Cluster Computing (2023) 26:423–436
https://doi.org/10.1007/s10586-022-03670-8

SFCSim: a network function virtualization resource allocation simulation platform

Lingyi Xu[1] · Hefei Hu[2] · Yuanan Liu[1]

Received: 28 August 2021 / Revised: 12 May 2022 / Accepted: 22 June 2022 / Published online: 12 July 2022

Abstract
As a novel network architecture, network function virtualization greatly improves the flexibility of service deployment but also increases the complexity of network function virtualization resource allocation (NFV-RA). Many scholars have studied the above problems and put forward corresponding schemes. These schemes are often individually designed and are difficult to compare. A general and efficient simulation platform for the NFV-RA problem is conducive to integration, comparison and analysis of these schemes and speeds up research progress. This paper presents a general simulation platform SFCSim for NFV-RA. SFCSim adopts the discrete-time event scheduling engine, which can support the simulation of static and dynamic deployment of service function chain, service migration, mobility management and other scenarios. Finally, a load-balancing shortest path and three heuristic embedding algorithms are implemented in the platform for the service function chain embedding problem can be used as the benchmark algorithm, and SFCSim is fully compared with the Alevin to verify the performance of the platform

Keywords Network function virtualization · Service function chain · Resource allocation · NFV orchestration · Heuristic algorithm

1 Introduction

Network function virtualization (NFV) [1] is a novel network architecture, which decouples software and hardware, and realizes various expensive network functions (NF) such as firewall, load balancer, deep packet detector and so on through virtualization technology on the commercial server. These network functions are referred to as virtual network functions (VNFs). This new network architecture makes the deployment of network services more flexible and scalable, and greatly reduces the capital (CAPEX) and operative (OPEX) costs of network operators.

Generally, network traffic needs to pass through a set of network functions in an orderly way and is defined as the Service Function Chain (SFC) [2]. Although NFV provides great flexibility, how to realize fast, scalable and flexible SFC composition and VNFs resource allocation is a new challenge, namely NFV resource allocation (NFV-RA) [3]. NFV-RA problem is usually divided into three stages: VNFs Chain Composing, VNF Forwarding Graph Embedding and VNFs Scheduling. Each of these three stages is an NP-hard problem. Many scholars have proposed their resource allocation optimization schemes [4–8] for these three stages, and certain effects have been achieved. Due to the special pertinence of these schemes, some large network simulation platforms such as NS2 [9] and OPNET [10] and Virtual Network Embedding (VNE) simulation platforms Alevin [11] and VNE Simulator [12] are usually unable to test these schemes. Therefore, the scheme proposer needs to write a lot of code and test data sets for the simulation scene, which makes these schemes difficult to implement and test. A general simulation platform for NFV-RA can help to compare and analyze the characteristics of these different schemes and accelerate the research progress.

✉ Hefei Hu
huhefei@bupt.edu.cn

1 School of Electronic Engineering, Beijing University of Posts and Telecom-munications, Beijing, China

2 School of Information and Communications Engineering, Beijing University of Posts and Telecommunications, Beijing, China

A large number of models and simulation designs for NFV-RA problems have emerged in the existing research work. These designs include resource allocation and scheduling at the three levels: physical network layer, virtualization layer and service function chain layer [3], and optimize the resource allocation of multiple scenarios such as static SFC deployment [7, 8], dynamic SFC deployment [13, 14], VNF migration [15], SFC deployment under mobile scenario [16]. In these scenarios, the parameters and restrictions considered in the model and simulation design vary greatly, such as the types of nodes [4, 7], the processing [17], propagation [18], queuing [19] and other delays of the physical link, the resource scaling [20] and resource sharing [14] of the VNF, and the composition of the user SFC [6, 7]. How to unify the above different, diverse and highly complex simulation designs and realize the general network function virtualization resource allocation simulation in various scenarios has become an urgent demand.

