Hindawi
Mobile Information Systems
Volume 2017, Article ID 8098394, 13 pages
https://doi.org/10.1155/2017/8098394

Research Article

Fuzzy Theory Based Security Service Chaining for Sustainable Mobile-Edge Computing

Guanwen Li, Huachun Zhou, Bohao Feng, Guanglei Li, Taixin Li, Qi Xu, and Wei Quan

School of Electronic and Information Engineering, Beijing Jiaotong University, Beijing, China

Correspondence should be addressed to Guanwen Li; 16111011@bjtu.edu.cn

Received 9 December 2016; Accepted 16 January 2017; Published 16 February 2017

Academic Editor: Karl Andersson

Mobile-Edge Computing (MEC) is a novel and sustainable network architecture that enables energy conservation with cloud computing and network services offloading at the edge of mobile cellular networks. However, how to efficiently manage various real-time changing security functions is an essential issue which hinders the future MEC development. To address this problem, we propose a fuzzy security service chaining approach for MEC. In particular, a new architecture is designed to decouple the required security functions with the physical resources. Based on this, we present a security proxy to support compatibility to traditional security functions. Furthermore, to find the optimal order of the required security functions, we establish a fuzzy inference system (FIS) based mechanism to achieve multiple optimal objectives. Much work has been done to implement a prototype, which is used to analyze the performance by comparing with a widely used method. The results prove that the proposed FIS mechanism achieves an improved performance in terms of Inverted Generational Distance (IGD) values and execution time with respect to the compared solution.

1. Introduction

With the popularity of massive mobile terminals, the global mobile traffic has an exploding growth in the past years. Cisco statistics show the mobile data traffic grew 74% in 2015 and will further increase nearly eight times from 2016 to 2020 [1]. However, the energy efficiency of traditional mobile network infrastructures cannot afford higher and higher requirements of mobile network services with the increasing of mobile users. To alleviate this problem, Mobile-Edge Computing (MEC) was proposed as a novel and sustainable mobile network framework [2]. Different from the energy-efficient technology in wireless sensor network [3, 4], the main idea of MEC is to deploy Information Technology (IT) based services at the edges of mobile networks, which benefits from the energy management of edge clouds. Based on the technology of cloud computing, it enables mobile computation offloading and improves the energy efficiency of mobile network. Due to the above features, MEC is expected to provide the network environment with low energy consumption and high bandwidth for mobile users.

As for the MEC, energy-efficient computation offloading mechanism is one of the greatest concerns in both academia and industry areas. Orsini et al. [5] presented a programming model for mobile application developers to easily benefit from the computation offloading. Chen et al. [6] proposed a game theoretic algorithm for multiuser computation offloading. Based on a successive convex approximation technique, Sardellitti et al. [7] researched the optimization of radio and computational resources. Beck et al. [8] introduced a way to save power by offloading video transcoding from mobile devices. However, the above researches focus on the common methods of computation offloading. There are few solutions concerned about security issues by using of these methods to save energy and increase mobile traffic.

With the fast development of mobile network, the security requirements are becoming increasingly diverse for mobile traffic. As we known, there is little work focusing on flexible and real-time changing security services to meet various security requirements in MEC. Therefore, our work is motivated to propose a new solution for MEC to satisfy diversity security requirements. We introduce the security service chaining for MEC by referring to the idea of service function chaining (SFC) [9]. The security service chaining architecture can provide dynamically changing security services in terms of mobile user requirements. Moreover, some network-related vulnerabilities for mobile devices, such as Denial of Service attack [10], can be solved by the security function in our architecture. With the proposed architecture, a complex mobile security service can be split to some existing simple security functions. Due to the high performance and virtualization of the cloud computing, it is flexible to compose several required security functions for different security requirements in MEC.

In general, the main contributions of this paper are fourfold:

(1) We propose the architecture for MEC with security service chaining, which deploys security functions on a mobile-edge cloud for the mobile user equipment. Because the traffic is steered with a standard SFC way, a security function proxy for traditional service functions is also proposed to be compatible with the new architecture. Moreover, the proxy can achieve the conversion from a stateless security function to a stateful one.

(2) A graph theoretic model is used to describe the proposed security service chaining. Because the decision-making process about a security service chain is influenced by many factors, we present a fuzzy inference system (FIS) based algorithm to find the proper order of required security functions.

(3) We implement the security service chain in prototype, including packet routing with Network Service Header (NSH) encapsulation [11] and the proposed security functions proxy. There are lots of work on performance analysis of our FIS based algorithm. To make a comparison, we select a simple additive weighting (SAW) method as for the contrast algorithm. The SAW is a widely used method for multiobjective optimization. The results show that our algorithm achieves a better performance than the SAW in terms of Inverted Generational Distance (IGD) values and execution time.

The remainder of this paper is organized as follows: Section 2 discusses the related work. Section 3 presents the architecture of security service chaining for MEC. In Section 4, we formulate the security service chain by the graph based model and present a fuzzy inference system based algorithm for security service composition. Section 5 introduces the implementation of security service chain in cloud. We also present a comparison algorithm and the corresponding evaluation method to evaluate our proposed algorithm in Section 5. Section 6 concludes this paper.

2. Related Work

Recently, many researches [12–16] have focused on combining different service functions to provide a scalable service chaining for user-defined requirement. The core idea of these works is similar to the SFC. Callegati et al. [12] proposed a solution to achieve a service chaining in a cloud environment. It focused on the design of the controller, and its implementation was tested on the Mininet [13], a popular emulation platform for Software-Defined Network (SDN). Different from [12], Gember et al. [14] proposed a new architecture named Stratos, which dynamically instantiated new middle boxes on demand in SDN. Stratos addressed three main problems in steering traffic to appropriate middleboxes: elastic scaling, middlebox placement and flow distribution. FlowTags architecture proposed in [15] achieved the integrate middleboxes into SDN network. FlowTags reduced the overhead as much as possible compared with traditional SDN mechanism, but needed the FlowTags enhanced middle boxes. Qazi et al. [16] implemented dynamic traffic routing without modifying traditional middleboxes. Taking resource constraints into account, it also proposed an algorithm to generate flow paths and forwarding rules. However, these approaches have little consideration on the services deployment for mobile network.

Security service deployment is important to find a proper security service composition. There have been some Web service composition researches, which tried to use a model of graph theory to describe the QoS aware service composition. Yu and Yuan [17] described the service composition as multiconstraint optimal path problem. Sun et al. [18] proposed an improved shortest path-relax method. Jiang et al. [19] used bipartite graph optimal matching algorithm for service selection. da Silva et al. [20] proposed a graph based particle swarm optimization algorithm. Restricted by the graph theory, many existed researches focus on the single-objective optimization of service composition or convert multiobjectives optimization into a multiconstraints problem. However, the process of making decision about security service composition is actually influenced by many extra factors and some of them are in conflict with each other. The existed single-objective optimization algorithms are not suitable to solve the problem.

The fuzzy theory is widely used to solve multiobjectives optimization, which becomes more and more popular in the past few years. Especially for decision-making process about service composition, there are some typical methods based on fuzzy theory. For example, Kashyap and Tyagi [21] optimized the membership function for an efficient service selection and composition. Bakhshi et al. [22] ranked different composite service based on fuzzification of quality criteria of services. However, the existing researches mainly focus on the expressions of user preferences in fuzzy way, but a few consider the order of required function.

In this paper, in order to provide the diversity security functions for different mobile users flexibly, we present a new architecture combining the idea of service function chaining with MEC. Besides, to find the proper security service composition, we use a graph model to describe the security service deployment. Different from the existed researches, we use fuzzy method with graphic theory to find the proper order of the required security functions. Inspired by the work of Dastjerdi and Buyya [23], we choose a fuzzy inference system to achieve the process of making decision with multiple influential factors.

