Hindawi
Security and Communication Networks
Volume 2017, Article ID 9534754, 17 pages
https://doi.org/10.1155/2017/9534754

WILEY | Hindawi

Research Article

A New Approach for Delivering Customized Security Everywhere: Security Service Chain

Yi Liu, Hong-qi Zhang, Jiang Liu, and Ying-jie Yang

Information Science Technology Institute, Zhengzhou, Henan 450000, China

Correspondence should be addressed to Yi Liu; liuyi9582@126.com

Received 28 July 2017; Revised 22 October 2017; Accepted 8 November 2017; Published 12 December 2017

Academic Editor: Guangjie Han

Security functions are usually deployed on proprietary hardware, which makes the delivery of security service inflexible and of high cost. Emerging technologies such as software-defined networking and network function virtualization go in the direction of executing functions as software components in virtual machines or containers provisioned in standard hardware resources. They enable network to provide customized security service by deploying Security Service Chain (SSC), which refers to steering flow through multiple security functions in a particular order specified by individual user or application. However, SSC Deployment Problem (SSC-DP) needs to be solved. It is a challenging problem for various reasons, such as the heterogeneity of instances in terms of service capacity and resource demand. In this paper, we propose an SSC-based approach to deliver security service to users without worrying about physical locations of security functions. For SSC-DP, we present a three-phase method to solve it while optimizing network and security resource allocation. The presented method allows network to serve a large number of flows and minimizes the latency seen by flows. Comparative experiments on the fat-tree and Waxman topologies show that our method performs better than other heuristics under a wide range of network conditions.

1. Introduction

Today's security service delivery approach is limited in dynamics, flexibility, scalability, and efficient resource utilization. Firstly, security services are configured in static and inflexible ways, such as deploying hardware firewall and IDS in the key position of network. They are coupled with the underlying physical topology [1], making it difficult to deliver customized security services according to user requirements and network constraints. Secondly, reconfiguring existing security service requires time-intensive manual operations, making the approach often inflexible and hard to cope with changeable requirements. Thirdly, there is a serious waste of security resources. It is inefficient for flows from multiusers or multibusinesses to share hardware-based security devices since their positions are fixed. What is worse, security devices need to work at full capacity so as to serve incoming flows, especially burst flows in time.

Recent research efforts on promising network technologies, such as software-defined networking (SDN) [2] and network function virtualization (NFV) [3], promise to revolutionize security service delivery approach. SDN decouples network control from forwarding and makes the former directly programmable [4], realizing the centralized network management. NFV moves network functions off proprietary hardware onto standard servers (e.g., ×86 based systems) in the form of virtual network function (VNF). This way of separating and abstracting functionalities from locations facilitates flexible orchestration of network functions [5]. Moreover, in the state of the art, VNFs can achieve approximate performance of hardware devices [6–8]. Together, SDN and NFV make networks and network devices agile [9].

As a consequence, the concept of Security Service Chain (SSC) [10] has been proposed, which refers to an ordered set of security functions composing a logical security service that must be applied to packets or flows. With the help of fine-grained flow management originated from SDN and flexible function orchestration originated from NFV, deploying SSC becomes a promising way to deliver security service. By placing security functions in a topology independent way,

it dynamically and flexibly adds or removes functions along the routing path of flow, thereby catering to changeable user demands and network conditions. The key problem is automatically converting abstract SSCs to the specific placement of security function instances or simply instances and routing paths of flows. We refer to this problem as the *SSC Deployment Problem* (SSC-DP). Generally, an SSC is derived from the security request of individual user or application. An instance is an operational software or hardware instance capable of delivering the treatment specified by the associated security function to packets or flows [11]. We only consider software instances, namely, virtualized security functions (e.g., virtual firewall, IPS, Web filter, and virus scanner). The server running them is called service node, which not only provides a runtime environment but also comprises facilities for attaching instances to the network.

However, several issues should be considered before solving SSC-DP due to limited network resources. First, instances belonging to the same security function may differ in service capacity. For example, the throughput of a single instance may be far less than the volume of flow which generates security request. So instances providing the same functionality should be combined to serve a big flow. However, instances may also differ in resource demand. Thus, in order to minimize resource consumption of service nodes, we need to select the optimal combination of instances and assign more flows to the instance with high service capacity. Second, instances may have different demands for various resources. For example, an instance needs two CPU cores and 4 MB memory while another consumes one CPU core and 6 MB memory. So resources on a server may have different occupancy ratios, which leads to resource fragmentation problem [12]. Specifically, as far as a single service node is concerned, if the occupancy ratio of certain resource reaches the threshold, the node cannot run new instances any more. Third, flows should be routed in such a way to follow the sequence specified by SSC while optimizing the latency of security service, since latency is an important factor in measuring network performance [13–15]. Hence, *an optimal solution of SSC-DP is needed to satisfy service demand of security request while minimizing resource consumption of service nodes and reducing resource fragmentation as well as forwarding flow through the best available path with the minimal security service latency*.

In this paper, we propose an approach that adopts the idea of SSC in the design of a solution for dynamically delivering customized security services. Since the key to effective operation of the proposed approach is to solve SSC-DP, we propose TPSSC, a three-phase method of finding near-optimal solutions of SSC-DP. Our main contributions are summarized as follows:

(i) We design an architecture to realize the idea of SSC by integrating the concepts of SDN and NFV, which facilitates security service delivery and management.

(ii) Taking into account the heterogeneity of service capacity and resource demand of miscellaneous instances, we propose the design operation before deploying SSCs to physical network. It contributes to reducing the total resource consumption of service nodes while allowing us to place instances in service nodes flexibly without worrying about service demands of security requests.

(iii) Based on considering both resource fragmentation and security service latency throughout the node mapping phase and the link mapping phase, we propose heuristic algorithms to select service nodes for instances and establish routing paths for flows. They contribute to optimizing network resource allocation and improving acceptance ratio of security requests.

The rest of this paper is organized as follows. We study the related work in Section 2. Section 3 describes the architecture of SSC-based security service delivery approach followed by illustrating the integrated ETSI NFV MANO architecture including the proposed architecture. In Section 4, we introduce some important definitions and formally define the SSC Deployment Problem. We present and evaluate the method TPSSC in Sections 5 and 6, respectively. Lastly, we conclude this paper with some future directions in Section 7.

2. Related Work

SSC-DP is similar to Virtual Network Embedding (VNE) [16] problem in some aspects, such as placing virtual network nodes (instances in our case) in physical infrastructure and chaining them while optimizing resource utilization or other objectives. However, solutions of VNE cannot be applied to solving SSC-DP directly, since the latter imposes additional constraints such as the service capacity of function specified by user's request. In other words, VNE directly maps virtual network to physical network, while SSC-DP maps SSC requests of flows to virtual network composed of instances and then maps the latter to physical network. Moreover, VNE only considers routers in physical network while SSC-DP needs to deal with a much wider number of different functions which have strict order.

Generally, SSC-DP can be regarded as a combination of VNF placement and traffic routing. A number of researches have been done in this field. Broadly we classify them into two domains as follows.

In the case that instances have been running on service nodes, researches focus on the optimal selection of instances and routing of flows. The method proposed by Dwaraki and Wolf [17] transforms the network topology to a layered graph and selects instances and routes for each flow by running the Dijkstra algorithm. But it needs to find the shortest path in large space and the storage of layered graph costs high. Worse still, big flows may be accepted early, preventing network from holding more subsequent flows. To conquer this problem, Cao et al. [18] propose an online routing algorithm which can enable network to accept flows as many as possible over time. But it does not take into account the service capacity of instance. Thus, the work by Xiong et al. [19] selects instances and routes based on the service capacities of instances and the bandwidths of physical links, respectively. But the end-to-end latency of a flow may be large resulting from long distance between two instances belonging to the same flow. In [20], Ghaznavi et al. compare different

operations of VNFs or flows. But they assume one VNF-instance type.

In the opposite case, researches focus on determining the required number of instances, deploying them to available service nodes and routing flows. Various models have been built using MIQCP [21], MILP [22–24], and ILP [12, 25–27], which optimize different parameters such as end-to-end latency and resource utilization. We analyze them from the aspect of their solving methods. Mehraghdam et al. [21] use Gurobi optimizer, which is slow and cannot reconcile multiple objectives. To speed up the solving process, Mohammadkhan et al. [22] propose limiting the scale of problem through diving flows into groups. But they also use an off-the-shelf solver to solve the problem of each group. Allybokus et al. [27] present a heuristic algorithm based on a linear relaxation. In the case that two objectives are in competition, the method presented by Addis et al. [23] prioritizes them and uses CPLEX to find solutions for only one objective in a phase. However, it needs to limit the execution time of CPLEX in each phase. Improper time setting may affect quality of solutions. Similarly, based on introducing binary search, the method in [25] limits the execution time of CPLEX in each iteration. Bari et al. [12] use Viterbi algorithm to find a near-optimal placement of instances from multistage directed graph. But the graph needs to be updated frequently. Reference [26] compares the effects of different deployment strategies of VNFs on network resource consumption. But it does not illustrate how to solve the developed model. D'Oro et al. [28] propose a distributed solution by exploiting noncooperative game theory. But it assumes that source-destination flow is not split among multiple paths. On the basis of decomposing network functions into more elementary components, Sahhaf et al. [29] propose an algorithm based on backtracking mechanism. Reference [24] also adopts decomposition strategy but decomposes functions to multiple instances based on their performance demands. However, with respect to our work no consideration is made on instance sharing explicitly. Beyond offline problems, Lukovszki and Schmid [30] propose deterministic online algorithms for deploying service chains.

From the above analysis, we can draw a conclusion that most researches do not clear up the relationship among flow, function, instance, and service node. Specifically, the function required by multiple flows can probably be mapped to an instance, or in other words those flows share an instance. Multiple instances providing the same functionality may be combined to serve a big flow, or in other words that flow is split among multiple paths. Meanwhile, multiple instances can run on a service node. Thus, in order to reduce the complexity of SSC-DP, it is necessary to determine the required instances for each SSC before placing them in the physical network. In addition, existing researches are insufficient in designing multipath routing of flows and improving solution quality of optimization model.

3. Architecture Description

We propose an SSC-based security service delivery approach. As shown in Figure 1, its architecture consists of the Security Service Management Platform (SSMP), the Security Function Orchestrating Engine (SFOE), and the Flow Steering Engine (FSE). SSMP is responsible for receiving and analyzing security requests from users or network attack detection tools. It extracts and organizes information about SSC from those requests, such as the required number and types of instances as well as their connections, which will be handed over to SFOE. Then SFOE places instances on suitable service nodes and gives the placement view to FSE. Meanwhile, it sends commands to those nodes, creating and starting the corresponding instances. Additionally, the instances should register with SSMP after being started and SSMP will issue security defense polices to them. Finally, according to SSC information and the placement view, FSE computes routing paths which are used to steer flows through instances in order. And those paths are realized by flow table rules issued by SDN controller. By this approach, instances of security functions can be placed anywhere in the network and dynamically composed to meet specific user or application demands. Once demands or network conditions change, instances can be automatically started or terminated and routing paths of flows can also be adjusted accordingly.