In this paper, a general and novel NFV-RA simulation platform SFCSim is proposed. SFCSim is an open-source simulation platform based on Python language, which can provide consistent underlying data structure and general network resource scheduling management function for rapid integration of various NFV-RA algorithms. The main contributions of this paper are as follows:

1. A general simulation platform is designed for the NFV-RA problem. As far as we know, SFCSim is the first NFV-RA simulation platform written in Python language, and can complete the static and dynamic SFC deployment, service migration, mobility management, and dynamic traffic simulation.
2. SFCSim is based on the NetworkX Python Framework. It summarizes most papers on NFV-RA, unifies the underlying data structure of the problem and provides a consistent interface. Meanwhile, the simulation platform expands the powerful analysis capability and algorithm of NetworkX.
3. SFCSim adopts the discrete-time event scheduling method, which can realize the automatic management of network resources and the maintenance of database between cycle intervals, and complete the SFC scheduling according to the custom algorithm.
4. A load-balancing shortest path and three heuristic scheduling algorithms are implemented in SFCSim. At the same time, SFCSim is fully compared with the VNE simulation platform Alevin to verify the performance of the platform.

The rest of this paper is structured as follows. Section 2 reviews the related work. Then Sect. 3 describes the SFCSim system architecture. Section 4 describes the implementation details and embedding algorithms, and Sect. 5 discusses the test scenarios and evaluation of the proposed platform. Finally, the paper is summarized in Sect. 6.

2 Related work

At present, the related work falls into the following two categories :(1) NFV-RA simulation and evaluation platform; (2) NFV-RA simulation design.

2.1 NFV-RA simulation and emulation platform

NFV-RA simulation and evaluation platform can be used to test various algorithms for resource allocation and SFC embedding, and compare and evaluate the performance of algorithms. The traditional large-scale network simulation platforms NS-2 [9] and NS-3 [10], can simulate various protocols (TCP, UDP) in the transmission layer and network layer between different wired and wireless media through the discrete event simulator mechanism. OPNET [21] is committed to simulating various real carrier level network access and transmission equipment. Networkx [22] provides a unified and efficient storage mode of graphs and various complex network analysis algorithms, which are widely used to create, operate and study the structure, dynamics and functions of complex networks. Although this software can provide general underlying network models and various network simulation plug-ins and algorithms, they are not a simulation platform specifically for NFV-RA problems and cannot meet the needs of network resource management and resource scheduling in the NFV environment, and cannot realize the simulation of NFV-RA problems efficiently and conveniently. In traditional Virtual Network Embedding (VNE) problem, there have been some special simulation tools, such as Alevin [11], VNE Simulator [12], and ViNE-Yard [23], which can be used to set the VNE simulation environment and parameters under the unified underlying resource architecture, and realize the evaluation and comparison of some VNE algorithms. These platforms can only simulate a small portion of NFV-RA, most of them focus on static SFC deployment and VNF forwarding graph embedding, and the lack of documentation and other instructions makes it difficult for these simulation tools to meet usage requirements. Table 1 shows the characteristics of these simulation platforms.

2.2 NFV-RA simulation design

NFV-RA simulation design and some key assumptions can affect the specific process of simulation. Many scholars have studied the simulation design of NFV-RA. The

Table 1 MAIN NETWORK SIMULATION PLATFORMS AND CHARACTERISTICS

Tools	Characteristics	language
Ns-2	Discrete event-driven Internet	C++
Ns-3	Environment simulation	
OPNET	Simulation analysis of complex network performance behavior	C++
NetworkX	Creation, manipulation, and study of complex network	python
Alevin	Simulation framework of VNE algorithm	java
ViNE-Yard	Simulation framework of VNE algorithm	C
VNE Simulator	VNE simulator	C
NFV-Orchestration	NFV orchestration algorithms simulation	java
middlebox-placement	VNF orchestration simulation tool	C++

modeling and simulation design of NFV-RA usually needs to consider the physical network layer, virtualization layer and service function chain layer. In terms of physical network layer model design, most studies believe that the underlying network includes server and switch nodes [26–30]. In recent years, with the development of edge computing and distributed data center, some scholars only include server nodes in the underlying network of simulation models [31–34]. For the setting of physical network links, on the basis of considering the limitation of link resources, many studies take link delay as an important measure. In the data center environment, the processing delay of data packets [17] and the queuing delay varying with the link load [18] need to be considered. In the edge computing environment, the propagation delay of the link cannot be ignored with the increase of the distance between nodes [19]. The virtualization layer contains different types of VNF entities instantiated on physical nodes. These VNFs usually occupy certain hardware resources. Some studies believe that the resources of VNF can be shared by users [35, 36]. In terms of SFC design, there are two models, that is, duplicate VNFs in SFC [4, 6, 29] and duplicate VNFs are not allowed [7]. In some studies, multiple VNF instances are allowed to be deployed for the NF of one SFC [6, 7]. Considering the scalability of the existing virtualization technology based on the Hypervisor or Container to the resources of the virtualization layer, many research works ignored the limitation of the resource allocation of the virtualization layer, and directly considered the occupation of the resources of the service function chain in the physical network layer [17, 18, 37], and only a few works [38, 39] considered the resource limitation of the virtualization layer.