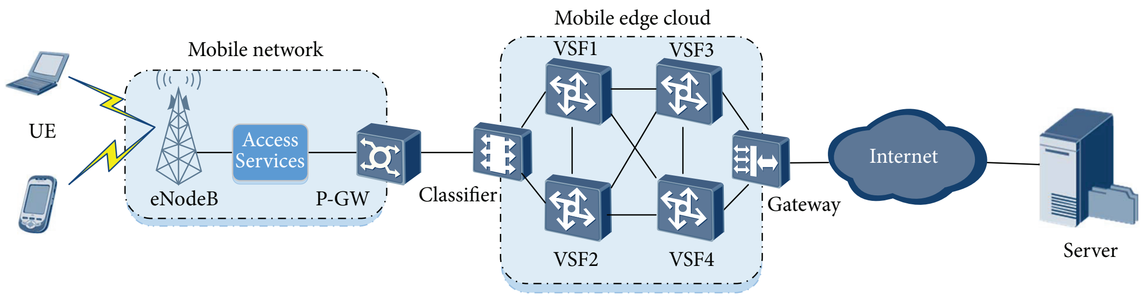


FIGURE 1: The architecture of mobile-edge security service chaining.

3. Architecture of Security Service Chaining

In this section, we propose an architecture of mobile-edge security service chaining at first. Then, we introduce main elements in the architecture in detail. Particularly, the proposed security function proxy is illustrated by an example for achieving a SFC-aware firewall.

The traditional mobile network is very complex due to integrating many different service functions. For example, in a typical 3GPP-based mobile network, the traffic from the user equipment (UE) is encapsulated in 3GPP specific tunnels terminating eventually at the packet gateway (P-GW). P-GW is the packet gateway of traditional mobile network, which is responsible to forward data to the other network. In other words, all the services are all standardized. However, it is unnecessary for each flow to pass through all security services in certain order, which may affect the resources efficiency in mobile network. Thus, we can classify these services into two categories: the access service and the security service. P-GW is the bridge between both of them. The former is necessary for mobile network. However, the latter can be implemented with the cloud technology to improve resources efficiency. Therefore, in our architecture, we divide the traditional mobile network into two independent networks: the dedicated access network and the mobile-edge cloud.

As shown in Figure 1, the proposed architecture is composed of four main parts: the UE, the simplified mobile network, the mobile-edge cloud, and the servers connected with the Internet. In this solution, the access service from enhanced NodeB (eNodeB) to P-GW is not detailed because the security service is focused on in this paper. An UE is a mobile user. The simplified mobile network is only responsible to user radio access. The mobile-edge network plays an important role in our architecture, where we deploy security functions required by mobile users. There are one classifier, some virtualized security functions (VSFs), and one gateway in mobile-edge network. The server connected with the Internet is the destination for the request of mobile user.

In this case, we assume that a mobile user is trying to use his/her cell phone to access the server pass through the Internet. At first, his/her phone, regarded as the UE, connects to the nearby eNodeB. Then, the data is transmitted from eNodeB to P-GW in the mobile network. In this architecture, the next hop of P-GW should be the mobile-edge network. The data will be assigned a security service chain which depended on the user security requirement. The following sections will describe the main elements of our proposed architecture in detail.

3.1. Access Service. The traditional user access services in mobile network are still necessary. After the eNodeB receives the user traffic from the UE, the mobile network will finish user access and authentication process as usual. Then, the traffic is steered to P-GW. Different from traditional mobile network architecture, P-GW is a communication bridge between the user access services and the mobile-edge cloud. It forwards all traffic from a mobile node to the presented cloud, which can provide the required security service chain on a virtualized platform.

3.2. Security Classifier. The traffic sent from P-GW is received by the classifier at first in the cloud. Once the classifier receives a new traffic flow, it will analyze the characteristic of this flow. The classifier identifies different flows based on their identity/type information (e.g., 5-tuple in TCP/IP). Then, there is a unique flow identifier created for this flow. This identifier is regarded as its service path identifier (SPI), which is used to mark the security service chain for this flow. At the same time, the classifier reports its classification to the control plane. After the control plane resolves the result and assigns a proper security service chain for this flow, the classifier will know its next hop. In terms of the forwarding information from control plane, the classifier encapsulates the flow with corresponding service index (SI), which indicates the first (next hop) virtualized security function. Finally, the flow is forwarded to the next hop by this classifier.

3.3. Virtualized Security Function. The virtualized security function in our architecture is a logical concept, which consists of three physical entities: the service forwarder, the security service proxy, and the security function. The following present each element of the virtualized security function, and the relations among them are illustrated as Figure 2.

3.3.1. Service Forwarder. Forwarding is the only one mission for a service forwarder, and it is a bidirectional process. On one hand, when a service forwarder receives a flow from previous service forwarder or the security classifier,

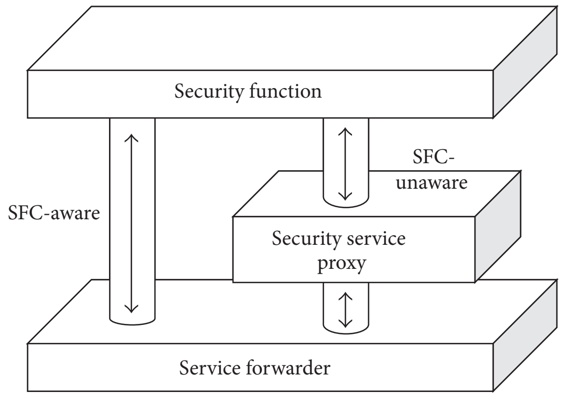


Figure 2: The relations among elements in virtualized security function.

it redirects the flow to the specific security function proxy according to the NSH encapsulation of the flow. On the other hand, it can also forward a flow back to the next proxy in the same or different service forwarder. The forwarding path is assigned by the controller.

3.3.2. Security Function Proxy. The security function proxy plays an important role in this architecture. Its main function is to support compatibility to an existing traditional (SFC-unaware) security function, because the routing way of the proposed architecture is completely different from the traditional one. Additionally, with the security function proxy, a stateless security function can be converted into a stateful one.

Apparently, the traditional security function cannot recognize a NSH encapsulated flow, because it is designed based on IP network infrastructures. We need a NSH-IP translator to translate the flows received from the service forwarder, which will be processed by a traditional security function. The security function proxy can be regarded as the required translator. It is deployed between a service forwarder and the corresponding security functions. Moreover, similar to the service forwarder, the proxy is also a bidirectional entity, which can translate NSH encapsulated flows into IP encapsulated flows and vice versa. Therefore, the security function proxy includes two kinds of interfaces: the forwarding interfaces and the service interfaces. The former achieves the translation between NSH and IP encapsulation, while the latter is used to communicate with the attached service function.

The other significant function of the proxy is to convert a stateless security function into a stateful one. Each proxy maintains a specific service state list for its attached function and updates it in time. The service state list is shown in Table 1, which includes three columns: *flow ID*, *matching info,* and *service state*. The *flow ID* is an identifier of the flow. To simplify it, we use the common 5-tuple of the flow as the *matching info*. The *service state* records the real state of security function, which can help proxy operate on the flow in the future. When the proxy receives the first packet of a new flow, it redirects the packet to the specified security function and adds the identification information of the flow to the service state list, including the flow ID (we use the SPI of this flow) and the matching info of the flow. Once the proxy gets the return flow or the processing result from the security function, it will record the service state of the flow. Furthermore, there is a timer for the proxy to count the return time or delivery time of the flow. If the threshold time is met, the service state of this flow will be written as "*deny*" automatically. Then, according to the service state, the proxy can process following packets of the flow independently to relieve the pressure of the security function.

Table 1: Service state list.