The proposed architecture can be integrated with ETSI NFV Architecture [31]. Figure 2 shows the integrated architecture. The Network Function Virtualization Orchestrator (NFVO) component is in charge of network services lifecycle management, such as instantiating, configuring, updating, and terminating. The Virtual Network Function Manager (VNFM) component is responsible for managing the lifecycle of VNF instances constituting specific network services. The Virtualized Infrastructure Manager (VIM) takes charge of managing NFV Infrastructure resources including computing, storage, and networking resources. The SDN controller component, which is logically placed with the VIM, is used for managing virtual networks through deploying flow table rules to the switches. Our proposed architecture can be regarded as the SSC Orchestrator component. Based on security requests, it constructs the placement view of VNF instances instantiated by the NFVO and creates routing paths of flows. There are two main interfaces exposed by the SSC Orchestrator. One is used by the SFOE to deliver the placement view to NFVO so as to instantiate the related VNFs. Another is used by the FSE to send routing paths of flows to the SDN controller so as to apply the flow table rules needed on the switches.

4. Definitions and Problem Statement

We first present a mathematical representation of a physical network, security request, and security function instance. Then we formally define the SSC-DP.

4.1. Definitions

Physical Network. We represent the physical network as an undirected graph $G_s = (N, E)$, where N and E denote the set of physical nodes and links, respectively. We classify those nodes into three groups as forwarding node n_{tr}, which only forwards packets to other nodes, such as switches; service node n_{sr}, which provides virtualized platform for running

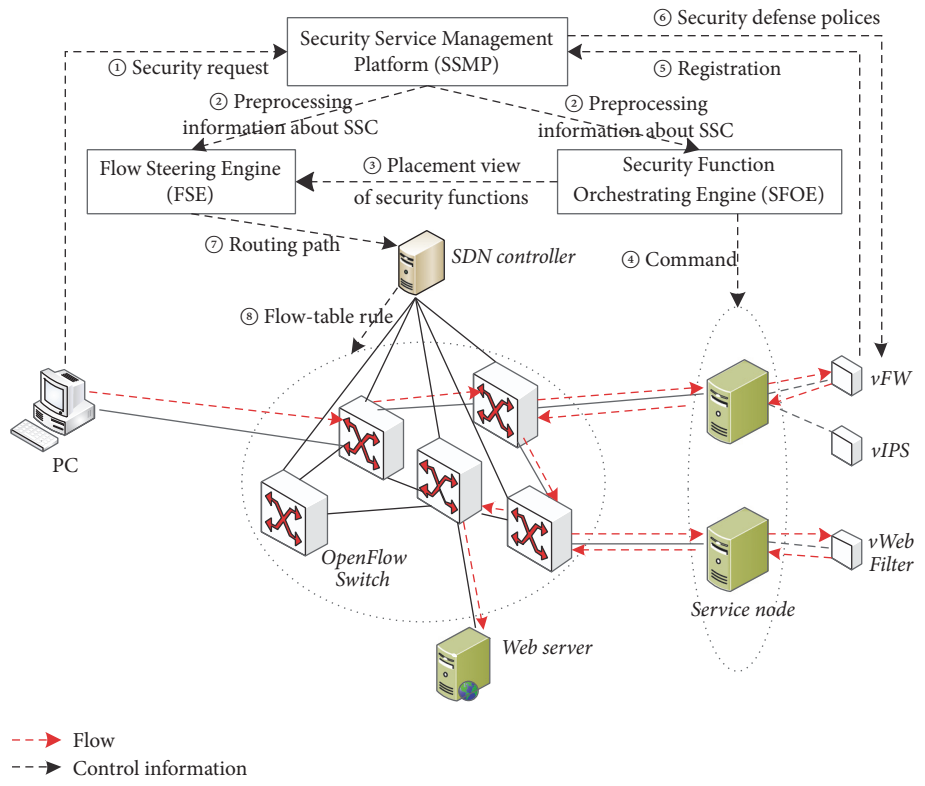

Figure 1: Architecture and component interactions.

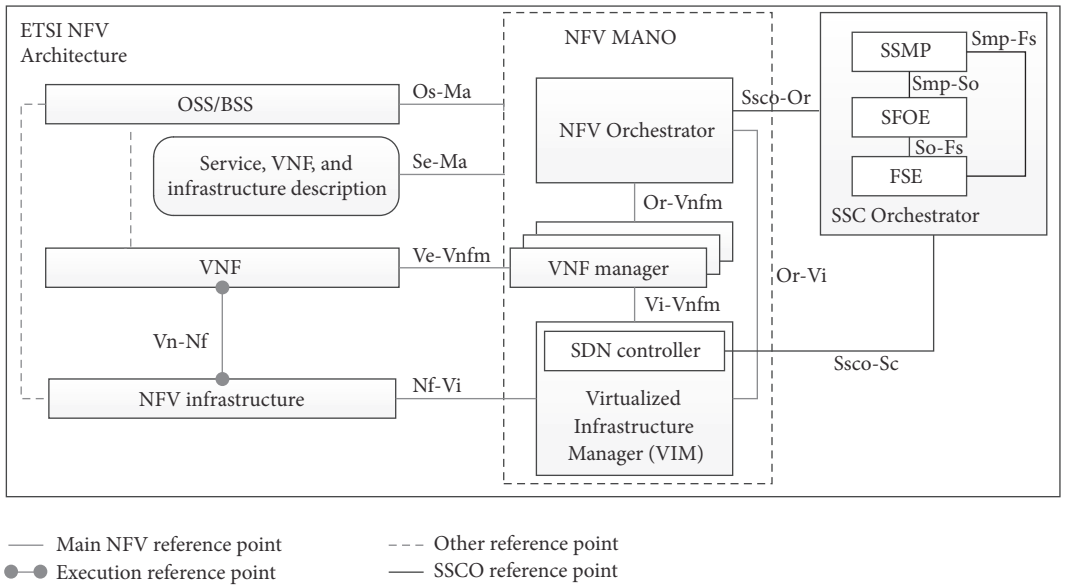

Figure 2: Integrated architecture.

instances; end node n_{end}, which is the source or destination of flow. Additionally, let R be the set of available resources on the service node. For each $r \in R$, $c(r)$ denotes the amount of r. Bandwidth and latency of physical link $e \in E$ are denoted as bw_e and lat_e, respectively.

Security Request. It is identified by 4-tuple, $rq := (\text{src}, \text{dst}, \text{ch}, \text{run})$, where src and dst are the source and destination of the flow generating rq, respectively. $ch = \{f_1, f_2, \ldots, f_n\}$ represents the SSC, where f_i denotes a security function, $1 \leq i \leq n$. If $1 \leq j = (i+1) \leq n$, f_i is the immediate predecessor of

f_j, denoted as $\text{pre}(f_j) = \{f_i\}$. Similarly, f_j is the immediate successor of f_i, denoted as $\text{succ}(f_i) = \{f_j\}$. run represents service demand of rq. We define it as the throughput demand for instance and assume that the throughput demand is the same for the whole SSC in this paper. Other indications can also be used, like process rate of instance. It is an important factor for determining the number and types of instances and splitting flows.

Security Function Instance. A security function has different types of instances with heterogeneous resource demands, service capacities, and processing delays. Let $\text{ins}(f_i) = \{it_{i1}, it_{i2}, \ldots, it_{in}\}$ be the instance set of f_i, where an instance it_{ij} is identified by 5-tuple, $it_{ij} := (p\_type, type, ins\_n, cap, pd)$. p_type is the type of associated security function, like firewall, IDS. $type$ is used to distinguish instances with the same p_type, like firewalls developed by different companies. ins_n is the set of demands for different resources on service node. For each $r \in R$, $rd_{ij}^r \in ins\_n.cap$ represents service capacity, which is defined as the throughput of instance in this paper, denoted as c_{ij}. pd is processing delay.

4.2. Problem Statement. Given a physical network $G_s = (N, E)$ and a set of security requests RQ, instantiate each security function required by $rq \in RQ$ on certain service nodes in G_s and determine the physical routing paths of the flow generating rq. This procedure seeks to minimize resource consumption of service nodes, resource fragmentation, and security service latency. It is subjected to the following constraints:

(i) A security function $f_i \in rq$ can be instantiated on several service nodes in the form of different instance types of f_i. Each selected service node has sufficient resources to accommodate the demand of an instance of f_i. Additionally, different flows can share a security function instance on a service node.

(ii) A flow can be split. It means that there may be multiple routing paths of the flow between two service nodes where two adjacent security functions run. In addition, the total bandwidth demand on a physical link cannot exceed its available bandwidth.

5. TPSSC: A Three-Phase Method for Solving SSC-DP

Based on the architecture presented in Section 3, we propose TPSSC which finds near-optimal solutions of SSC-DP in three phases: designing, node mapping, and link mapping. The former phase, conducted by SSMP, designs a virtual security service topology according to security requests, which describes the required instances and their relations.

However, although the number of instances and the throughput demand for each instance can be obtained from the designing phase, optimally mapping virtual security service topology to physical network is still NP-Hard [24]. We consider two optimization objectives in this paper: reducing resource fragmentation and security service latency. A naive way to perform the mapping is to treat each optimization as an independent subproblem and solve them sequentially, namely, solving first the placement of instances and then the routing of flows. However, this way usually makes it difficult to reduce latency because adjacent instances of the same flow may be placed far away from each other. On the other hand, some studies in the field of VNE have proposed an isomorphic graph search based algorithm to solve those two problems together [32]. But the algorithm is complex, and unlike VNE, end nodes of SSC are fixed physically and the sequence of security functions is often unidirectional. By combining the advantages of those two methods, we propose that a mapping procedure considers both resource fragmentation and security service latency throughout the mapping process. It can be divided into two phases, namely, node mapping and link mapping. The two phases jointly map the virtual security service topology to physical network, namely, determining placement of instances and routing paths of flows, which are conducted by SFOE and FSE, respectively. In each phase, we optimize SSC deployment from different perspectives. For the sake of convenient query, we list the related symbols used in this paper in Abbreviations.