To sum up, there is no unified simulation platform for NFV-RA, and the existing simulation design has obvious characteristics of differentiation, diversification and high complexity. Aiming at the above problems, a general NFV-RA simulation experiment platform was established, which could well complete the simulation of resource allocation and SFC deployment in the NFV scenario.

3 System architecture

SFCSim is a general simulation evaluation platform for NFV-RA problems. The underlying design of SFCSim is based on the open-source network simulation tool NetworkX and extends the related functions. This chapter describes the overall architecture of the simulation system and the simulation engine.

3.1 Overall system architecture

Figure 1 describes the system architecture of SFCSim. SFCSim adopts the modular design, which is divided into the substrate network module, VNF module, SFC module and scheduler module. The first three modules provide consistent data structure, basic operation methods and interfaces, while the scheduler module describes the interoperability of other modules, and provides basic deployment and orchestration methods, data storage and recording functions, used to implement various upper-level algorithms.

1) Substrate network module: The substrate network module redefines the data structure of network nodes based on the original Graph structure of NetworkX. The underlying network contains server and switch nodes. Each node contains CPU, memory, storage and other attributes. Nodes are connected by links that contain bandwidth, delay and other attributes. The nodes and links of the substrate network provide management interfaces, which realize the management of infrastructure through network interface.
2) VNF module: The VNF module provides templates for various NFs, defines the resources occupied by these NF, the traffic scaling factor, and the resources consumed of processing per unit of traffic. Through this module, various VNFs can be instantiated on the server nodes. VNF module also provides the interface of VNF attribute and state management for Scheduler.