Flow ID	Matching info	Service state
1	(src_ip1, dst_ip1, protocol1, src_port1, dst_port1)	STATE1
2	(src_ip2, dst_ip2, protocol2, src_port2, dst_port2)	STATE2

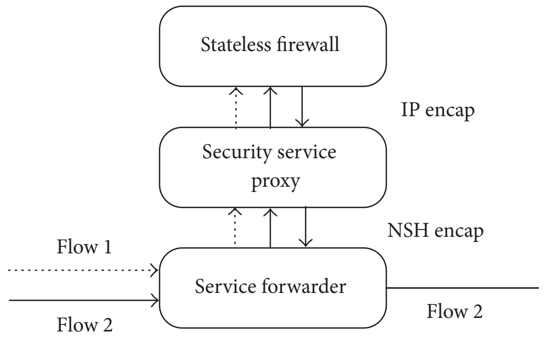


Figure 3: The workflow of a firewall with the security service proxy.

3.3.3. Security Function. The security functions are the key functional entities in a security service chain. In traditional mobile network architecture, the security services are integrated with other access services. However, in the proposed architecture, the required security services are deployed on mobile-edge network and implemented by the security software. There are two kinds of security functions: the SFC-unaware security functions and the SFC-aware ones. Most of the existed security functions are the former, which should cooperate with the security service proxy. It receives the translated flow from the proxy and executes the predefined security rules, back to the service function proxy. The other security functions are designed for NSH encapsulated flows specially. There is no need to prepare extra proxies for SFC-aware security functions if they are stateful.

The workflow of the virtualized security function is illustrated by an example of achieving a stateful firewall. Figure 3 shows how this stateless firewall is served as a stateful and SFC-aware firewall with the proxy.

There are two flows, flow 1 and flow 2, that enter into the given service forwarder at the same time. Once the service forwarder detects the new flows, it will forward them into its corresponding security service proxy.

When the proxy receives the first packet of a new flow, it will deencapsulate the NSH header and record its flow ID at first. After that, the flow is sent to the attached firewall immediately. At the same time, the proxy writes the 5-tuple information of the flow into its service state list and launches a timer. If the firewall allows this flow pass through according to its predefined rule, the first packet will return to the proxy in time. And then, the proxy will know its state and record the service state as "*allow*." Otherwise, the service state will be written as "*deny*" when time is over. In the example, we can see that flow 1 is denied but flow 2 is allowed.

Once the service state of the flow is filled in, the proxy can process the following packets of the flow without the help of firewall. In other words, the following packets of the flow are not required to be processed by the firewall any more. So, flow 2 will be forwarded to the service forwarder by this proxy next time and flow 1 will be dropped directly. Finally, for flow 2 which is allowed to be forwarded to next service function, the proxy will restore the NSH encapsulation of flow 2 before sending it out.

3.4. Gateway. The gateway in mobile-edge network is similar to P-GW in traditional mobile network. It is regarded as a packet exit of the mobile-edge network and the entrance to the Internet. In other words, any flow that wants to reach the Internet should pass through this gateway. Moreover, because the gateway is the last hop in mobile-edge network (the end of the security service chain), it checks whether the NSH header of the flow is deencapsulated or not.

In a word, the core idea of the proposed architecture is to remove the security services from traditional mobile network and deploy virtualized security functions on a cloud close to the user side. This change allows the network system to meet flexible and scalable mobile security requirements better.

4. Optimization for Security Service Chaining

In the proposed architecture for mobile communication, we present a security service chain to meet flexible user security requirements by decoupling logical security services with the physical resources in the cloud environment. Then, with the known security requirements, we need to generate a corresponding security service chaining to put required security services into a sequence. In this case, the key point is to determine the order of this sequence, which is dependent on many factors. However, it is difficult to find a best deployment solution because many of the optimal objects are in conflict with each other. For example, if we want to minimize the service processing delay, we have to choose a service node with low CPU usage, which is not beneficial to save power of the whole cloud. Therefore, the practical way to address this problem is to find a proper trade-off, which is not only to guarantee the mobile user experience but also to decrease the energy consumption as much as possible.

In this section, we firstly present a graph based model for security function deployment, which is used to find the proper order of required security functions by our proposed algorithm. Due to the performance of a given security service chain influenced by many factors, we discuss how to make a decision with more than one factor at the same time. Finally, we present a security service composition algorithm based on above considerations, which is used to choose a suitable security service chain with most acceptable performance.

4.1. Graph Based Security Service Model. The security service chaining requires a complex and integrated security service, which consists of a sequence of specialized security functions. In order to avoid the interference from others, different security functions are deployed on different virtual machines (VMs) in the cloud. Because the execution of security functions is in order and there is no security function reused in one security service chain, we use graph theory to describe all security functions and their execution sequence. The definitions are as follows.

4.1.1. Security Function Graph. According to the character of security service chaining, a directed acyclic graph is used to describe all service functions and their execution sequence. The security function graph is formalized as $G = (V, E)$, where V is the set of vertices and E is the set of edges.

4.1.2. Vertex. In the security function graph, a vertex represents an available VM in the cloud. Therefore, a virtualized security function can be deployed on this vertex. Besides, there are two special vertices, which are on behalf of the classifier and the gateway of the cloud correspondingly. The classifier vertex acts as the only entry vertex, and the gateway vertex acts as the only exit vertex. To avoid the interference, each VM can only run one security function in a security service chain. Furthermore, there may be more than one security function running on the same VM for different security service chains.

4.1.3. Edge. Each edge in the security function graph represents the connectivity between two required security functions. The edge between two different VMs is directed because the security functions deployed on them are executed in order. The security function in former vertex should be executed before the later one (next hop). If there are several security service chains, some vertices may have one more edge directed to different next hops. They can be used for different security service chains to deploy their security functions on the same vertex. Besides, there are at least one directed edge from the classifier vertex and one directed edge to the gateway vertex in a security function graph.

4.1.4. Security Function Path. Taking the way of security functions composition into account, the security function path can be expressed as a subgraph of the security function graph. Because all security service chains are started from the classifier and ended at the gateway, a security function path is regarded as a path from the classifier to the gateway in the security function graph. It is formalized as $G_{\text{sfp}} = (V_{\text{sfp}}, E_{\text{sfp}})$, where V_{sfp} is a subset of V and E_{sfp} is a subset of E.

4.1.5. Security Service Chain. The security service chain consists of several different security functions, which are deployed on the security function path. A given security

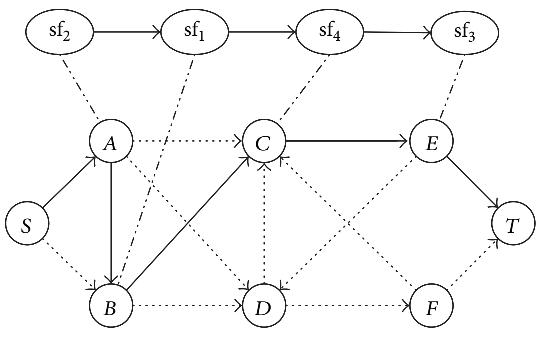


Figure 4: An example of security function graph.

Table 2: Rules for Mamdani fuzzy inference system.

Delay	CPU usage	Acceptance level
Low	High	Highly acceptable
Low	Middle	Acceptable
Low	Low	Acceptable
Middle	High	Acceptable
Middle	Middle	Uncertain
Middle	Low	Unacceptable
High	High	Unacceptable
High	Middle	Unacceptable
High	Low	Completely unacceptable

service chain is formalized as a sequence $\{\mathrm{sf}_1, \mathrm{sf}_2, \ldots, \mathrm{sf}_n\}$ where n is the total number of required security functions. There is a one to one mapping relation between the security functions of a security service chain and the vertices of its corresponding security function path. Furthermore, the size of V_{sfp} should be greater than or equal to $n+2$ if the number of security functions equals n. Although each security service chain is corresponding to a unique security function path, there may be some vertices and edges reused in the security function graph.