5.1. Designing Phase. In this phase, we propose an algorithm to map each SSC to a combination of instances. From the perspective of optimization, the combination of instances satisfying throughput demand of request should consume resources on service node as little as possible. And the flow cannot be too scattered, considering reducing the possibility of transmission interruption caused by link failure

$$I = \sum_{rq_m \in RQ} \sum_{f_i \in rq_m.\text{ch}} \sum_{it_{ij} \in \text{ins}(f_i)} \sum_{r \in R} \tau_r \cdot x_{ij}^m \cdot rd_{ij}^r, \tag{1}$$

$$D = \sum_{rq_m \in RQ} \sum_{f_i \in rq_m.\text{ch}} \sum_{it_{ij} \in \text{ins}(f_i)} x_{ij}^m. \tag{2}$$

Let x_{ij}^m be the number of instances it_{ij} assigned to the SSC of request rq_m. As shown in (1), I is the total resource consumption of service nodes, where τ_r is weighting factor used to adjust the relative importance of resources, $\sum_{r \in R} \tau_r = 1$. As shown in (2), D represents the total scatter degree of flows. Given certain SSC, the more the instances a flow needs to traverse, the more the microflows that flow should be split to. The basic procedure of our algorithm is shown in Algorithm 1. It accepts security request set RQ, security function set F, and the maximum number of iterations R as input. Note that each f_i in F has an instance set, in which elements are sorted by their throughput (i.e., c_{ij}) in descending order. Here, we use a temporary variable t_{ij}^m to keep track of the candidate value of x_{ij}^m and use p_i^m to indicate the instance under consideration. Since there may be too many available combinations of instances, the proposed algorithm, based on the idea of greedy, gives priority to instances with high throughput when assigning them to each SSC and limits the maximum number of iterations.

Input: Security request set RQ, security function set F, maximum number of iterations R
Output: Instance combinations of SSCs
Initialize $r = 0$; $t_{ij}^m = 0$; $p_i^m = 1$
while $(r < R)$
 for all $rq_m \in RQ$ and $f_i \in rq_m$.ch **do**
  $d = rq_m.\text{run} - \sum_{0 \le q < p_i^m} \left(t_{iq}^m \cdot c_{iq}\right)$
  Find the minimum $t_{ip_i^m}^m$ satisfying $t_{ip_i^m}^m \cdot c_{ip_i^m} \ge d$
 end for
 Compute $(\alpha I + \beta D)$
 if $r = 1$ or $(\alpha I + \beta D) < \delta$ **then**
  $\delta = (\alpha I + \beta D)$
  Assign all t_{ij}^m to x_{ij}^m
 end if
 update(all t_{ij}^m)
 $r = r + 1$
end while
return $\{x_{ij}^m \mid rq_m \in RQ, f_i \in rq_m.\text{ch}, it_{ij} \in \text{ins}(f_i)\}$
Function *update*(all t_{ij}^m)
for all $rq_m \in RQ$ and $f_i \in rq_m$.ch **do**
 if $p_i^m \ge |ins(f_i)|$ **then**
  Backtrack set T_i^m from $p_i^m - 1$th element until
  find the first element satisfying $t_j > 0$
  $p_i^m = j$
 end if
 $t_{ip_i^m}^m = t_{ip_i^m}^m - 1$
 $p_i^m = p_i^m + 1$
end for

Algorithm 1: Mapping SSC.

Let $\{x_{ij}^m \mid it_{ij} \in \text{ins}(f_i)\}$ represent the instance set of f_i which is assigned to rq_m. For each it_{ij}, th_{ij}^m denotes the throughput demand of rq_m for it. It satisfies $th_{ij}^m \le c_{ij}$; $\forall f_i \in rq_m.\text{ch}$, $\sum_{it_{ij} \in \text{ins}(f_i)} th_{ij}^m = rq_m.\text{run}$; the larger c_{ij} is, the smaller $(c_{ij} - th_{ij}^m)$ is.

Since Algorithm 1 does not take into account sharing instances among flows, we can merge the instance combinations of different SSCs, which can further reduce resource consumption. Then virtual security service topology, represented as a directed graph $G_v = (V, L)$, is built. The meanings of symbols are as follows.

$v \in V$ is a virtual node. If it represents an instance it_{ij}, its weight $w(v)$ is defined as the set of throughput demands of requests for this instance. Specifically, if $\sum_{x_{ij}^m > 0} th_{ij}^m \le c_{ij}$, $w(v) = \{th_{ij}^m \mid x_{ij}^m > 0\}$. Otherwise, there are a set of nodes $\{v'\}$ representing it_{ij}, and for each v', $\sum_{th_{ij}^m \in w(v')} th_{ij}^m \le c_{ij}$. If v represents the source of flow, denoted as s, $w(v)$ is defined as the set of throughput demands of requests whose flow starts from s; that is, $w(v) = \{rq_m.\text{run} \mid rq_m.\text{src} = s\}$. The definition of $w(v)$ is similar if v represents the destination.

$l \in L$ is a virtual link, representing the order between two instances or between the source/destination of flow and an instance. Assume that security functions f_i and f_j belong to SSCs of rq_m and rq_n, respectively, and they satisfy $\text{pre}(f_j) = f_i$. If the instance it_{ip} (it_{iq}) of f_i (f_j) is represented by the virtual node v_p (v_q), there is a virtual link from v_p to v_q, denoted as $l(v_p, v_q)$. Its weight is defined as the set of throughput demands of rq_m and rq_n; that is, $w(l(v_p, v_q)) = \{db_{(v_p,v_q)}^m, db_{(v_p,v_q)}^n\}$, where $db_{(v_p,v_q)}^m = \min(th_{ip}^m, th_{iq}^m)$ and $db_{(v_p,v_q)}^n = \min(th_{ip}^n, th_{iq}^n)$. If an end node of l represents source or destination, relevant definitions are similar. Additionally, we use $id(l) = \{rq_m, rq_n, \ldots\}$ to record the requests whose SSCs use the edge l.

5.2. Node Mapping Phase. In this phase, we aim to select a suitable service node to run the instance represented by each virtual node.

5.2.1. Formulation

$$\min \quad \max_{n_i \in N_{\text{sr}}} \left(\text{fra}_i\right), \tag{3}$$

$$\text{utl}_i^r = \frac{\sum_{v_f \in V_{\text{ins}}} x_{if} \cdot \text{res}_f^r}{c_i(r)}, \tag{4}$$

$$\overline{\mathrm{utl}_i} = \frac{\sum_{r\in R} \mathrm{utl}_i^r}{|R|}, \tag{5}$$

$$\mathrm{fra}_i = \sqrt{\sum_{r\in R}\left(\frac{\mathrm{utl}_i^r}{\overline{\mathrm{utl}_i}} - 1\right)^2}, \tag{6}$$

$$\min \quad \max_{(s,d)\in\Phi}\left[\max_{\pi_i\in\pi(s,d)}\left(\sum_{l(u,v)\in\pi_i}\sum_{m,n\in N} x_{um}\cdot x_{vn}\cdot \mathrm{hop}_{(m,n)}\right)\right], \tag{7}$$

$$\text{s.t.} \quad \forall v_f \in V_{\mathrm{ins}} : \sum_{n_i\in N_{\mathrm{sr}}} x_{if} = 1, \tag{8}$$

$$\forall n_i \in N_{\mathrm{sr}},\ \forall r \in R : \sum_{v_f\in V_{\mathrm{ins}}} x_{if}\cdot \mathrm{res}_f^r \le c_i(r) \tag{9}$$

$$\forall v_s \in V_{\mathrm{end}},\ \forall n_i \in N_{\mathrm{end}} - \{a\} : x_{as} = 1,\ x_{is} = 0\ , \tag{10}$$

$$\forall v_d \in V_{\mathrm{end}},\ \forall n_i \in N_{\mathrm{end}} - \left\{a'\right\} : x_{a'd} = 1,\ x_{id} = 0\ , \tag{11}$$

$$\forall n_i \in N_{\mathrm{sr}},\ \forall v_f \in V_{\mathrm{ins}} : x_{if} \in \{0,1\}\,. \tag{12}$$

As resource fragmentation limits network to accept security requests, we take minimizing the maximum resource fragmentation of service nodes ((3)) as an objective. fra_i ((6)) measures the resource fragmentation of n_i by computing the deviation between utilizations of different resources ((4) and (5)). The smaller the deviation is, the more balanced the utilizations are. Additionally, we consider security service latency by optimizing the length of routing path. For the sake of simplicity, we define the path length between two instances (we regard the source and destination of flow as instances with fixed physical locations in node mapping phase and link mapping phase) as the minimum number of hops of all paths between the service nodes they are placed on. Then for a virtual path, its length is the sum of path lengths between instances along it. As a flow may correspond to several virtual paths constructed by different instances in the virtual security service topology, we define the length of routing path as the maximum length of all virtual paths. So minimizing the maximum length of routing path ((7)) is regarded as another objective.

Thus, we provide a constrained multiobjective optimization formulation, denoted as Problem P. It seeks to obtain the optimal selection of service nodes without violating the constraints of capacities of physical nodes and links. Our formulation is as follows.

Equation (8) guarantees that an instance must be placed on exactly one service node; (9) constrains the fact that resource demands of all instances placed on a service node should be less than or equal to available resources in that node; (10) and (11) ensure that physical locations of the source and destination of a flow are respected, respectively; (12) constrains decision variables to be 0 or 1. For the sake of clarify, we denote (3) and (7) as $f_1(X)$ and $f_2(X)$, where $X = (x_{ij})_{|N_{\mathrm{sr}}+N_{\mathrm{end}}|\times|V_{\mathrm{ins}}+V_{\mathrm{end}}|}$.

5.2.2. Proposed Algorithm. To obtain the Pareto-optimal solutions of the above problem, inspired by immune memory clonal algorithms [33, 34], we propose a service node selection algorithm based on bidirectional memory. The key idea is to approximate the Pareto-optimal solutions from feasible and infeasible regions. The basic procedure of our algorithm is shown in Algorithm 2. It first establishes the memory unit and the standby unit to reserve the current Pareto-optimal feasible and infeasible solutions, respectively. After implementing clone, mutation, and selection operation, it extracts preponderant antibody population and neighboring antibody population from the whole population. Then the former integrates with the previous Pareto-optimal solutions in memory unit, which ensures that the quality of solutions is not degraded. The latter cooperates with the standby unit to approximate the Pareto-optimal solutions from infeasible region, which maintains diversity of antibody population. Additionally, the newly obtained Pareto-optimal solutions are used as the initial antibody population in next iteration, which accelerates convergence rate of the proposed algorithm. The main data structures and detailed operations are presented as follows.

(1) Main Data Structure Description. There are three main data structures used in the proposed algorithm.