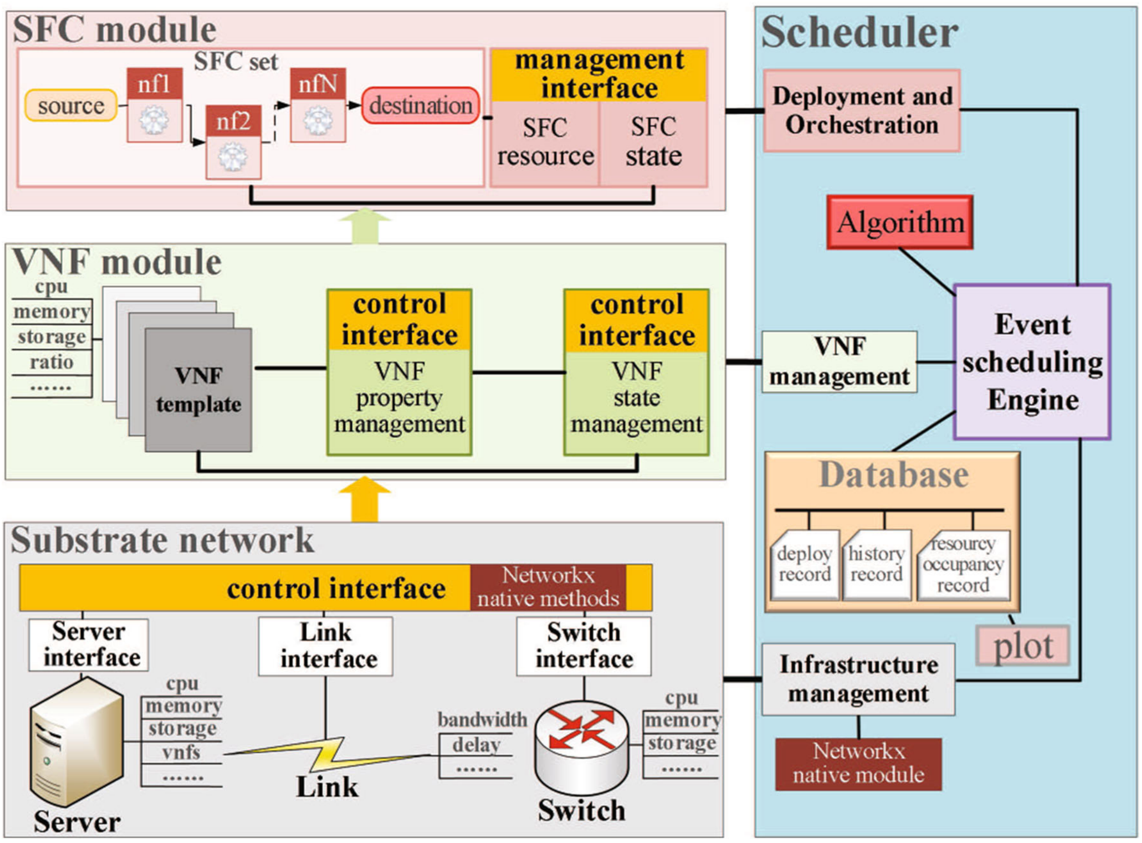


Fig. 1 SFCSim system architecture

3) SFC module: The SFC module contains a set of user services, and each user service represents an SFC. This module can support static SFC, dynamic SFC, mobile SFC (describing user movement), dynamic traffic SFC (describing traffic change), and their hybrid simulation. The SFC module implements the functions of SFC resource, state management and related interfaces, which can support the deployment and orchestration of the Scheduler.
4) Scheduler module: The scheduler module implements infrastructure management, VNF management, SFC deployment and orchestration functions, and can easily expand various scheduling algorithms. The core of the scheduler is a discrete-time event scheduling engine. According to the upper schemes, the engine completes the deployment and orchestra of user services, and the management of underlying infrastructure resources between each discrete time interval. The scheduler also includes a drawing tool based on the Python Matplotlib library, which can display the changes of network topology and resources in real-time.

3.2 Discrete-time event scheduling engine

The core of SFCSim is a discrete-time event scheduling engine, which can automatically update the resources of the underlying infrastructure and virtual network functions at each discrete-time interval, manage and orchestrate service according to the upper user-defined algorithm. Figure 2 shows the workflow of the scheduling engine. The time scale of the simulation platform is divided into time intervals T. T is an abstract concept, which can be arbitrarily defined to simulate the network state on different time scales. The scheduling engine works after the end of the last discrete time interval and before the start of the next discrete time interval, the network and service state are stable within each time interval, and the change of state only occurs at the switching moment of the time interval. After entering the scheduling engine, it will first query the events that need to be processed in the current cycle. These events include infrastructure resource update events, network internal traffic migration events, SFC state maintenance events, SFC life cycle management events. Finally, the SFC with a state change event is processed according to user-defined policies. The whole scheduling process will automatically update and maintain records.

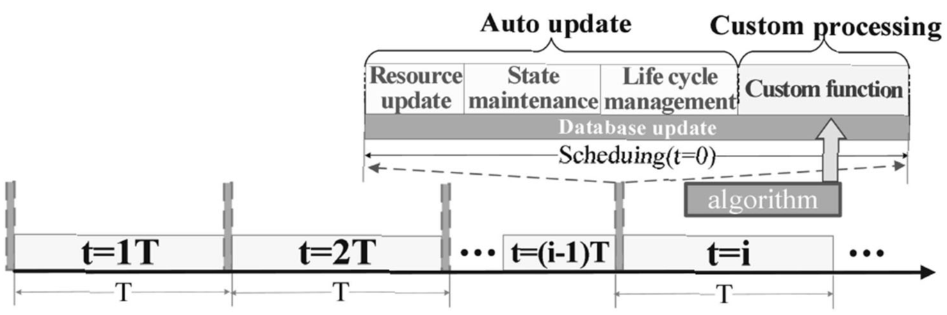


Fig. 2 The flow of the slot scheduling engine

4 Implements

4.1 Substrate network

The substrate network is the network class, which is most closely associated with NetworkX simulation tools. It encapsulates two network nodes, Server and Switch, and defines basic resource control methods based on the Graph structure of NetworkX. The network class can use a lot of original Graph generators and analytical optimization methods. Two real network topologies, NSFNET and CERNNET2, are added to SFCSim, as shown in Fig. 3.