In terms of the graph based security service model, we can formalize a security service chain and its corresponding security function path. The order of security functions can be regarded as match between them, which is the foundation of our proposed algorithm. For example, Figure 4 shows a security function graph $G' = (V', E')$, where $V' = \{S, A, B, C, D, E, F, T\}$. The vertices S and T represent the classifier and the gateway, respectively. The other vertices are normal VMs in this graph, on which we can deploy required security functions. E' is composed of directed lines. We mark a specific security function path with full lines, while the other dotted lines represent other existed security function paths in this graph. This security function path can be described as $G'_{\mathrm{sfp}} = (V'_{\mathrm{sfp}}, E'_{\mathrm{sfp}})$, where $V'_{\mathrm{sfp}} = \{S, A, B, C, E, T\}$. We can deploy an example security service chain $\{\mathrm{sf}_2, \mathrm{sf}_1, \mathrm{sf}_4, \mathrm{sf}_3\}$ on the vertices A, B, C, E in order, and there is a match between the functions and their locations.

4.2. Constrained Optimization for Security Services. In order to meet the user security requirements as much as possible, we choose the required security functions to create the security service chain. However, the order of these functions is supposed to be optimized, which is effected by many extra factors.

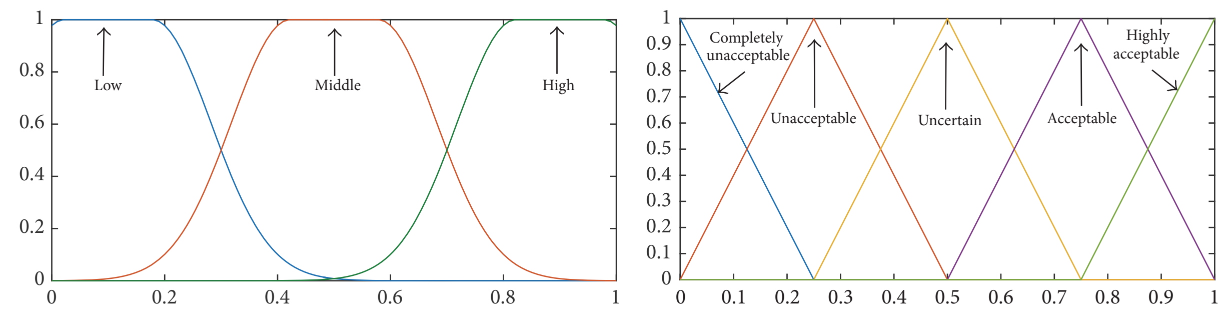
To solve this multiobjective optimization, we should assign proper weight to these objectives. Traditionally, we make the decision by our experience or experts' advices. But this way is coarse-grained because we have to use precise numbers to describe evaluations with ambiguous and semantic words. Thus, we use a fuzzy inference system to describe their weight, which uses fuzzy set theory to map input to output with defined rules [24]. There are two popular types of fuzzy inference system, *Mamdani* [25] and *Takagi-Sugeno*. We choose the Mamdani with centroid defuzzification because it is closer to the human thinking. The input is decided by the number of influential factors, but the output is unique. The fuzzy rules are set by experience or expert advices.

In this paper, we consider two typical aspects, the user experience and system energy consumption. In terms of mobile user experience, the most important factor is delay, including transmission delay and function processing delays. The former is determined when the security function path is given, while the latter is decided by the order sequence. On the other hand, we choose the CPU usage of each VM to evaluate its energy consumption. Therefore, two main optimal objectives are processing delay and CPU usage. At best, we want to choose a composition of security functions with the lowest processing delay and the highest CPU usage.

For our fuzzy inference system, the inputs are processing delay and CPU usage, which are classified in three levels: *low*, *middle,* and *high*. The membership functions of both two inputs are the same shown in Figure 5(a). The only one output is the acceptance level, which is classified in five levels and shown in Figure 5(b). Both input value and output value are normalized numbers. In this case, the output value "zero" in output means the result is completely unacceptable, whereas the value "one" means it is highly acceptable. The rules are as shown in Table 2, which are set by our experience. For example, the first rule means if the processing delay is low and the CPU usage is high, it is regarded as a highly acceptable result.

As illustrated in Figure 6, the proposed fuzzy inference system is able to describe the acceptable level of the output in terms of the CPU usage and the processing delay. In other words, this fuzzy inference system describes a more reasonable weight of these influence factors.

4.3. FIS Based Security Service Composition. For a given security function path $G_{\mathrm{sfp}} = (V_{\mathrm{sfp}}, E_{\mathrm{sfp}})$, there are x candidate VM nodes $\{\mathrm{vm}_1, \mathrm{vm}_2, \ldots, \mathrm{vm}_x\}$ which can employ required security functions $\{\mathrm{sf}_1, \mathrm{sf}_2, \ldots, \mathrm{sf}_x\}$. Taking many different influential factors into account, such as their processing delay and CPU usage, we need to find an appropriate deployment of these security functions. Therefore, based on the



(a) The shape of input membership function

(b) The shape of output membership function

FIGURE 5: The input and output membership functions for the purposed fuzzy inference system.

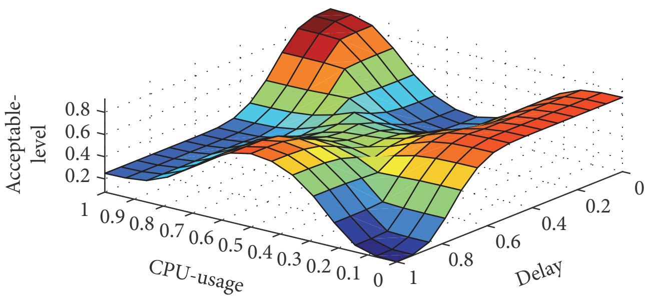


FIGURE 6: The acceptable level with different CPU usage and processing delay.

fuzzy inference system, we propose a new algorithm to find the match between required security functions and their locations (orders). In this way, we assume that the chosen influential factors of each security function are measured previously.

The algorithm to find the most suitable security function composition is presented as follows.

Step 1. Set the influential factors of each security function F_k $(k = 1, 2, \ldots, y)$ for each alternative VM node N_j $(j = 1, 2, \ldots, x)$ with respect to security function S_i $(i = 1, 2, \ldots, x)$ denoted by $R_k = X_{ijk}$.

Step 2. Compute relative membership degrees for the VM nodes and the security functions with different influential factors. For a certain influential factor $(k = u)$, if the bigger influential factor is better, its relative membership degree is shown in

$$r_{ijk}\big|_{k=u} = \frac{x_{ijk}\big|_{k=u} - x_{\min}}{x_{\max} - x_{\min}}, \tag{1}$$

where $x_{\min}$ is the minimum $x_{ijk}|_{k=u}$ and $x_{\max}$ is the maximum $x_{ijk}|_{k=u}$.

Otherwise, its relative membership degree is shown in

$$r_{ijk}\big|_{k=u} = \frac{x_{\max} - x_{ijk}\big|_{k=u}}{x_{\max} - x_{\min}}. \tag{2}$$

Step 3. Set relative membership degrees of the chosen influential factors $(r_{ij1}, r_{ij2}, \ldots, r_{ijy})$ as the input of our fuzzy inference system; then we can get the acceptable level r_{ij} in output.

Step 4. Therefore, the fuzzy relation matrix is shown in

$$R = \left[r_{ij}\right]_{x*x}. \tag{3}$$

Step 5. Use the Hungarian method [26] to find the independent zero element in R, which represents the matching relation between the security function and its corresponding VM.

Step 6. Deploy these security functions in order.

5. Implementation and Evaluation

5.1. Prototype Implementation. For a typical scenario of 3GPP mobile LTE network, our proposed mobile-edge architecture should include an UE, some necessary access services for LTE, P-GW, required security functions, and the interface to the Internet. However, to make it clear, we only implement the proposed mobile-edge cloud in our prototype.