Antibody Population. The algorithm maintains an antibody population $A(it) = \{a_1(it), a_2(it), \ldots, a_{N_a}(it)\}$ at the itth generation, where N_a is the size of the population. Antibody $a_i(it)$ is the encoding of candidate solution X for the Problem P; that is, $a_i(it) = e(X) = (a_i^1(it), a_i^2(it), \ldots, a_i^n(it))$, $1 \le i \le N_a$, where n is the length of the antibody and $a_i^f(it) = k$ means placing the instance represented by v_f on the service node n_k; namely, $x_{kf} = 1$. In particular, if v_f represents the source or

Input: Maximum number of iterations T, maximum size of antibody population N_a, maximum size of memory unit N_m, size of standby unit N_b, initial mutation probability mp_0
Output: Memory unit $M(it)$
Initialize $it = 0$; initializing antibody population $A(it)$, memory unit $M(it)$ and standby unit $B(it)$.
Step 1. Generate $C(it)$ from $A(it)$ by the clone operation O_c:
$C(it) = O_c(A(it)) == \{c_1(it), c_2(it), \ldots, c_{N_c}(it)\}$, where $N_c = \sum_{i=1}^{N} p_i(it)$.
Step 2. Update $C(it)$ by the mutation operation O_m:
$D(it) = O_m(C(it)) = \{d_1(it), d_2(it), \ldots, d_{N_c}(it)\}$.
Step 3. Generate preponderant antibody population $P(it)$ and neighboring antibody population $Q(it)$ from $D(it)$ by selection operation O_s.
Step 4. If the size of $P(it)$ is larger than N_a, sort antibodies by their crowding distances [35] in descending order and select the top N_a antibodies to form new antibody population $P'(it)$, otherwise $P'(it) = P(it)$:
$P'(it) = O_u(P(it)) = \{p'_1(it), p'_2(it), \ldots, p'_{N_a}(it)\}$.
Step 5. Produce new memory unit $M'(it)$ by applying study operation O_l on $M(it)$ and $P'(it)$:
$M'(it) = O_l(M(it), P'(it)) = \{m'_1(it), m'_2(it), \ldots, m'_{R(it+1)}(it)\}$, where $R(it+1) \le N_m$.
Step 6. Update $Q(it)$ by the self-repairing operation O_r:
$Q'(it) = O_r(Q(it)) = \{q'_1(it), q'_2(it), \ldots, q'_{N_q}(it)\}$.
Step 7. Produce new standby unit $B'(it)$ by applying replacement operation O_a on $B(it)$ and $Q'(it)$:
$B'(it) = O_a(B(it), Q'(it)) = \{b'_1(it), b'_2(it), \ldots, b'_{N_b}(it)\}$.
Step 8. If $it \ge T$, output $M'(it)$ and end, otherwise $A(it+1) = P'(it)$, $M(it+1) = M'(it)$, $B(it+1) = B'(it)$, $it = it + 1$, go to Step 1.

Algorithm 2: Service node selection algorithm based on bidirectional memory.

destination of flow, $a_i^f(it)$ is a known quantity and will not be changed by the following operations.

By this encoding method, the two-dimensional mapping relation between instances and service nodes is transformed to one-dimensional vector, which satisfies (8) and (10)–(12) inherently. So (9) is used to judge the feasibility of $a_i(it)$.

We introduce a new function $g_i^r(X(i,:)) = \max\{0, \sum_{v_f \in V_{\mathrm{ins}}} x_{fi} \cdot \mathrm{res}_f^r - c_i(r)\}$, where $n_i \in N_{\mathrm{sr}}$ and $r \in R$. Assume that $f_3(X) = \sum_{n_i \in N_{\mathrm{sr}}} \sum_{r \in R} g_i^r(X(i,:))$. If $f_3(e^{-1}(a_i(it))) = 0$, namely, $X = e^{-1}(a_i(it))$ satisfies all constraints of the Problem P, $a_i(it)$ is called feasible antibody. Otherwise, $a_i(it)$ is called infeasible antibody. Furthermore, $f_3(e^{-1}(a_i(it)))$ is used to measure the degree of constraint violation of an infeasible antibody. The larger it is, the deeper the degree of constraint violation of $a_i(it)$ is.

If two feasible antibodies, $a_i(it)$ and $a_j(it)$, satisfy the condition that $(\forall k = 1, 2 : f_k(e^{-1}(a_i(it))) \le f_k(e^{-1}(a_j(it)))) \wedge (\exists g = 1, 2 : f_g(e^{-1}(a_i(it))) < f_g(e^{-1}(a_j(it))))$, $a_i(it)$ is said to Pareto dominate $a_j(it)$, denoted as $a_i(it) \succ a_j(it)$. Furthermore, $a_i(it)$ is called Pareto-optimal if there does not exist another feasible antibody $a^*(it)$ in $A(it)$ that satisfies $a^*(it) \succ a_i(it)$.

Memory Unit. Memory unit $M(it) = \{m_1(it), m_2(it), \ldots, m_{R(it)}(it)\}$ is defined as the set of all Pareto-optimal antibodies in $A(it)$, whose size $R(it)$ changes dynamically. In other words, it contains service node selection schemes that are Pareto-optimal. We can implement any one scheme in it. To improve the quality of solutions, we assume that the upper limit of $R(it)$ is N_m.

Standby Unit. Standby unit $B(it) = \{b_1(it), b_2(it), \ldots, b_{N_b}(it)\}$, whose size is N_b, is defined as the set of infeasible antibodies with relatively low degree of constraint violation in $A(it)$.

(2) Operation Description. We describe operations in the proposed algorithm successively.

Initializing Operation. To improve the quality of initial solutions, we propose an antibody population initialization algorithm based on preference. The preference of the instance v_f for the service node n_i is defined as $pf_f^i = 1/((1/|R|)\sum_{r \in R} \lambda_r^2 - ((1/|R|)\sum_{r \in R} \lambda_r)^2 + \sigma)$, where $\lambda_r = \mathrm{res}_f^r / c_i(r)$ is the occupancy ratio of resource r in n_i occupied by v_f and σ is a small positive constant to avoid dividing by zero in computing the preference. Then the preference list of v_f, denoted as $PL(v_f)$, is built by sorting service nodes by pf_f^i in descending order. Note that preference lists are known before running the algorithm.

The key idea of our algorithm is to traverse preference list of each instance until finding the service node satisfying resource constraint and distance constraint so as to achieve optimization objectives initially. The basic procedure is shown in Algorithm 3. Here, function $Selectfirst(PL(v_f))$ means traversing $PL(v_f)$ in sequence until finding n_i which is the first service node satisfying two conditions simultaneously: (i) $\forall r \in R$, $\mathrm{res}_f^r \le c_i(r)$; (ii) $\forall v_j \in \mathrm{pre}(v_f)$, $a^j = g$, $\mathrm{hop}(n_g, n_i) \le \theta$. If all nodes in $PL(v_f)$ just satisfy only one condition, it selects the first node satisfying condition (i). If all nodes do not satisfy condition (i), it randomly selects a service node occupied by an instance of $\mathrm{pre}(v_f)$.

Input: G_v, preference lists of all instances, the upper limit of hops between two nodes θ
Output: Initial antibody population
for $m = 1$ to N_a **do**
  Trs = $\varnothing$
  while $(Trs \neq V_{\text{ins}})$
    for all $v_f \in V_{\text{ins}}$ **do**
      if $\text{idg}_{v_f} = 0$ **then** //idg_{v_f} is the in-degree of v_f
        $n_i = Selectfirst\,(PL(v_f))$
        $a_m^f = i$
        for all $r \in R$ **do**
          $c_i(r) = c_i(r) - \text{res}_f^r$
        end for
        $\text{idg}_{v_f} = \text{idg}_{v_f} - 1$
        $\text{Trs} = \text{Trs} \bigcup \{v_f\}$
        for all $v_k \in \text{succ}(v_f)$ **do**
          $\text{idg}_{v_k} = \text{idg}_{v_k} - 1$
        end for
      end if
    end for
  end while
  $m = m + 1$
end for
return $A(0) = \{a_1(0), a_2(0), \ldots, a_{N_a}(0)\}$

Algorithm 3: Antibody population initialization algorithm based on preference.

Clone Operation. O_c is defined as $O_c(A(it)) = \{O_c(a_1(it)), O_c(a_2(it)), \ldots, O_c(a_{N_a}(it))\}$, where $O_c(a_i(it)) = \{a_{11}(it), \ldots, a_{1p_i(it)}(it)\}$, $1 \le i \le N_a$, and $p_i(it)$ is the clone scale of $a_i(it)$

$$p_i\,(it) = \text{Int}\left(H \cdot \frac{\eta_i\,(it)}{\sum_{j=1}^{N(it)} \eta_j\,(it)} \cdot \frac{1}{\psi_i\,(it)}\right). \tag{13}$$

$p_i(it)$ is given by (13) and can self-adaptively be adjusted by the antibody-antibody affinity $\psi_i(it)$ and the antibody-antigen affinity $\eta_i(it)$. Their detailed definitions are as follows:

(i) Antibody-antibody affinity $\psi_i(it)$: it is measured by the Euclidean distance between $a_i(it)$ and other antibodies: $\psi_i(it) = \min\{\exp(-\|a_i(it) - a_j(it)\|)\}$, where $i \neq j$, $1 \le j \le N_a$; $\|\cdot\|$ represents Euclidean distance, and it is normalized to $[0, 1]$.

(ii) Antibody-antigen affinity $\eta_i(it)$.

$$R_i^j\left(a_k\,(it)\right) = \begin{cases} 1 & f_j\left(e^{-1}\left(a_i\,(it)\right)\right) \le f_j\left(e^{-1}\left(a_k\,(it)\right)\right), \\ 0 & \text{else}, \end{cases} \tag{14}$$

$$\eta_i^j\,(it) = \sum_{k=1}^{N_a} R_i^j\left(a_k\,(it)\right), \tag{15}$$

$$\eta_i\,(it) = \sum_{j=1}^{3} \eta_i^j\,(it). \tag{16}$$

To unify the affinity computation of feasible and infeasible antibodies, we regard $\min f_3(X)$ as an objective of the Problem P. So given a single antigen (i.e., an objective function) $f_j(X)$, we can obtain relative affinity between the antibody $a_i(it)$ and the antigen $f_j(X)$, denoted as $\eta_i^j(it)$ ((15)), through comparing the objective value of $a_i(it)$ (i.e., $f_j(e^{-1}(a_i(it)))$) with objective values of other antibodies ((14)). For the Problem P with multiple objectives, $\eta_i(it)$ is defined as the sum of relative affinities between $a_i(it)$ and each objective ((16)). This method can eliminate the bad influence of a too large or too small objective value on the affinity.

H is a given value relating to clone scale (we assume that $H = 3N$) and the function Int() returns the value of a number rounded upwards to the nearest integer. Apparently, the clone scale decreases with the increase of inhibitory effect between antibodies (namely, $\psi_i(it)$ increases) and the decrease of antigen stimulation (namely, $\eta_i(it)$ decreases).