4.2 Virtualized network function

SFCSim defines a unified data structure of VNF through vnf_type class, VNF can be deployed on nodes and occupy CPU, memory and storage resources of nodes. Considering the traffic processing ability of different types of VNF. For example, Firewalls will discard some risky traffic, and Classifiers will add new encapsulated header, etc., which will lead to traffic size changes. The basic linear traffic processing resource coefficient and traffic scaling factor are provided according to most of the papers [4, 7, 16, 17, 19, 25, 29], that is, for any size of traffic f_{in}, the required resources of type $i(i \in \{\text{cpu}, \text{memory}, \text{storage}, ...\})$ and the traffic size f_{out} after processing are refer to (1) and (2).

$$r_i = k_i * f_{\text{in}}(i \in \{\text{cpu}, \text{memory}, \text{storage}, ...\}) \tag{1}$$

$$f_{\text{out}} = ratio * f_{\text{in}}(ratio > 0) \tag{2}$$

Where k_i is the resource processing coefficient of type $i\,(i \in \{\text{cpu}, \text{memory}, \text{storage}, ...\})$. Formula 1 represents the node resources required to process traffic, and formula 2 represents the change of the traffic after VNF processing.

4.3 Customer service

SFC class defines the unified data structure of customer service. Present SFCSim supports static SFC class, dynamic SFC class, mobile SFC class and dynamic traffic SFC class, covering the common service scenarios in the current paper, and supporting the simulation of different user service types. As shown in Fig. 4 (a), static SFC is the base class, contains the source and destination nodes of the server as well as a group of network functions that need to pass through, and the traffic demand remain unchanged. The dynamic SFC contains a finite life cycle which triggers the extinction event at the end of the life cycle. Mobile SFC is aimed at user mobile scenario, where the source node will move over time and trigger mobile events, as shown in Fig. 4(b). The traffic required by dynamic traffic SFC will change over time, triggering the flow change event, as shown in Fig. 4(c). SFC class can realize the automatic calculation of service resource requirements and the automatic maintenance of service time.