As illustrated in Figure 7, the prototype consists of a classifier, a gateway, four security functions with their corresponding proxies and service forwarders. The classifier is responsible to create a flow and assign an appropriate security

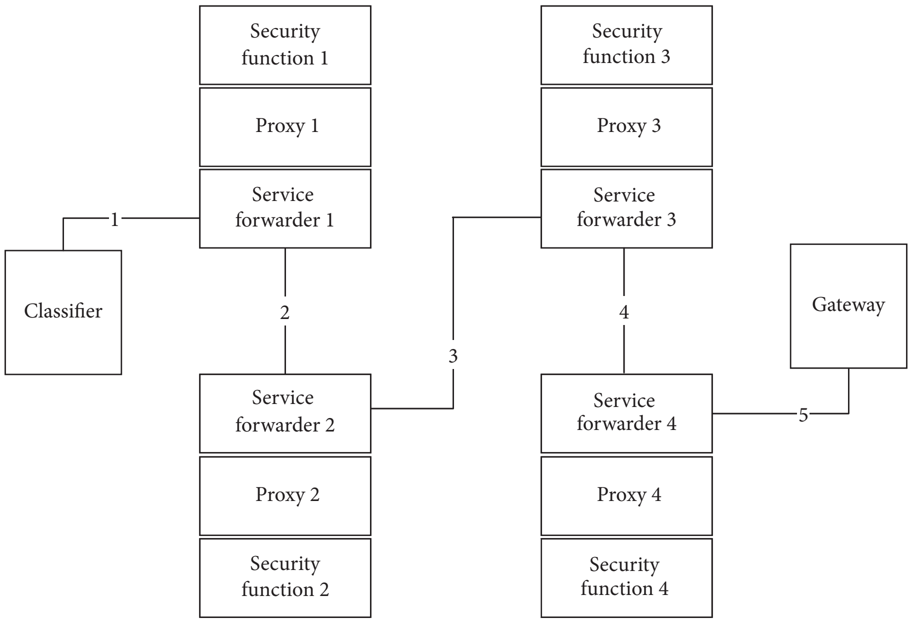


Figure 7: The test bed of mobile-edge cloud.

service chain. The flow will pass through the security functions 1–4 in order. Finally, the gateway receives this flow. There is a security function proxy between each service forwarder and its security function to translate the flow from IP to NSH encapsulation and vice versa. Each of them runs on a VM with dual-core CPU, 2 GB memories and several NICs, which is created by Kernel-based Virtual Machine (KVM) in OpenStack.

The classifier and service forwarders are implemented by OpenVSwitch [27]. OpenVSwitch is a software switch, which is able to forward specific flow based on predefined rules. In our prototype, we use the SDN controller POX [28] to control the behavior of the OpenVSwitch. To achieve SFC routing, service forwarders are equipped with Nshkmod [29]. Nshkmod is an open source Linux kernel module implementation of Network Service Header, which is used to encapsulate and deencapsulate NSH for service flows. The implementation of the security function proxy is also based on OpenVSwitch and Nshkmod, in order to remove/insert NSH encapsulation of packets and redirect them to the next service forwarder or the receiver.

To achieve the standard SFC routing between two different VMs, each virtual NSH interface (named nshx), created by Nshkmod, is bound up with a NIC (named ethx) in an ovs net-bridge. The ovs net-bridge has the ability to forward packets from a virtual NSH interface to its related NIC in order to encapsulate it by NSH header and vice versa. Additionally, the data transmission between two virtual NSH interfaces is accomplished by a special tunnel in Linux kernel. In this way, a service forwarder can communicate with its corresponding security function proxy. The security function proxy for a service forwarder can send/receive data to/from its corresponding security function in a traditional routing way (IP routing).

Table 3: Processing delay of the required functions on different VMS.

	VM1	VM2	VM3	VM4
Service 1	15	12.5	14.6	13.8
Service 2	10.4	10.8	11.5	11.8
Service 3	8.7	9.1	8.2	8.9
Service 4	5.8	8.1	7.3	6.4

Table 4: CPU usage of different VMS with the required functions.

	VM1	VM2	VM3	VM4
Service 1	83.1	85.2	84.7	84.4
Service 2	25.6	24.2	25.5	27.1
Service 3	22.1	24.7	23.9	23.3
Service 4	68.5	72.3	71.9	70.8

Considering the most common security services in mobile network, we choose the firewall, network address translator (NAT), Deep Packet Inspection (DPI), and load balancer (LB) as an example. The firewall and NAT are both implemented by iptables [30], which is integrated into the Linux kernel. The DPI is implemented by nDPI [31], a famous open source software. The LB is implemented by Linux Virtual Server (LVS) [32], a popular technology of load balance server.

We assume that security function path is decided as shown in Figure 7. We choose two important influence factors mentioned in Section 4, the processing delay of security functions, and the CPU usage of their corresponding VMs. Based on the prototype, we measure these parameters for each security function on four decided VMs independently. The measurement result of processing delay is shown in Table 3 and the CPU usage is shown in Table 4.

5.2. Contrast Algorithm for Security Service Composition. In order to evaluate the performance of our proposed algorithm, a contrast algorithm is presented to solve the problem of security service composition. The simple additive weighting (SAW) [33] is widely used for service composition, which transforms a multiobjective optimization into single-objective optimization. However, the SAW algorithm is traditionally used without fuzzy theory. To make it reasonable to evaluate our fuzzy algorithm, we present a fuzzy SAW based algorithm, which combines the SAW algorithm with fuzzy relation theory.

The problem of security service composition and its conditions are the same to Section 4.3. The detailed algorithm is presented as follows.

Step 1. Set the performance evaluations of each security function P_k $(k = 1, 2, \ldots, y)$ for each alternative VM node N_j $(j = 1, 2, \ldots, x)$ with respect to security function S_i $(i = 1, 2, \ldots, x)$ denoted by $R_k = X_{ijk}$.

Step 2. Compute relative membership degrees for the VM nodes and the security functions with different performance evaluations. For a certain performance evaluation $(k = u)$, if the bigger performance evaluation is better, its relative membership degree is shown in

$$r_{ijk}\big|_{k=u} = \frac{x_{ijk}\big|_{k=u} - x_{\min}}{x_{\max} - x_{\min}}, \tag{4}$$

where $x_{\min}$ is the minimum $x_{ijk}|_{k=u}$ and $x_{\max}$ is the maximum $x_{ijk}|_{k=u}$.

Otherwise, its relative membership degree is shown in

$$r_{ijk}\big|_{k=u} = \frac{x_{\max} - x_{ijk}\big|_{k=u}}{x_{\max} - x_{\min}}. \tag{5}$$

Step 3. Set the fuzzy weight $W = (w_1, w_2, \ldots, w_y)$, where $\sum w_k = 1$ $(k = 1, 2, \ldots, y)$ in terms of our experience or experts' advices.

Step 4. According to the fuzzy weight W, compute the aggregate fuzzy performance evaluations, which is shown in

$$r_{ij} = \sum w_k \cdot r_{ijk}. \tag{6}$$

Step 5. Therefore, the fuzzy relation matrix is shown in

$$R = \left[r_{ij}\right]_{x*x}. \tag{7}$$

Step 6. Use the Hungarian method to find the independent zero element in R, which represents the matching relation between the security function and its corresponding VM.

Step 7. Deploy these security functions in order.

5.3. Evaluation Method and Criteria. A single-objective optimization algorithm can be easily evaluated by calculating the best value, because its target is to find only one extreme value of the given problem. However, it is very hard to evaluate a multiobjective optimization algorithm. Usually, a multiobjective optimization problem is influenced by some factors which are in conflict with each other. It is impossible to find the solution to satisfy every optimal objective in one time. Therefore, for a multiobjective optimization, there exist a lot of Pareto optimal solutions. The Pareto optimal solution cannot be improved in any objective for this problem if we do not degrade the other objectives. The Pareto front is a set of all the Pareto optimal outcomes, which can be used to describe the best solution of a multiobjective optimization problem.