Mutation Operation. Since every value in antibody indicates a service node, we propose a new mutation strategy to make the mutation operation meaningful. For each antibody in $C(it)$, we select two values in it randomly and exchange them with probability of $\text{mp} = \text{mp}_0 \cdot (1 - it/T)$, where mp_0 is the initial mutation probability, and it and T are the current and the maximum number of iterations, respectively. Apparently, mp decreases with the proposed algorithm running.

Selection Operation. First of all, we separate feasible antibodies from infeasible ones. Then, we extract Pareto-optimal antibodies from the former to form the preponderant antibody population $P(it)$. Meanwhile, we choose N_q antibodies

with the lowest degree of constraint violation from infeasible antibodies to form the neighboring antibody population $Q(it)$. Adding antibodies of $Q(it)$, which approximate the edge of feasible region, to the next iteration can improve diversity of antibody population.

Self-Repairing Operation. Through migrating instances from overloaded service nodes to those with abundant resources, self-repairing operation can reduce the degree of constraint violation of infeasible antibody and make it enter or be more close to feasible region. Let $Q(it) = \{q_1(it), q_2(it), \ldots, q_{N_q}(it)\}$ be the neighboring antibody population. Assume that antibodies in $Q(it)$ are sorted by the degree of constraint violation in descending order. For an antibody $q_i(it)$, the resource burden of service node n_j is defined as $\text{bur}_j = \sum_{r\in R}(\max\{0, \sum_{k\in\{k|q_{ik}=j\}} \text{res}_k^r - c_j(r)\}/c_j(r))$. If there exists a node without burden, that is, $\text{bur}_j = 0$, self-repairing operation will be applied to $q_i(it)$. Specifically, starting from n_j with the heaviest burden, it selects an instance v_k randomly and migrates it to the node $n_{j'}$ with $\text{bur}_{j'} = 0$, that is, replacing $q_{ik} = j$ with $q_{ik} = j'$. If the degree of constraint violation of the new antibody $q_i'(it)$ is smaller than that of $q_i(it)$, this migration is accepted. Otherwise, it selects another instance and repeats the previous migration until all instances on n_j are traversed. Then it tries to do migration in other nodes until there are no nodes without burden. Assuming that self-repairing operation is only applied to m antibodies, the time consumption can be controlled by adjusting m. In other words, m is the depth of self-repairing operation.

Replacement Operation. It replaces the antibodies in the standby unit with the antibodies which have lower degree of constraint violation in the neighboring antibody population. It ensures that the standby unit approximates feasible region gradually.

Study Operation. Through comparing antibodies in the updated preponderant antibody population with those in the memory unit, study operation updates the memory unit with the newest Pareto-optimal antibodies. If the size of memory unit exceeds N_m, updating operation based on crowding distance will be applied [35].

5.3. Link Mapping Phase. This phase is to route flows among the selected service nodes based on the virtual security service topology. We refer to this problem as the service path establishment problem. Since a flow can be split and the capacity of physical link is limited, we treat virtual links as commodities and model the service path establishment problem as the capacitated multicommodity flow problem.

5.3.1. Formulation

$$\min \quad \max_{(s,d)\in\Phi} \left(clt_{sd}\right), \tag{17}$$

$$clt_{sd} = \max_{\pi_i\in\pi(s,d)} \left[\sum_{l(u,v)\in\pi_i} \sum_{e(m,n)\in E} stf\left(y_{(m,n)}^{(u,v)}\right) \cdot \text{lat}_{(m,n)} + \sum_{v_f\in\varphi(\pi_i)} pd_f \right], \tag{18}$$

$$\text{s.t.} \quad \forall e(m,n) \in E: \sum_{l(u,v)\in L} y_{(m,n)}^{(u,v)} \le bw_{(m,n)}, \tag{19}$$

$$\begin{aligned} &\forall l(u,v) \in L,\ x_{ub} = x_{vb'} = 1,\ \forall n \in N - \left\{b, b'\right\}: \\ &\sum_{c\in N} y_{(c,n)}^{(u,v)} - \sum_{c'\in N} y_{(n,c')}^{(u,v)} = 0, \\ &\sum_{c\in N} y_{(c,b)}^{(u,v)} = \sum_{c'\in N} y_{(b',c')}^{(u,v)} = 0, \\ &\sum_{c\in N} y_{(b,c)}^{(u,v)} = \sum_{c'\in N} y_{(c',b')}^{(u,v)} = db_{(u,v)}, \end{aligned} \tag{20}$$

$$\forall e(m,n) \in E,\ l(u,v) \in L: y_{(m,n)}^{(u,v)} \le z_{(m',n')}^{(m,n)} \cdot x_{um'} \cdot x_{vn'} \cdot \min\left(bw_{(m,n)}, db_{(u,v)}\right), \tag{21}$$

$$\forall e(m,n) \in E,\ l(u,v) \in L: y_{(m,n)}^{(u,v)} \ge 0. \tag{22}$$

A security request may correspond to multiple virtual paths constructed by different instances in the virtual security service topology. So the maximum latency of those paths is regarded as the service latency of security request ((18)), where stf() is step function and it is assumed that $\text{stf}(0) = 0$; (19) is a constraint on the maximum bandwidth of physical links that can be assigned to different virtual links; (20) is the flow conservation constraint. It ensures that, for every node n in the physical network, if one of its incoming links belongs to the path which a virtual link is mapped to, one of its outgoing

links also belongs to that path. Excluded from this rule is the case where the node is one of the nodes to which the two end nodes of virtual link are mapped; (21) constrains the fact that a virtual link must be mapped to the path between the two nodes whose end nodes are mapped to. Note that $z_{(m',n')}^{(m,n)}$, $x_{um'}$, and $x_{vn'}$ are not decision variables but introduced to express that constraint; (22) is a constraint on the values of decision variables.

5.3.2. Proposed Algorithm. Since the capacitated multicommodity flow problem is NP-Hard [36], we propose a heuristic named service path establishment algorithm based on hybrid taboo search. Before introducing the algorithm, we describe the main data structures followed by discussing the key aspects of the algorithm: neighborhood search method, evaluation function, and termination condition.

(1) Main Data Structure Description. There are three main data structures used in the proposed algorithm.

Initial Solution Set. The initial solution set is defined as $S_0 = \{s_1, s_2, \ldots, s_n\}$, where $s_i = (y_{(m,n)}^{(u,v)})_{E\times L}$, $1 \le i \le n$. Take the process of generating an initial solution as an example. Assume that every virtual link has a set of k-shortest physical paths between the two nodes whose start and end node are mapped to. The algorithm randomly selects a node of which the in-degree equals 0 in the virtual security service topology and maps virtual links connected to that node to physical paths. For each virtual link, single path mapping is applied first. Specifically, physical path in its k-shortest path set is traversed in increasing order of their lengths until a path satisfying bandwidth demand is found. If no such single path exists, multipath mapping is applied, which prefers to assign as much bandwidth demand as possible to relatively short physical path. After all virtual links connected to the node have been mapped, the node and virtual links are marked as "traversed" and removed from the virtual topology. Then another node of which the in-degree equals 0 will be chosen to repeat the above operation. An initial solution is generated after all nodes and links in the virtual topology have been traversed.

Dominant Solution Set. For $S_0 = \{s_1, s_2, \ldots, s_n\}$, solutions are sorted by their objective function values (i.e., $\max_{(s,d)\in\Phi}(clt_{sd})$, denoted as $h(s_i)$) in ascending order. So the first m solutions are chosen to form the dominant solution set. It is denoted as DS $= \{s_1, s_2, \ldots, s_m\}$, where $h(s_1) \le \cdots \le h(s_m)$. During the running time, DS will be updated continually to ensure that it always keeps the optimal solutions. Moreover, the depth of local search is adjusted dynamically according to whether DS is updated or not.

Tabu List. Assuming that virtual links and physical links are identified by positive integers, the tabu list is defined as TL $= (\{(l, \{y_{ab} \mid 1 \le a \le |E|,\ b = l\}) \mid 1 \le l \le |L|\}, ctl)$, where l is the identification of virtual link and y_{ab} is the bandwidth assigned to the virtual link b by the physical link a. The length of TL is set to be 7 [36].

(2) Key Aspect Description. In what follows, we discuss three key aspects with respect to the proposed algorithm.

Neighborhood Search Method. Given the solution s_i, its neighborhood $N(s_i) = NeborCons(s, ns) = \{s_i^1, s_i^2, \ldots, s_i^{ns}\}$ can be built as follows. First, a virtual link is selected randomly. Then through adjusting the bandwidth demand assigned to each physical path in its k-shortest path set, a new service path establishment scheme, or in other words a neighboring solution s_i^1 is generated. It is particularly important that the new scheme should satisfy the overall bandwidth demand of the selected virtual link. Similarly, other $(ns - 1)$ neighboring solutions can be generated. The size of $N(s_i)$ is defined as $ns = ns_{\min} + nt \cdot (ns_{\max} - ns_{\min})/NT$, where $ns_{\max} = L$ and $ns_{\min} = 0.5L$ are the maximum and the minimum size of neighborhood, respectively, and nt and NT are the current and the maximum number of iterations of neighborhood search, respectively. Note that though the generated neighboring solutions satisfy (20) and (21), some of them may violate (19), which should be eliminated.

Evaluation Function. It is denoted as *Evaluate()*. We take the objective function $h(s)$ as the evaluation function. To eliminate infeasible solutions, a highest evaluation value will be assigned to them.

Termination Condition. The algorithm will terminate as long as one of the following conditions are satisfied: the number of iterations exceeds T; solutions are not improved; namely, the objective function value remains constant after R successive iterations.

The basic procedure of our algorithm is shown in Algorithm 4. Here, we use the functions $Update(DS, S_0)$ and $Update(DS, N(s))$ to replace some solutions in DS with better ones in S_0 and $N(s)$, respectively. We also use roulette wheel strategy to select the candidate solution set from $N(s)$, denoted as $Roulette\_wheel(N(s))$.

The algorithm generates multiple initial solutions based on greedy strategy and randomly selects one solution as the start point of iteration, which can solve the problem that tabu search relies heavily on initial solution. Additionally, it takes into account both diversification and intensification strategy. On the one hand, the longer the dominant solution set is not updated during neighborhood search, the larger the search space is. It reflects that the algorithm increases diversity of solutions by expanding search space, which benefits finding better solutions. On the other hand, the dominant solution set keeps the first m excellent solutions so far and every iterative search starts from them. It reflects that the algorithm is exploited in promising space through concentrating on searching the neighborhood of the current excellent solutions.