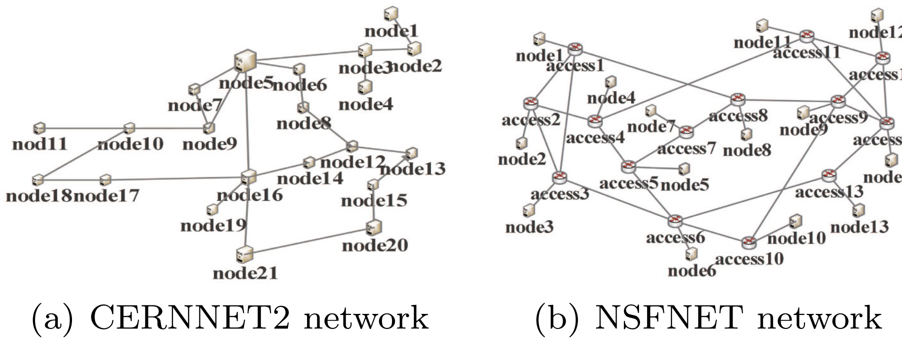


Fig. 3 Real network topologies

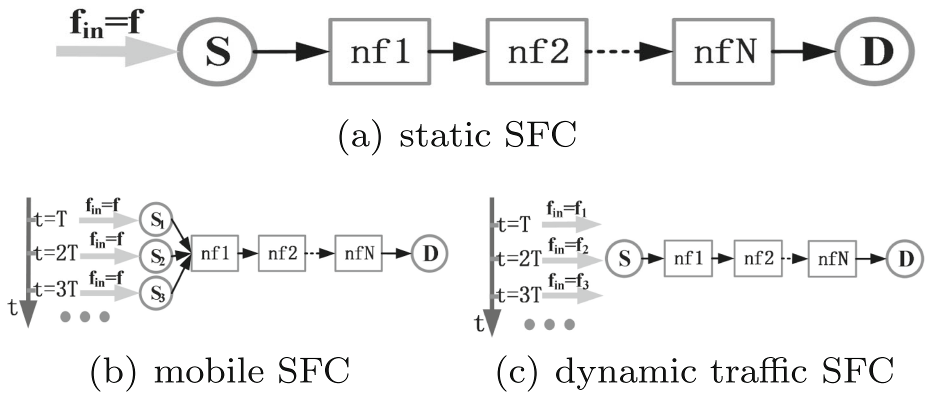


Fig. 4 SFC module

4.4 Scheduler

The scheduler class implements the scheduling engine of SFCSim, and completes the deployment and scheduling of SFC according to the upper algorithm and lower interface. The scheduler class can realize the automatic management of resources, including the automatic allocation and recycling of underlying network resources, the instantiation and deletion of VNF, the maintenance of SFC status, and the update and maintenance of data records. The process is shown in Fig. 5. After the initialization of network and VNF, new user service SFCs can be added at the beginning of any cycle T, and the SFC can be deployed according to the polices in the custom algorithm to record the corresponding events, and then the events can be responded according to the user-defined processing. Through the interface provided by the scheduler, the simulator no longer needs to consider the complex underlying data structure and resource maintenance, to focus more on the implementation of the upper algorithm.

4.5 Embedding algorithms

To test the performance of the simulation platform, a load-balanced shortest path embedding algorithm (LBSP) and three heuristic algorithms (Tabu Search [40], Simulated Annealing [41] and Particle Swarm Optimization [42] are implemented in the platform for the most basic and important VNF-FGE problem of NFV-RA. The three heuristic algorithms construct the solution space by k-shortest path algorithm (KSP-SS), generate the initial solution by LBSP or random method, and perform the same digital neighborhood operation to search for the optimal embedding strategy in their unique way.

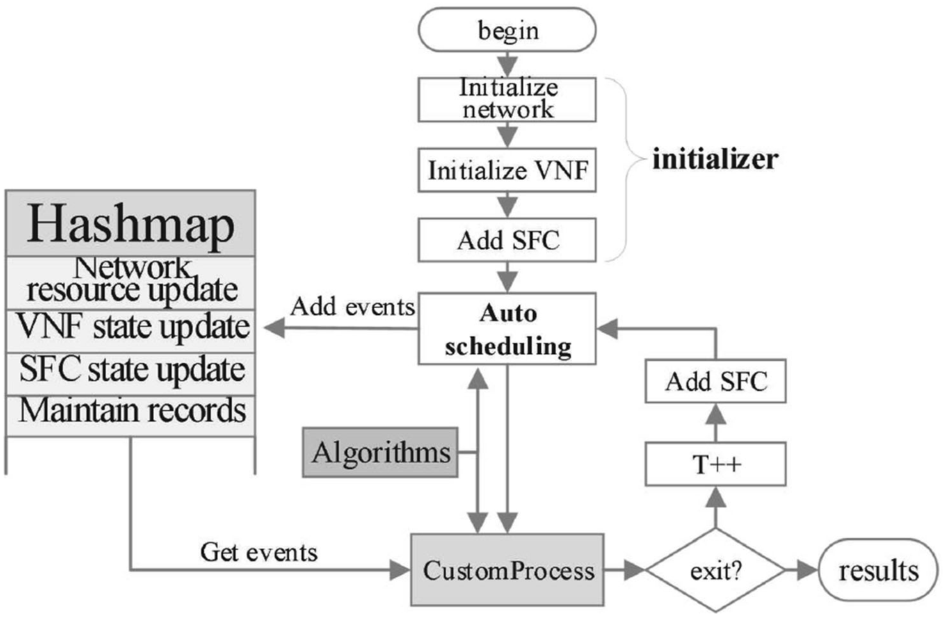


Fig. 5 Process of scheduling

4.5.1 Solution space construction strategy based on k-shortest path algorithm (KSP-SS)

KSP-SS algorithm is used to construct the solution space as complete as possible, and remove the poor solutions. Based on the physical link delay, the algorithm calculates the K shortest paths from the source to the destination node and finds all the solutions of the user SFCs on the K shortest paths. To improve the continuity of solution space, these solutions are arranged in the order of service function displacement in physical nodes. As shown in Fig. 6, the source and destination of SFC are on the node in and node out respectively, and need to go through nf1 and nf2. There are two physical links in the underlying network from the node in to the out, one of which has a total delay of 2.2ms, and the other has a total delay of 2.5ms. When $K = 2$, these two paths are selected and NFs are embedded on the nodes in order, and the single user SFC solution space is constructed with a natural number coding scheme (solution $1 - n$). All the embedding schemes of SFCS form a complete solution space Ω.