Therefore, it is necessary to find the Pareto front before we evaluate the proposed algorithm. Because the Pareto front of the proposed security service composition problem is unknown, it is supposed to choose an existing algorithm to estimate the Pareto front. The genetic algorithm is one of the most popular algorithms for multiobjective optimization, which can help us obtain the Pareto front. In this paper, we use NSGA-II [34] algorithm, which is a kind of improved genetic algorithm, to estimate the Pareto front. Because the Pareto front achieved by this algorithm is finite and dynamically changed every run time, we run this algorithm for 10 times under the same situation to collect the Pareto points as much as possible.

The NSGA-II algorithm we used is decided by three main parameters: the population size, the maximum number of generation, and the fitness function for our problem. There is no comprehensive solution for setting the population size and the maximum number of generations. For example, with a smaller population, it is fast to find the solution but may lead to local optimum, while a larger population may cause low efficiency. The decision of maximum number of generations is similar to the population size. Therefore, we evaluate the algorithm under several situations with different settings of the population size and the maximum number of generations.

On the other hand, it is important to choose a suitable fitness function for our problem, which has a great effect on the rate of convergence of the algorithm. In terms of the model proposed in Section 4.3, we assume that there are k influential factors of each security function and j alternative VM node with its respective security function where $j \in \{1, 2, \ldots, x\}$ and $k \in \{1, 2, \ldots, y\}$. The value of the kth influential factors on each VM is denoted as q_{kj}. Moreover, these factors can be classified into two categories: the positive factor Q^{+} and the negative factor Q^{-}. For the former, larger value means the better, while for the latter, small value means the better. Then, the fitness functions f_k are shown in

$$f_k = \begin{cases} \displaystyle\sum_{j=1}^{x} q_{kj}, & \text{if } q_{kj} \in Q^{-}, \\ \displaystyle-\sum_{j=1}^{x} q_{kj}, & \text{if } q_{kj} \in Q^{+}. \end{cases} \tag{8}$$

According to the calculation outcomes, we can obtain a large number of Pareto optimal points. Each point is a possible optimal solution for our problem, which is regarded as the estimated Pareto optimal point. With the estimated Pareto front, the next step is to evaluate our fuzzy based optimization

Table 5: Relative membership degree of processing delay.

	VM1	VM2	VM3	VM4
Function 1	0	0.27	0.04	0.13
Function 2	0.5	0.46	0.38	0.35
Function 3	0.68	0.64	0.74	0.66
Function 4	1	0.75	0.84	0.93

Table 6: Relative membership degree of CPU usage.

	VM1	VM2	VM3	VM4
Function 1	0.97	0	0.99	0.99
Function 2	0.06	0.03	0.05	0.02
Function 3	0	0.04	0.03	0.02
Function 4	0.73	0.80	0.79	0.77

algorithm. The reasonable way to evaluate a multiobjective optimization algorithm is to make a comparison between the calculated result and the Pareto front. There is a widely used indicator to describe the difference between the estimated Pareto front and the result calculated by our algorithm, called IGD. IGD represents the distance from the result achieved by our algorithm to the estimated Pareto front. It is obvious that a smaller value is better, because it is closer to the Pareto front. IGD can be calculated by

$$\mathrm{IGD} = \frac{1}{n}\sqrt{\sum_{i=1}^{n}\left|x_i - x\right|^2}, \tag{9}$$

where n is the total number of Pareto optimal points, x is the influential vector of the result, and x_i is the influential factor vector of the ith Pareto optimal point.

Therefore, we can use NSGA-II algorithm to find the Pareto front under a certain situation at first. Then, we calculate the IGD value of a given security service composition algorithm according to the achieved Pareto front. Finally, we evaluate the performances of different algorithms by comparing the IGD values between them, and the algorithm with smaller IGD values is better.

5.4. Experimental Results. According to the proposed algorithm, we calculate the corresponding relative membership degrees for each measurement firstly, as shown in Tables 5 and 6, respectively.

Then, we use the fuzzy inference system presented in Section 4 to make a trade-off decision between these two factors. Set the relative membership degrees of processing delay and CPU usage as the input; then we get the fuzzy relation matrix R.

$$R_{\mathrm{FIS}} = \begin{bmatrix} 0.8975 & 0.7436 & 0.8946 & 0.8946 \\ 0.5714 & 0.6329 & 0.7153 & 0.7163 \\ 0.1315 & 0.1865 & 0.1255 & 0.1522 \\ 0.2557 & 0.3166 & 0.2940 & 0.2654 \end{bmatrix}. \tag{10}$$

Table 7: Matching relation between services and VMS.

	VM1	VM2	VM3	VM4
Function 1	*	0	*	*
Function 2	0	*	*	*
Function 3	*	*	0	*
Function 4	*	*	*	0

Table 8: Reference matching relation between services and VMS.

	VM1	VM2	VM3	VM4
Function 1	*	0	*	*
Function 2	*	*	*	0
Function 3	0	*	*	*
Function 4	*	*	0	*

Finally, we use Hungarian method to find out the matching relation between security functions and VMs, which is marked by "0" in Table 7.

According to Table 7, we should deploy the security function 1 on VM2, function 2 on VM1, function 3 on VM3, and service 4 on VM4. Therefore, the service deployment order of the given security service chain is function 2, function 1, function 3, and function 4. The total processing delay in this way equals 37.5 and its corresponding total CPU usage equals 205.5.

To evaluate our algorithm, the same problem is also calculated by the fuzzy SAW based algorithm presented in Section 5.2. We set the reference fuzzy weight $W_{\mathrm{SAW}} = (W_{\mathrm{delay}}, W_{\mathrm{CPU}}) = (0.6, 0.4)$. It means the influence of processing delay is a little larger than that of CPU usage, which is similar to rules we defined for our fuzzy inference system. Different from the algorithm, we use a precise number to describe our subjective feeling in this way. With this empirical fuzzy weight, we use (6) to compute the aggregate membership degree, which is shown in (11).

$$r_{ij} = \sum w_k \cdot r_{ijk} = 0.6 r_{ij1} + 0.4 r_{ij2}. \tag{11}$$

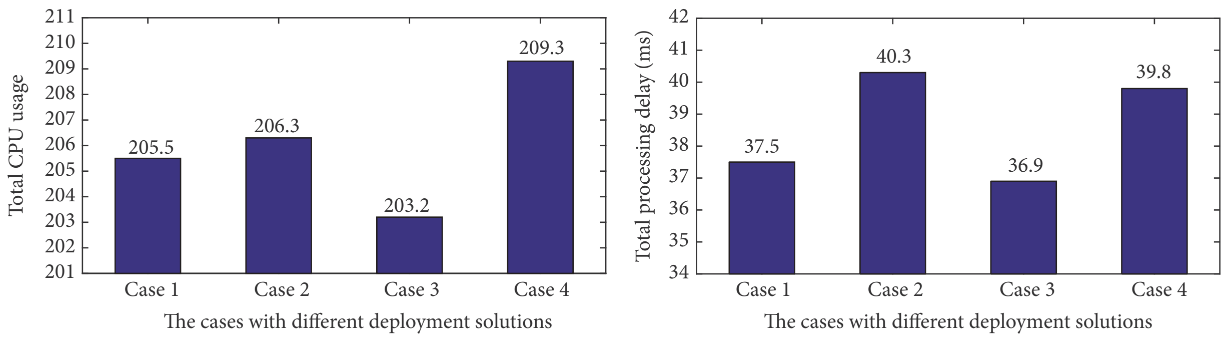
So, the reference fuzzy relation matrix is

$$R_{\mathrm{SAW}} = \left[r_{ij}\right]_{4*4} = \begin{bmatrix} 0.388 & 0.162 & 0.420 & 0.474 \\ 0.324 & 0.288 & 0.248 & 0.242 \\ 0.408 & 0.400 & 0.456 & 0.404 \\ 0.532 & 0.500 & 0.518 & 0.531 \end{bmatrix}. \tag{12}$$

Similarly, we use Hungarian method to find out the reference matching relation between security functions and VMs, which is marked by the "0" in Table 8.