5.4. Time Complexity Analysis. We mainly analyze the time complexity of two algorithms in mapping phase.

For the service node selection algorithm based on bidirectional memory, we take the first iteration as an example to analyze its time complexity step by step. N_a, N_m, and N_b

```
Input: NT, T, R
Output: The best solution s*
Initialize: DS = ∅; TL = ∅; t = 0; impro = 0
Generate initial solution set S0
Update (DS, S0)
while (t < T and noimpro < R)
    Randomly select s from DS
    r = h(DS[1])
    nt = 0
    s_best = s
    while (nt < NT)
        N(s) = NeborCons(s, ns)
        Evaluate (N(s))
        isupdated = Update (DS, N(s))
        if isupdated then nt = 0
        else nt + +
        end if
        CS = Roulette_wheel (N(s))
        if isaspiration then
            s = SelectBest (CS)
            s_best = s
            Update the tabu list TL
        else
            s = SelectBest_notabu (CS)
            Update the tabu list TL
        end if
    end while
    t + +
    if h(DS[1]) < r then noimpro = 0
    else noimpro + +
    end if
end while
return DS[1]
```

Algorithm 4: Service path establishment algorithm based on hybrid taboo search.

represent the size of the initial antibody population, memory unit, and standby unit, respectively. Firstly, assuming that the number of service nodes and instances is N_{sr} and N_{ins}, respectively, the worst time complexity of initializing operation is $O(N_a \cdot N_{ins} \cdot N_{sr})$. Then, the time complexity of building memory unit and standby unit is $O({N_a}^2)$ and $O(N_a + N_a \log N_b)$, respectively. Thirdly, if H is a given value relating to clone scale, the worst time complexity of clone operation is $O(N_a H)$. Fourthly, if the size of antibody population after cloning is N_c, the time complexity of mutation operation is $O(N_c)$. Fifthly, selection operation includes computing the degree of constraint violation of antibodies, selecting Pareto-optimal antibodies, and choosing N_q antibodies with the lowest degree of constraint violation. Their time complexities are $O(N_c)$, $O(N_c^2)$, and $O(N_c \log N_q)$, respectively. Then, if the size of preponderant antibody population after selection operation is K and there are N_a Pareto-optimal antibodies that should be reserved, the time complexity of updating preponderant antibody population is $O(N_a K^2)$. Next, the time complexity of self-repairing operation is $O(m \cdot N_{sr} \log N_{sr})$ if its depth is m. Then the worst time complexity of replacement operation is $O(N_b \cdot N_q)$. Finally, the time complexity of study operation is $O((N_a + N_m)^2)$. Therefore, the worst time complexity of the proposed algorithm during the first iteration is $O(N_a \cdot N_{ins} \cdot N_{sr} + N_c^2 + N_a \cdot K^2 + {N_a}^2 + N_m^2)$.

Table 1: Simulated networks used in evaluation.

Network	FT-6-A	FT-6-B	FT-8	Waxman
Forwarding node	45	45	80	24
End node	38	27	90	403
Service node	16	27	38	173

For the service path establishment algorithm based on hybrid taboo search, we take an iteration as an example to analyze its time complexity. Assume that there are L virtual links. The worst time complexity of generating the initial solution set with n solutions is $O(nkL)$. Subsequently, the time complexity of updating the dominant solution set with m solutions is $O(n \log m)$. The main operation in an iteration is building neighborhood with variable size. Considering the process from $nt = 0$ to $nt = NT$, the number of generated neighboring solutions is $NT \cdot ns_{min} + 0.5 \cdot (NT + 1) \cdot (ns_{max} - ns_{min})$. So the time complexity of this process can be regarded as $O(NT \cdot ns_{max})$. In conclusion, the time complexity of an iteration of the proposed algorithm is $O(nkL + NT \cdot ns_{max})$.

6. Evaluation

6.1. Experimental Setup. Our experiments are conducted on the fat-tree and Waxman topologies. The number of different types of nodes in simulated networks is shown in Table 1. The CPU and memory of service nodes are random numbers uniformly distributed between 200 and 500. The latency of each physical link is a number with uniform distribution between 1 and 10 time units. The bandwidth of each physical link is set to 5000. Note that the units for the bandwidth of physical link, the throughput demand of security request, and the throughput of instance are the same in the experiments. We omit them for the sake of simplicity. The above setting is the same for all experiments.

For the service node selection algorithm of TPSSC, we set $T = 200$, $N_a = 300$, $N_m = N_b = 30$, and $mp_0 = 0.7$. For the service path establishment algorithm, we set $NT = 20$, $T = 200$, and $R = 50$. We design a random algorithm and two greedy algorithms with single objective for comparison, which are denoted as RD, GD-1, and GD-2, respectively. RD randomly selects service nodes and establishes service paths. The goal of GD-1 is to reduce resource fragmentation. For a security request, GD-1 traverses its SSC sequentially and selects the "best" service node for running instance of each security function. The "best" service node has the minimum resource fragmentation after the instance is placed on it. With the objective to reduce security service latency, GD-2 defines the "best" service node as the one that the latency from the previous placed instance to it is the minimum. Note that instances can be placed on the same service node.

In the evaluation, we first compare TPSSC against the above three algorithms with respect to length of SSC and throughput demand of security request. Experiments are conducted in the FT-6-B network. In each experiment, there

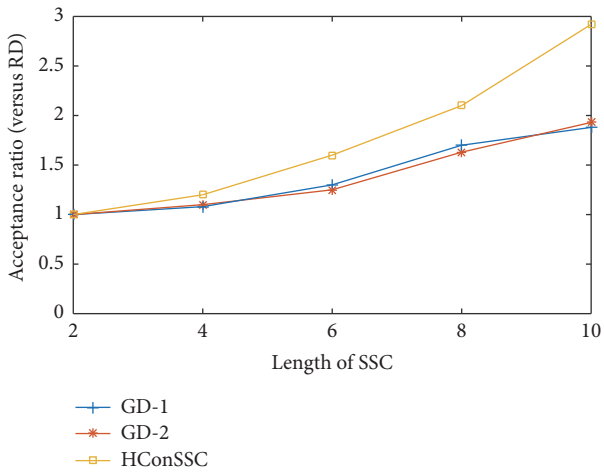

Figure 3: Comparison of acceptance ratio.

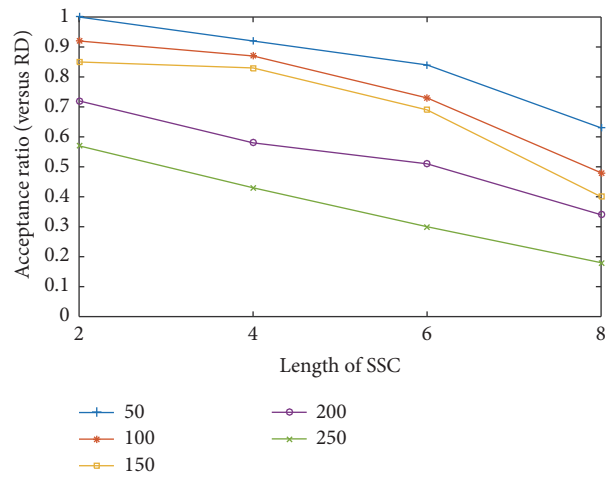

Figure 4: Acceptance ratio of TPSSC in the FT-6-A network.

are 300 service requests arriving in a Poisson process. Their sources and destinations are uniformly distributed in the simulated network. Their throughput demands are numbers uniformly distributed between 50 and 250. The length of SSC is fixed to one of {2, 4, 6, 8, 10}. As the above three algorithms do not consider the case where several instances of the same security function should be combined to serve a flow, we assume that every security function has only one instance. The CPU demand, memory demand, and throughput of an instance are 10, 10, and 300, respectively. We measure the acceptance ratio, the maximum resource fragmentation, the maximum security service latency, and the time overhead of algorithm. Each experiment is iterated 50 times and the arithmetic mean is reported. The results are shown with their 95% confidence intervals. For the convenience of comparative analysis, we take RD's value as benchmark and the ratio of the value of other three algorithms to RD's corresponding value is reported. We also conduct some experiments to analyze the performance of TPSSC specifically.

6.2. Results

6.2.1. Acceptance Ratio. Figure 3 shows the acceptance ratio of GD-1, GD-2, and TPSSC versus RD. When the length of SSC is less than 4, the acceptance ratios are similar to each other. However, TPSSC, GD-1, and GD-2 have better results for longer SSC. Particularly, TPSSC becomes more and more superior as the length increases. The first reason is that TPSSC takes into account reusing instance for several flows in the designing phase. As a result, fewer resources of service nodes are needed. The second reason is that TPSSC optimizes the allocation of network resources based on the full analysis of all security requests over time. So the situation, where "small" security requests are refused because of network resources having been occupied by "big" security requests in advance, can be avoided to a certain extent.

In order to further evaluate the ability of TPSSC to accept security requests, we measure the acceptance ratio of TPSSC by changing some parameters of security request and network environment.

(1) Acceptance Ratio under Different Throughput Demands. Experiments are conducted in the FT-6-A network. In each experiment, there are 400 security requests arriving in a Poisson process. The length of SSC is fixed and the throughput demand of each request is set to one of {50, 100, 150, 200, 250}. Every security function has 4 instances. The CPU and memory demands of an instance are numbers with uniform distribution between 5 and 30. The throughput of an instance is a number with uniform distribution between 10 and 300. Each experiment is iterated 50 times and the arithmetic mean is reported. The results are shown in Figure 4. Security requests with higher throughput demand and longer SSC have the less opportunity to be accepted because of limited network resources. Moreover, the acceptance ratio drops to less than 20% when the length of SSC is 8 and the throughput demand is 250 due to serious lack of network resources.

We also conduct the above experiment in the FT-6-B network. The results are shown in Figure 5. The acceptance ratio in the FT-6-B network is higher than that in the FT-6-A network. This is because the former has more service nodes for running more instances. Moreover, it allows TPSSC to choose from a wider range of service nodes and routing paths.

(2) Acceptance Ratio under Different Network Scales. Experiments are conducted in the FT-6-A, FT-8, and Waxman network, respectively. In each experiment, there are 1000 security requests arriving in a Poisson process. The length of SSC is fixed and the throughput demands are uniformly distributed between 50 and 250. Every security function has 4 instances. The CPU and memory demands of an instance are numbers with uniform distribution between 5 and 30. The throughput of an instance is a number with uniform distribution between 10 and 500. Each experiment is iterated 50 times and the arithmetic mean is reported. Figure 6 shows the results.