4.5.2 Digital neighborhood operations

After the KSP-SS algorithm is used to construct the solution space, the same neighborhood operations are defined for the three heuristic algorithms. Since the solution space of SFC is encoded in the digital form of $1 - n$, and has good continuity, three digital neighborhood operations are defined on it. As shown in Fig. 6, solution 2 moves forward to solution 3 is defined as +1, moving backward to solution 1, defined as -1, and when it cancels embedding (transferring resources to other users) is defined as 0. The encoding method is shown in Fig. 7. All user services SFCs form a complete candidate solution set after neighborhood operation.

4.5.3 An embedding strategy based on load-balancing shortest path (LBSP)

The initial solution is the key factor affecting the performance of heuristic algorithms. Usually, a random initial solution generation strategy is used, that is, a combination of embedding schemes is randomly selected in each SFC's solution space to form the initial solution. Besides the random selection scheme, a new embedded strategy based on the load-balancing shortest path is proposed (LBSP), which can be used as the initial solution. The core of LBSP is to distribute user services evenly to the underlying network as far as possible, extend the minimum traffic virtual link, and combine the NFs of both ends of the maximum traffic virtual link. As shown in Fig. 8, the algorithm can be divided into three cases. When the number of NFs m required by the SFC is equal to the number of shortest path servers n, it is embedded according to one-to-one correspondence (nf1 $\rightarrow$ in, nf2 $\rightarrow$ node1, nf3 $\rightarrow$ out). When $m < n$, extend the minimum traffic virtual link (extend nf2-nf3, embed it in node1-node2-out), and then embed it according to $m = n$. When $m > n$, the NFs at both ends of the virtual link with the maximum traffic are combined in cycle and embed on the same server until $m = n$ (nf2, nf3 $\rightarrow$ node1). LBSP can effectively distribute the network

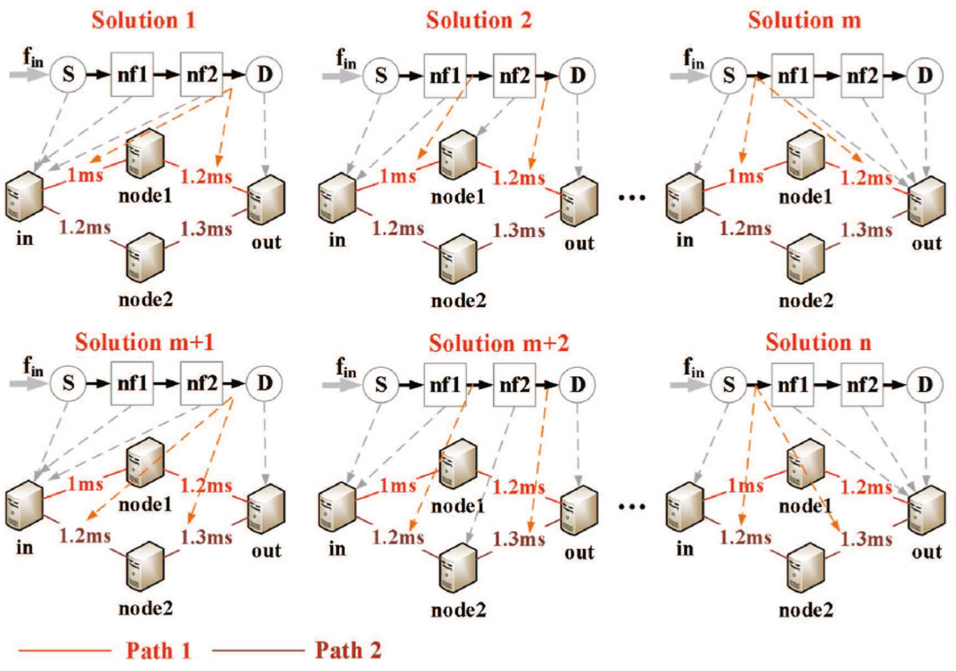


Fig. 6 Solution space strategy when K=2

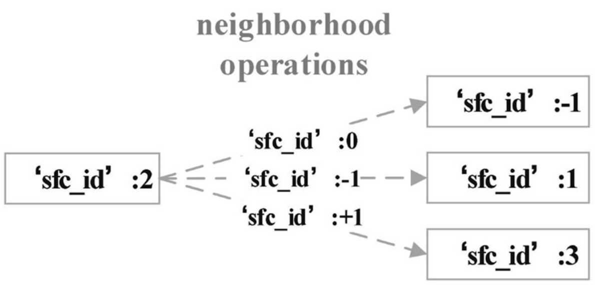


Fig. 7 Three digital neighborhood operations

load evenly, reduce resource consumption, and generate a more uniform and high-quality initial solution.

