1): 100147-100161.
[11] Ghaznavi M, Shahriar N, Kamali S, et al. Distributed service function chaining. IEEE Journal on Selected Areas in Communications, 2017, 35(11): 2479-2489.
[12] Ouyang C, Wei Y, Leng S, et al. Service chain performance optimization based on middlebox deployment. IEEE International Conference on Communication Technology, 2017: 1947-1952.
[13] Huang H, Li P, Guo S, et al. Near-Optimal Deployment of Service Chains by Exploiting Correlations between Network Functions. IEEE Transactions on Cloud Computing, 2017.
[14] Liu J, Lu W, Zhou F, et al. On dynamic service function chain deployment and readjustment. IEEE Transactions on Network and Service Management, 2017, 14(3): 543-553.
[15] Sun G, Xu Z, Yu H, et al. Toward SLAs Guaranteed Scalable VDC Provisioning in Cloud Data Centers. IEEE Access, 2019, 7(1): 80219-80232.
[16] Liang X, Huang X, Li D, et al. Dynamic Orchestration Mechanism of Service Function Chain in Hybrid NFV Networks. IEEE Asia Communications and Photonics Conference, 2018: 1-3.
[17] Sun G, Li Y, Li Y, et al. Low-Latency Orchestration for Workflow-Oriented Service Function Chain in Edge Computing. Future Generation Computer Systems, 2018, 85: 116-128.
[18] Tomassilli A, Giroire F, Huin N, et al. Provably efficient algorithms for placement of service function chains with ordering constraints. Université Côte d'Azur, CNRS, I3S, France; Inria Sophia Antipolis, 2018.
[19] Feng H, Llorca J, Tulino A M, et al. Approximation algorithms for the NFV service distribution problem. IEEE INFOCOM, 2017: 1-9.
[20] Nadig D, Ramamurthy B, Bockelman B, et al. Optimized Service Chain Mapping and reduced flow processing with Application Awareness. IEEE Conference on Network Softwarization and Workshops, 2018: 303-307.
[21] Jia Y, Wu C, Li Z, et al. Online scaling of NFV service chains across geo-distributed datacenters. IEEE/ACM Transactions on Networking, 2018, 26(2): 699-710.
[22] Tashtarian F, Varasteh A, Montazerolghaem A, et al. Distributed VNF scaling in large-scale datacenters: An ADMM-based approach. IEEE International Conference on Communication Technology, 2017: 471-480.
[23] Zhong X, Wang Y, Qiu X. Service function chain orchestration across multiple clouds. China Communications, 2018, 15(10): 99-116.
[24] Tseng F H, Wang X, Chou L D, et al. Dynamic Resource Prediction and Allocation for Cloud Data Center Using the Multiobjective Genetic Algorithm. IEEE Systems Journal, 2018, 12(2): 1688-1699.
[25] Sun G, Liao D, Bu S, et al. The Efficient Framework and Algorithm for Provisioning Evolving VDC in Federated Data Centers. Future Generation Computer Systems, 2017, 73: 79-89.
[26] Sun G, Liao D, Zhao D, et al. Live migration for multiple correlated virtual machines in cloud-based data centers. IEEE Transactions on Services Computing, 2018, 11(2): 279-291.
[27] Qu L, Assi C, Shaban K, et al. A reliability-aware network service chain provisioning with delay guarantees in nfv-enabled enterprise datacenter networks. IEEE Transactions on Network and Service Management, 2017, 14(3): 554-568.
[28] Cheng Y, Yang L, Zhu H. Deployment of service function chain for NFV-enabled network with delay constraint. IEEE International Conference on Electronics Technology, 2018: 383-386.
[29] Li T, Zhou H, Luo H, et al. Service function chain in small satellite-based software defined satellite networks. China Communications, 2018, 15(3): 157-167.
[30] Cai Y, Wang Y, Zhong X, et al. An approach to deploy service function chains in satellite networks. IEEE/IFIP Network Operations and Management Symposium, 2018: 1-7.
[31] Lei T H, Hsu Y T, Wang I C, et al. Deploying QoS-assured service function chains with stochastic prediction models on VNF latency. IEEE Conference on Network Function Virtualization and Software Defined Networks, 2017: 1-6.
[32] Leivadeas A, Falkner M, Lambadaris I, et al. Balancing Delay and Cost in Virtual Network Function Placement and Chaining. IEEE Conference on Network Softwarization and Workshops (NetSoft), 2018: 433-440.
[33] Yang Y, Chen Q, Zhao G, et al. The Stochastic-Learning-Based Deployment Scheme for Service Function Chain in Access Network. IEEE Access, 2018, 6: 52406-52420.
[34] Thai M T, Lin Y D, Lai Y C. A joint network and server load balancing algorithm for chaining virtualized network functions. IEEE International Conference on Communications, 2016: 1-6.
[35] Fei X, Liu F, Xu H, et al. Towards load-balanced VNF assignment in geo-distributed NFV Infrastructure. IEEE/ACM 25th International Symposium on Quality of Service, 2017: 1-10.
[36] Hu Y, Li T. Enabling Efficient Network Service Function Chain Deployment on Heterogeneous Server Platform. IEEE International Symposium on High Performance Computer Architecture, 2018: 27-39.
[37] Kwang-Man K O, Mansoor A M, Ahmad R, et al. Efficient Deployment of Service Function Chains (SFCs) in a Self-Organizing SDN-NFV Networking Architecture to Support IOT. IEEE International Conference on Ubiquitous and Future Networks, 2018: 650-653.
[38] Sun G, Yu H, Anand V, et al. A cost efficient framework and algorithm for embedding dynamic virtual network requests[J]. Future Generation Computer Systems, 2013, 29(5): 1265-1277.