As shown in Figure 6, the acceptance ratio in the Waxman network is higher than that in the other two networks. The reason is that there are more network resources in

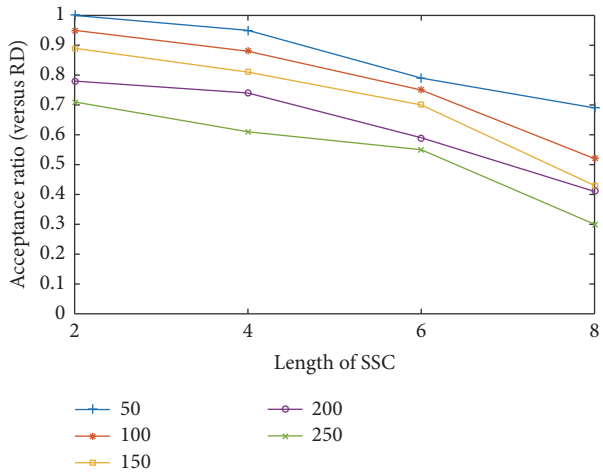

Figure 5: Acceptance ratio of TPSSC in the FT-6-B network.

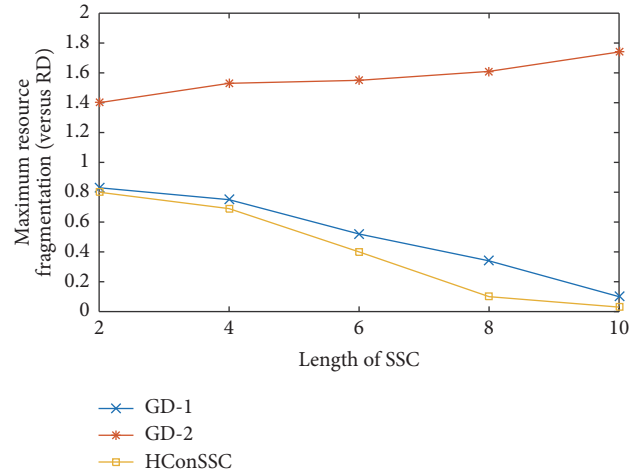

Figure 7: Comparison of the maximum resource fragmentation.

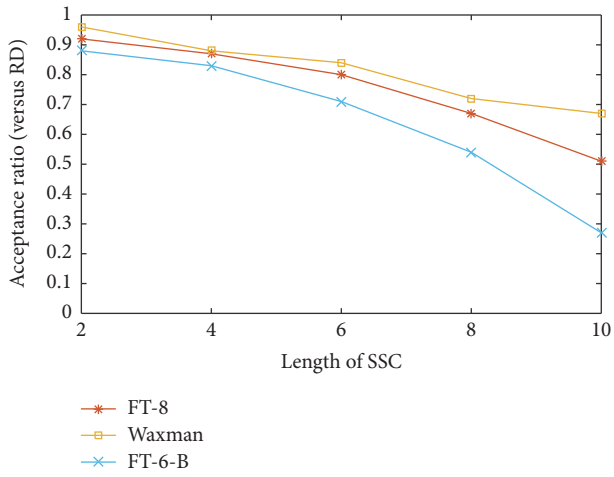

Figure 6: Acceptance ratio of TPSSC in different network scales.

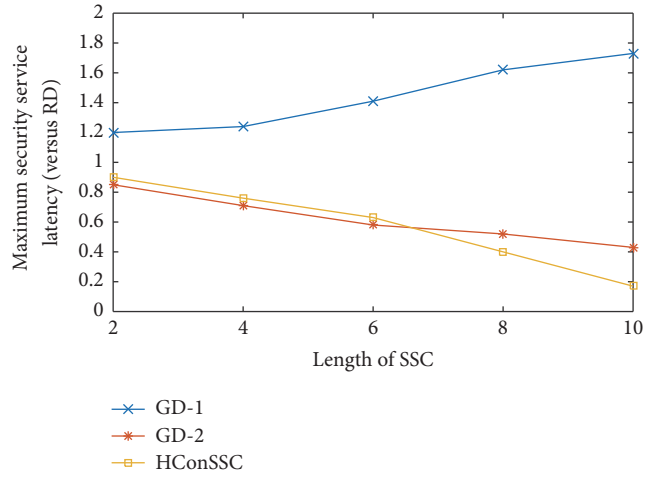

Figure 8: Comparison of the maximum security service latency.

the Waxman network. This experiment also indicates that TPSSC has good scalability and large search space, which can perform well in the network with more nodes and links. In addition, when the length of SSC exceeds 6, the acceptance ratio in the FT-6-B network drops rapidly due to insufficient resources.

6.2.2. Resources Fragmentation. Figure 7 shows the maximum resource fragmentation of GD-1, GD-2, and TPSSC versus RD. The results of TPSSC and GD-1 are significantly lower than those of RD and GD-2 since the first two try to minimize resource fragmentation when selecting service nodes. In contrast, RD does not optimize the selection of service nodes and GD-2 prefers to place multiple instances in the same service node. Moreover, the resource fragmentation of GD-2 is higher than that of RD as the length of SSC increases. The reason is that GD-2 can accept more security requests than RD, which further increases resource fragmentation. Meanwhile, the advantage of TPSSC is more and more obvious with the increasing length. This is mainly attributed to the service node selection algorithm based on bidirectional memory, which has high capability of global search and can find the near-optimal service node selection scheme to reduce resource fragmentation as far as possible.

6.2.3. Security Service Latency. Figure 8 shows the maximum security service latency (namely, the maximum end-to-end latency of flow) of GD-1, GD-2, and TPSSC versus RD. The results of GD-2 and TPSSC are better than those of RD and GD-1 since the first two aim at reducing security service latency. Conversely, GD-1 ignores latency when selecting service nodes, which may result in long security service latency since the selected service nodes may be far away from each other. It is worth noting that the maximum security service latency of TPSSC is slightly longer than that of GD-2 when the length of SSC is less than or equal to 6. It is due to the fact that TPSSC does not take into account the bandwidth of the shortest physical path between two service nodes when placing instances. In fact, it may not satisfy throughput demands of flows. So the placement of instances may still affect reducing security service latency. However, as the length of SSC increases further, GD-2 has to place

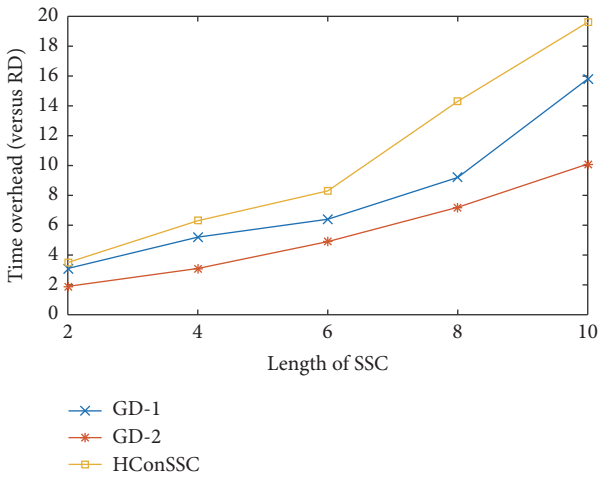

Figure 9: Comparison of the time overhead of algorithm.

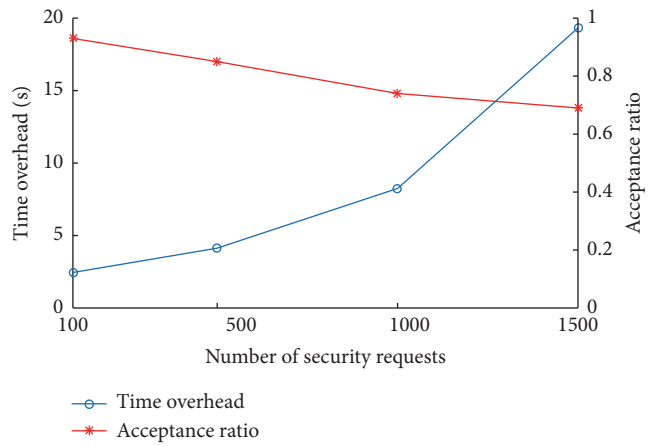

Figure 10: Performance of TPSSC under different number of security requests.

more instances, which leads to higher possibility of placing instances far away from each other. As a result, the maximum security service latency of TPSSC is shorter than GD-2.

6.2.4. Time Overhead of Algorithm. Figure 9 shows the time overhead of GD-1, GD-2, and TPSSC versus RD. As expected, the time overhead increases with the length of SSC. RD has the least time overhead since it does not consider any optimizations. When the length of SSC is 10, the time overhead of TPSSC is about 18 times higher than that of RD. But we can see from Figures 3, 7, and 8 that TPSSC can accept nearly 2 times security requests than RD and the maximum resource fragmentation is reduced by about 97%. Those data reflects that TPSSC can improve the possibility of network to accept more security requests. Meanwhile, the maximum security service latency of TPSSC is reduced by about 80%, which reflects that user experience improves significantly. From the above analysis, it is apparent that TPSSC is suitable for processing security requests in batch mode.

In order to evaluate the performance of TPSSC further, we conduct experiments with different number of security requests in the FT-8 network and measure the time overhead and the acceptance ratio. In each experiment, security requests arrive in a Poisson process. The lengths of SSCs are approximately uniformly distributed between 1 and 10. Note that the length must be a positive integer. The throughput demands are uniformly distributed between 50 and 500. Every security function has 4 instances. The CPU and memory demands of an instance are numbers with uniform distribution between 5 and 30. The throughput of an instance is a number with uniform distribution between 500 and 800. Each experiment is iterated 50 times and the arithmetic mean is reported. Figure 10 shows the results.

With the increasing number of security requests, the time overhead of TPSSC increases and the increasing rate also broadens. Therefore, it is necessary to process several batches of security requests in parallel. In addition, the acceptance ratio decreases. But it is due to the fact that the lifetime of security request is not considered in the experiment. Thus, TPSSC should recycle the released resources before running so as to allow the network to accept more subsequent security requests.

7. Conclusions and Future Work

The method of deploying SSC promises to address cost reduction and flexibility in security service delivery. Yet, SSC-DP remains a challenging problem to be tackled. In this paper, we present a novel approach to integrate SSC into delivering security service and propose a three-phase method TPSSC to find near-optimal solutions of SSC-DP. In the case that instances differ in service capacity and resource demand, TPSSC not only determines the combination of instances, placement of instances, and routing of flows, but also seeks to minimize resource consumption of service nodes, resource fragmentation, and security service latency. It facilitates delivering customized security service as well as optimizing utilization of network and security resources. Our evaluations show that although the time overhead of TPSSC is higher than RD, TPSSC has better performance in acceptance ratio, the maximum resource fragmentation, and the maximum security service latency. Moreover, compared with the two greedy algorithms, TPSSC becomes more and more superior as the length of SSC increases. As perspectives for future work, we intend to extend the evaluation of the proposed method by applying it to real network. Moreover, we plan to introduce failure-resilience mechanism into SSC maintenance for guaranteeing the continuity of security service.