According to Table 8, we should deploy the security function 1 on VM2, function 2 on VM4, function 3 on VM1, and service 4 on VM3. Therefore, the service deployment order of the given security service chain is function 3, function 1, function 4, and function 2. The total processing delay in this way equals 40.3 and its corresponding total CPU usage equals 206.3.

To evaluate our algorithm objectively, we also calculate two extreme cases in this scenario, which are the shortest



(a) Total processing delay different cases

(b) Total CPU usage for different cases

FIGURE 8: Comparisons between total processing delay and total CPU usage for different cases.

total processing delay case and the maximum total CPU usage case. In the former case, the total processing delay equals 36.9 and its corresponding total CPU usage equals 203.2. While the total processing delay of the latter case equals 39.8 and its total CPU usage equals 209.3.

For convenience, we mark the four cases as cases 1–4: case 1 represents the deployment with FIS based algorithm, case 2 represents the deployment with fuzzy SAW based algorithm, and case 3 and case 4 represent shortest processing delay case and maximum CPU usage case, respectively.

To make it clear, we discuss the two factors, total processing delay and total CPU usage equal independently. Figure 8(a) shows the total processing delays for different cases. We can see case 1 is very similar to case 3, which represents the minimal total processing delay. It is obvious that the deployment with our algorithm can obtain lower processing delay, which is close to our object. While case 2 and case 4 are much higher than them, particularly case 2 is even higher than case 4. Maybe case 2 is the worst case in this scenario because it obtains neither the shortest processing delay nor the highest CPU usage. Although the subjective feeling of contrast algorithm seems like FIS based algorithm, the result shows that it is extremely different. The reason is the criteria for these factors are semantic and complexity. It is difficult to describe the criteria with only one precise number. The result also proves that our fuzzy inference system is useful to solve this problem.

On the other hand, Figure 8(b) shows the total CPU usage for different cases. Although there is no other case similar to case 4, which represents the maximum total CPU usage, case 1 and case 2 are much higher than case 3. Case 3 may be the lowest CPU usage case in this scenario at the cost of its shortest processing delay. Case 2 is very close to case 1, so both of them can be the deployment candidate with higher CPU usage. However, according to the above analysis about processing delay, case 2 is the worst solution. Therefore, the most proper deployment is case 1. In other words, our algorithm provides a best trade-off between these two factors. It proves that the algorithm we proposed is useful to create an appropriate security service chain with multiple influence factors.

5.5. Performance Analysis. In this section, we firstly estimate the Pareto front of our problem by the NSGA-II algorithm, because there are four VMs needed to deploy security functions in the above scenario and two influential factors to be considered. So, in terms of (4), the fitness functions are supposed to be

$$\begin{aligned} f_1 &= q_{11} + q_{12} + q_{13} + q_{14}, \\ f_2 &= q_{21} + q_{22} + q_{23} + q_{24}. \end{aligned} \tag{13}$$

In order to make it closer to the real scenario, we set these decision variables of two fitness functions (objective functions) based on the measurements in Tables 3 and 4. Thus, the range of these decision variables is as follows:

$$\begin{aligned} q_{11} &\in (12, 16), \\ q_{12} &\in (10, 12), \\ q_{13} &\in (8, 10), \\ q_{14} &\in (5, 9), \\ q_{21} &\in (83, 86), \\ q_{22} &\in (24, 28), \\ q_{23} &\in (22, 25), \\ q_{24} &\in (68, 73). \end{aligned} \tag{14}$$

Because the population size and the maximum number of generations are difficult to determine in this scenario, we use four different pairs of values for the population size and the maximum number of generation to evaluate the performance of algorithms. These pairs of values are set as (20, 100), (50, 200), (50, 100), and (100, 200) where the former represents its population size and the latter represents its maximum number of generation.

After we obtain all the Pareto optimal points for all the situations, we calculate their IGD values with (8). In this paper, we calculate the IGD values for both FIS based algorithm and fuzzy SAW based algorithm under above four

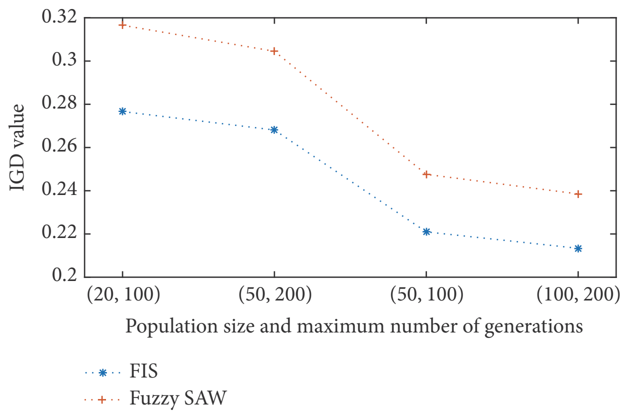


Figure 9: Comparisons of the IGD values of different security service composition algorithms.

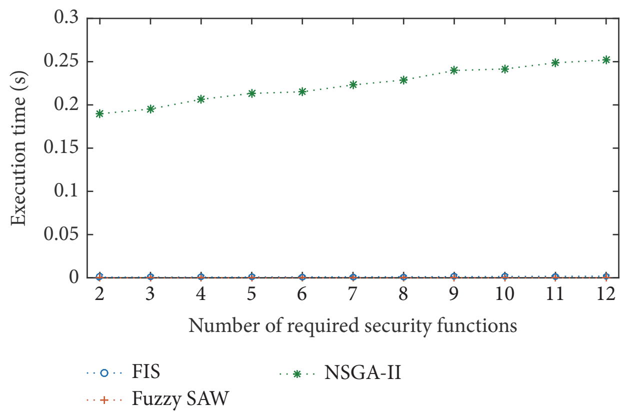


Figure 10: Comparing the execution time of different security service composition algorithms.

different situations. As shown in Figure 9, the dotted line with plus signs represents IGD values of proposed FIS based algorithm, while the dotted line with asterisk represents that of the fuzzy SAW based algorithm. Each line consisted of four points and each point represents one of the situations. For example, the first point in the dotted line with plus signs means that IGD value of FIS based algorithm is calculated with population size 20 and maximum number of generations 100.

It is obvious that the dotted line with asterisk is always lower than the dotted line with plus signs. Additionally, there are downturns for both of the dotted lines shown in Figure 9. The probable reason is that the population size is not large enough to find adequate number of Pareto optimal points by NSGA-II algorithm. Moreover, too much generations with gene mutation also make the IGD values larger because the IGD value means the stability of chosen samples. In general, IGD values of FIS based algorithm are always smaller than that of fuzzy SAW based algorithm under each situation. It indicates that the result achieved by our proposed algorithm is closer to the best solution (Pareto front). Thus, the solutions of FIS based algorithm are better than fuzzy SAW algorithm.

Besides, we also make a comparison about the execution time of different algorithms shown in Figure 10. The dotted line with circles represents the execution time of proposed FIS based algorithm, the dotted line with plus signs represents that of comparison algorithm, and the dotted line with asterisk represents that of the NSGA-II algorithm. In this experiment, we calculate the execution time of each algorithm for different number of required security functions in a security service chain, which is from 2 to 12. Evidently, the execution time of NSGA-II is always much larger than other algorithms. With the number of required security functions increasing, the execution time is also increasing. However, according to our achieved results, the performance of FIS based algorithm is similar to fuzzy SAW based algorithm, and both the execution times of FIS based and fuzzy SAW based algorithms are lower than 1 ms. Thus, it is acceptable for security service chaining to choose either the FIS based algorithm or the fuzzy SAW based one in terms of the execution time.

Based on the above analysis, the overall performance of FIS based algorithm is better than that of the fuzzy SAW based algorithm.