Abbreviations

N_{sr}: Set of service nodes in physical network
N_{end}: Set of end nodes in physical network
E: Set of physical links
RQ: Set of security requests
V_{ins}: Set of virtual nodes representing instances
V_{end}: Set of virtual nodes representing sources and destinations of flows
L: Set of virtual links

rd_{ij}^{r}: Demand of the instance it_{ij} for resource r
c_{ij}: Throughput of the instance it_{ij}
th_{ij}^{m}: Throughput demand of the security request rq_m for the instance it_{ij}
Φ: Set of virtual node pairs representing source-destination pairs of all security requests $\Phi = \{(v_s, v_d), \ldots (v_{s'}, v_{d'})\}$
$\pi(s, d)$: Set of virtual paths between v_s and v_d which represent the source and destination of flow, respectively; that is, $\pi(s, d) = \{\pi_1, \pi_2, \ldots \pi_k\}$, in which $\pi_i = \{l(v_s, v_1), l(v_1, v_2), \ldots, l(v_j, v_d)\}$ $(1 \leq i \leq k)$ and $\bigcap_{l(v_g, v_{g'}) \in \pi_i} id(l(v_g, v_{g'})) = rq_m$
$\varphi(\pi_i)$: Set of virtual nodes along the virtual path π_i
$PT(m', n')$: Set of k-shortest paths between physical node m' and n'
x_{if}: If the instance represented by v_f is placed on service node n_i, $x_{if} = 1$, or else $x_{if} = 0$
$y_{(m,n)}^{(u,v)}$: Bandwidth assigned to virtual link $l(u, v)$ by physical link $e(m, n)$
$z_{(m',n')}^{(m,n)}$: If physical link $e(m, n)$ is a part of physical path $p(m', n')$ and $p(m', n') \in PT(m', n')$, $z_{(m',n')}^{(m,n)} = 1$, or else $z_{(m',n')}^{(m,n)} = 0$
res_r^f: Demand of the instance represented by v_f for resource r
pd_f: Processing delay of the instance represented by v_f
$c_i(r)$: The amount of resource r on service node n_i
utl_i^r: Utilization of resource r on service node n_i
fra_i: Resource fragmentation of service node n_i
$\mathrm{bw}_{(m,n)}$: Bandwidth of physical link $e(m, n)$
$\mathrm{lat}_{(m,n)}$: Latency of physical link $e(m, n)$
$\mathrm{hop}_{(m,n)}$: The number of hops of the shortest path between physical nodes m and n
$\mathrm{db}_{(u,v)}$: Bandwidth demand of virtual link $l(u, v)$.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This research was supported by National High-Tech Research and Development Project (863) of China (2012AA012704) and Zhengzhou Science and Technology Talents (131PLJRC644).

References

[1] P. Quinn and T. Nadeau, "Problem Statement for Service Function Chaining," RFC Editor RFC7498, 2015.

[2] N. McKeown, T. Anderson, H. Balakrishnan et al., "OpenFlow: enabling innovation in campus networks," *Computer Communication Review*, vol. 38, no. 2, pp. 69–74, 2008.

[3] M. Chiosi, D. Clarke, P. Willis, and A. Reid, *Network functions virtualisation-introductory, White paper*, ETSI, 2012.

[4] O. N. Foundation, *Software-defined networking: The new norm for networks, White paper*, ONF, 2012.

[5] S. Clayman, E. Maini, A. Galis, A. Manzalini, and N. Mazzocca, "The dynamic placement of virtual network functions," in *Proceedings of the IEEE Network Operations and Management Symposium (NOMS '14)*, pp. 1–9, Krakow, Poland, May 2014.

[6] J. Hwang, K. K. Ramakrishnan, and T. Wood, "NetVM: high performance and flexible networking using virtualization on commodity platforms," *IEEE Transactions on Network and Service Management*, vol. 12, no. 1, pp. 34–47, 2015.

[7] J. Martins, M. Ahmed, C. Raiciu, and F. Huici, "Enabling fast, dynamic network processing with ClickOS," in *Proceedings of the 2013 2nd ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking, HotSDN 2013*, pp. 67–72, China, August 2013.

[8] W. Zhang, G. Liu, W. Zhang et al., "OpenNetVM," in *Proceedings of the the 2016 workshop*, pp. 26–31, Florianopolis, Brazil, August 2016.

[9] J. Wilson, "Delivering security virtually everywhere with sdn and nfv," Technical Report, 2015.

[10] W. Lee, Y. Choi, and N. Kim, "Study on Virtual Service Chain for Secure Software-Defined Networking," in *Proceedings of the The 6th International Conference on Control and Automation*, pp. 177–180.

[11] J. Blendin, J. Ruckert, N. Leymann, G. Schyguda, and D. Hausheer, "Position paper: Software-defined network service chaining," in *Proceedings of the 3rd European Workshop on Software-Defined Networks, EWSDN 2014*, pp. 109–114, Hungary, September 2014.

[12] M. F. Bari, S. R. Chowdhury, R. Ahmed, and R. Boutaba, "On orchestrating virtual network functions," in *Proceedings of the 11th International Conference on Network and Service Management, CNSM 2015*, pp. 50–56, Spain, November 2015.

[13] D. Bhamare, R. Jain, M. Samaka, and A. Erbad, "A survey on service function chaining," *Journal of Network and Computer Applications*, vol. 75, pp. 138–155, 2016.

[14] D. Bhamare, R. Jain, M. Samaka, G. Vaszkun, and A. Erbad, "Multi-cloud distribution of virtual functions and dynamic service deployment: OpenADN perspective," in *Proceedings of the 2015 IEEE International Conference on Cloud Engineering, IC2E 2015*, pp. 299–304, USA, March 2015.

[15] G. Katsikas, *Realizing high performance nfv service chains [Master, thesis]*, KTH School of Information and Communication Technology, 2017, https://www.researchgate.net/publication.

[16] X.-L. Li, H.-M. Wang, B. Ding, C.-G. Guo, and X.-Y. Li, "Research and development of virtual network mapping problem," *Ruan Jian Xue Bao/Journal of Software*, vol. 23, no. 11, pp. 3009–3028, 2012.

[17] A. Dwaraki and T. Wolf, "Adaptive Service-Chain Routing for Virtual Network Functions in Software-Defined Networks," in *Proceedings of the the 2016 workshop*, pp. 32–37, Florianopolis, Brazil, August 2016.

[18] Z. Cao, M. Kodialam, and T. V. Lakshman, "Traffic steering in software defined networks: Planning and online routing," in *Proceedings of the ACM SIGCOMM 2014 Workshop on Distributed Cloud Computing, DCC 2014*, pp. 65–70, USA, August 2014.

[19] G. Xiong, Y. Hu, J. Lan, and G. Cheng, "A mechanism for configurable network service chaining and its implementation," *KSII Transactions on Internet and Information Systems*, vol. 10, no. 8, pp. 3701–3727, 2016.

[20] M. Ghaznavi, A. Khan, N. Shahriar, K. Alsubhi, R. Ahmed, and R. Boutaba, "Elastic virtual network function placement," in *Proceedings of the 4th IEEE International Conference on Cloud Networking, CloudNet 2015*, pp. 255–260, Canada, October 2015.

[21] S. Mehraghdam, M. Keller, and H. Karl, "Specifying and placing chains of virtual network functions," in *Proceedings of the 2014 3rd IEEE International Conference on Cloud Networking, CloudNet 2014*, pp. 7–13, Luxembourg, October 2014.

[22] A. Mohammadkhan, S. Ghapani, G. Liu, W. Zhang, K. K. Ramakrishnan, and T. Wood, "Virtual function placement and traffic steering in flexible and dynamic software defined networks," in *Proceedings of the 21st IEEE International Workshop on Local and Metropolitan Area Networks, Lanman 2015*, China, April 2015.

[23] B. Addis, D. Belabed, M. Bouet, and S. Secci, "Virtual network functions placement and routing optimization," in *Proceedings of the 4th IEEE International Conference on Cloud Networking, CloudNet '15*, pp. 171–177, October 2015.

[24] M. Ghaznavi, N. Shahriar, R. Ahmed, and R. Boutaba, Service function chaining simplified, CoRR abs/1601.00751, 2016.

[25] M. C. Luizelli, L. R. Bays, L. S. Buriol, M. P. Barcellos, and L. P. Gaspary, "Piecing together the NFV provisioning puzzle: efficient placement and chaining of virtual network functions," in *Proceedings of the 14th International Symposium on Integrated Network Management, IM '15*, pp. 98–106, IEEE, Toronto, Canada, May 2015.

[26] A. Gupta, M. F. Habib, U. Mandal, P. Chowdhury, M. Tornatore, and B. Mukherjee, On service-chaining strategies using virtual network functions in operator networks, CoRR abs/1611.03453, 2016.

[27] Z. Allybokus, N. Perrot, J. Leguay, L. Maggi, and E. Gourdin, "Virtual function placement for service chaining with partial orders and anti-affinity rules," *Networks*, 2017.

[28] S. D'Oro, L. Galluccio, S. Palazzo, and G. Schembra, "Exploiting congestion games to achieve distributed service chaining in NFV networks," *IEEE Journal on Selected Areas in Communications*, vol. 35, no. 2, pp. 407–420, 2017.

[29] S. Sahhaf, W. Tavernier, D. Colle, and M. Pickavet, "Network service chaining with efficient network function mapping based on service decompositions," in *Proceedings of the 1st IEEE Conference on Network Softwarization, NETSOFT 2015*, UK, April 2015.

[30] T. Lukovszki and S. Schmid, "Online admission control and embedding of service chains," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics): Preface*, vol. 9439, pp. 104–118, 2015.

[31] ETSI, Network functions virtualisation (nfv), architectural framework, ETSI GS NFV 002 V1.2.1, ETSI, 2014.

[32] J. Lischka and H. Karl, "A virtual network mapping algorithm based on subgraph isomorphism detection," in *Proceedings of the 1st ACM SIGCOMM Workshop on Virtualized Infrastructure Systems and Architectures (VISA '09)*, pp. 81–88, ACM, New York, NY, USA, 2009.

[33] R.-C. Liu, L.-C. Jiao, and H.-F. Du, "Clonal strategy algorithm based on the immune memory," *Journal of Computer Science and Technology*, vol. 20, no. 5, pp. 728–734, 2005.

[34] R. Shang and W. Ma, "Immune Clonal MO Algorithm for ZDT Problems," in *Advances in Natural Computation*, vol. 4222 of *Lecture Notes in Computer Science*, pp. 100–109, Springer Berlin Heidelberg, Berlin, Heidelberg, 2006.

[35] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 2, pp. 182–197, 2002.

[36] C. Lagos, B. Crawford, E. Cabrera, R. Soto, J.-M. Rubio, and F. Paredes, "Combining Tabu search and genetic algorithms to solve the capacitated multicommodity network flow problem," *Studies in Informatics and Control*, vol. 23, no. 3, pp. 265–276, 2014.