6. Conclusion

In this paper, we present a new architecture for sustainable MEC with the idea of security service chaining. The proposed service chaining can meet more diversity and flexible mobile user requirements. Moreover, we present a security function proxy, which provides the compatibility to traditional security functions in the new architecture. We also propose an algorithm to find a proper way to composite required security functions with multiple influential factors. In detail, the security service chain is described by a graph model, and a fuzzy inference system is adopted to find the suitable order of the required security functions. Finally, with an implemented prototype, we calculate and analyze the performance of our proposed algorithm by comparing a widely used SAW algorithm. The experimental results obviously prove the proposed FIS based algorithm is better than the contrast one with a common evaluation criterion.

Competing Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

Acknowledgments

This work is supported by National High Technology of China ("863 program") under Grant no. 2015AA015702, National Basic Research Program of China ("973 program") under Grant no. 2013CB329101, NSFC of China under Grant no. 61271202 and no. 61602030, and NSAF under Grant no. U1530118.

References

[1] Cisco, *Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update, 2015–2020*, CISCO, 2016.

[2] M. Patel, B. Naughton, C. Chan et al., "Mobile-edge computing—introductory technical white paper," Tech. Rep. 1, The European Telecommunications Standards Institute, 2014.

[3] C. E. Weng, V. Sharma, H. C. Chen, and C. H. Mao, "PEER: proximity-based energy-efficient routing algorithm for wireless sensor networks," *Journal of Internet Services and Information Security*, vol. 6, no. 1, pp. 47–56, 2016.

[4] S. Aram, I. Khosa, and E. Pasero, "Conserving energy through neural prediction of sensed data," *Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications*, vol. 6, no. 1, pp. 74–97, 2015.

[5] G. Orsini, D. Bade, and W. Lamersdorf, "Computing at the mobile edge: designing elastic android applications for computation offloading," in *Proceedings of the 8th IFIP Wireless and Mobile Networking Conference (WMNC '15)*, pp. 112–119, Munich, Germany, 2015.

[6] X. Chen, L. Jiao, W. Li, and X. Fu, "Efficient multi-user computation offloading for mobile-edge cloud computing," *IEEE/ACM Transactions on Networking*, vol. 24, no. 5, pp. 2795–2808, 2015.

[7] S. Sardellitti, G. Scutari, and S. Barbarossa, "Joint optimization of radio and computational resources for multicell mobile-edge computing," *IEEE Transactions on Signal and Information Processing over Networks*, vol. 1, no. 2, pp. 89–103, 2015.

[8] M. T. Beck, S. Feld, A. Fichtner, C. Linnhoff-Popien, and T. Schimper, "ME-VoLTE: network functions for energy-efficient video transcoding at the mobile edge," in *Proceedings of the 18th International Conference on Intelligence in Next Generation Networks (ICIN '15)*, Paris, France, February 2015.

[9] E. J. Halpern and E. C. Pignataro, "Service function chaining (sfc) architecture," Tech. Rep. RFC 7665, Internet Engineering Task Force (IETF), 2015.

[10] B. Rashidi and C. Fung, "A survey of android security threats and defenses," *Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications*, vol. 6, no. 3, pp. 3–35, 2015.

[11] P. Quinn and U. Elzur, "Network service header," IETF Internet-Draft, 2016, https://datatracker.ietf.org/doc/draft-ietf-sfc-nsh/04/.

[12] F. Callegati, W. Cerroni, C. Contoli, and G. Santandrea, "Dynamic chaining of virtual network functions in cloud-based edge networks," in *Proceedings of the 1st IEEE Conference on Network Softwarization (NETSOFT '15)*, IEEE, London, UK, April 2015.

[13] B. Lantz, B. Heller, and N. McKeown, "A network in a laptop: rapid prototyping for software-defined networks," in *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks (HotNets-IX '10)*, pp. 1–6, Monterey, Calif, USA, 2010.

[14] A. Gember, A. Krishnamurthy, S. S. John et al., "Stratos: a network-aware orchestration layer for virtual middleboxes in clouds," https://arxiv.org/abs/1305.0209.

[15] S. K. Fayazbakhsh, L. Chiang, V. Sekar, M. Yu, and J. C. Mogul, "Enforcing network-wide policies in the presence of dynamic middlebox actions using flowtags," in *Proceedings of the 11th USENIX Conference on Networked Systems Design and Implementation (NSDI '14)*, pp. 543–546, Seattle, Wash, USA, 2014.

[16] Z. A. Qazi, C.-C. Tu, L. Chiang, R. Miao, V. Sekar, and M. Yu, "SIMPLE-fying middlebox policy enforcement using SDN," in *Proceedings of the ACM SIGCOMM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM '13)*, pp. 27–38, Hong Kong, China, August 2013.

[17] C. Yu and M.-T. Yuan, "Web service composition using graph model," in *Proceedings of the 2nd IEEE International Conference on Information Management and Engineering (ICIME '10)*, pp. 656–660, IEEE, Chengdu, China, April 2010.

[18] P. J. Sun, P. C. Zhang, W. R. Li, X. J. Guo, and J. Feng, "Sky-MCSP-R: an efficient graph-based Web service composition approach," in *Proceedings of the 20th Asia-Pacific Software Engineering Conference (APSEC '13)*, pp. 589–594, IEEE, Bangkok, Thailand, December 2013.

[19] C.-Q. Jiang, W. Du, and C.-C. Yi, "A method of web service composition based on bipartite graph optimal matching and QoS," in *Proceedings of the International Conference on Internet Technology and Applications (ITAP '10)*, pp. 1–5, IEEE, Wuhan, China, August 2010.

[20] S. A. da Silva, H. Ma, and M. J. Zhang, "A graph-based Particle Swarm Optimisation approach to QoS-aware web service composition and selection," in *Proceedings of the IEEE Congress on Evolutionary Computation (CEC '14)*, pp. 3127–3134, Beijing, China, July 2014.

[21] N. Kashyap and K. Tyagi, "Dynamic composition of web services based on Qos parameters using fuzzy logic," in *Proceedings of the 2nd International Conference on Advances in Computer Engineering and Applications (ICACEA '15)*, pp. 778–782, March 2015.

[22] M. Bakhshi, F. Mardukhi, and N. Nematbakhsh, "A fuzzy-based approach for selecting the optimal composition of services according to user preferences," in *Proceedings of the 2nd International Conference on Computer and Automation Engineering (ICCAE '10)*, vol. 5, February 2010.

[23] A. V. Dastjerdi and R. Buyya, "Compatibility-aware cloud service composition under fuzzy preferences of users," *IEEE Transactions on Cloud Computing*, vol. 2, no. 1, pp. 1–13, 2014.

[24] L. A. Zadeh, "Fuzzy logic = computing with words," *IEEE Transactions on Fuzzy Systems*, vol. 4, no. 2, pp. 103–111, 1996.

[25] E. H. Mamdani and S. Assilian, "Experiment in linguistic synthesis with a fuzzy logic controller," *International Journal of Man-Machine Studies*, vol. 7, no. 1, pp. 1–13, 1975.

[26] H. W. Kuhn, "The Hungarian method for the assignment problem," *Naval Research Logistics Quarterly*, vol. 2, pp. 83–97, 1955.

[27] B. Pfaff, J. Pettit, T. Koponen et al., "The design and implementation of open vswitch," in *Proceedings of the 12th USENIX Conference on Networked Systems Design and Implementation (NSDI '15)*, pp. 117–130, Oakland, Calif, USA, 2015.

[28] N. Repo and contributors, The pox controller, 2012–2014, https://github.com/noxrepo/pox.

[29] R. Nakamura, Nshkmod, 2015-2016, https://github.com/upa/nshkmod.

[30] Netfilter Core Team and Contributors, iptables, 1999–2014, http://www.netfilter.org/projects/iptables/index.html.

[31] The Ntop Team, nDPI, 2016, http://www.ntop.org/.

[32] Linux Virtual Server, 2016, http://www.linuxvirtualserver.org/.

[33] S.-J. Chen and C.-L. Hwang, *Fuzzy multiple attribute decision making*, vol. 375 of *Lecture Notes in Economics and Mathematical Systems*, Springer-Verlag, Berlin, Germany, 1992.

[34] 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.