4.5.4 Tabu search algorithm

Tabu search (TS) is a search algorithm with short-term memory function, which can avoid falling into local optimal effectively by flexible memory method of Tabu List. The initial solution of TS is generated by LBSP algorithm. and the corresponding digital coding in KSP-SS solution space is found. As shown in Fig. 9, the candidate solution is generated by performing digital neighborhood operation on each SFC in the initial solution, while the records in the Tabu List and the neighborhood operation are negative to each other to prevent the repeated neighborhood operation within the period of the Tabu Length. Algorithm 1 describes the main steps of Tabu Search. Lines 5-16 are the update operations of Tabu List. If the Tabu List contains the neighborhood operations corresponding to the optimal solution, the optimal solution or the suboptimal solution will be used according to whether the violation conditions are met.

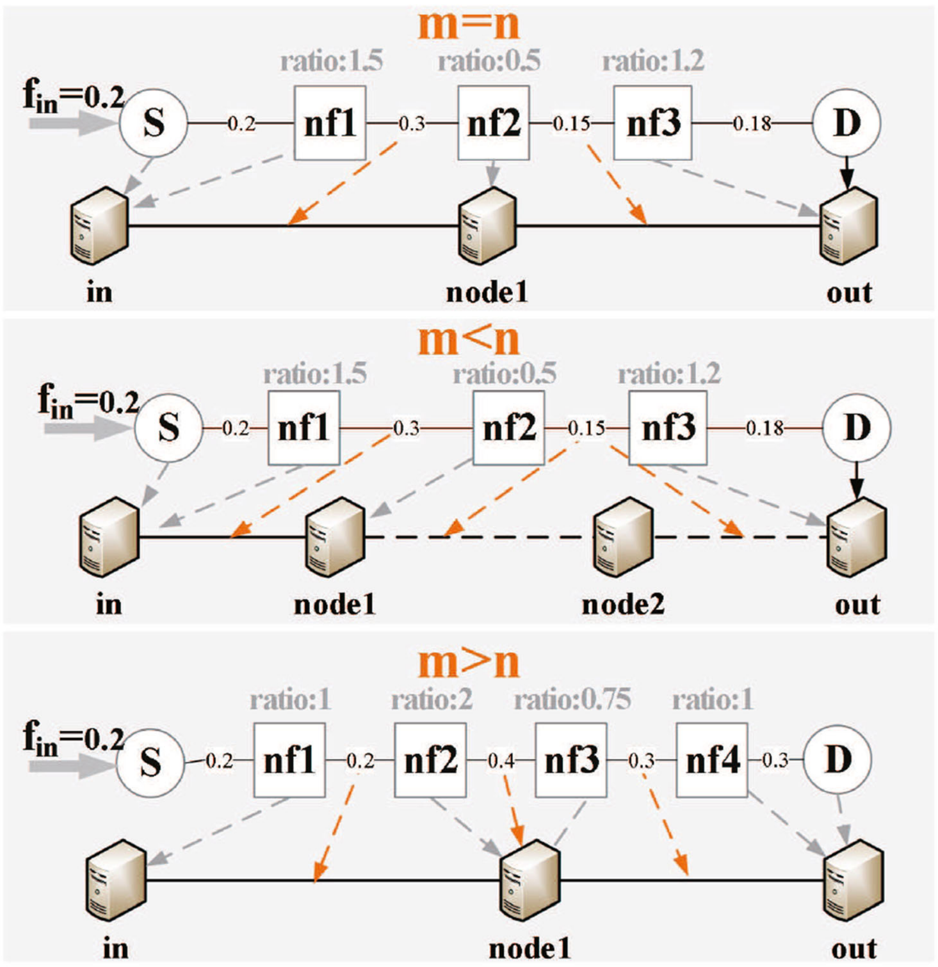


Fig. 8 Three cases of the LBSP algorithm

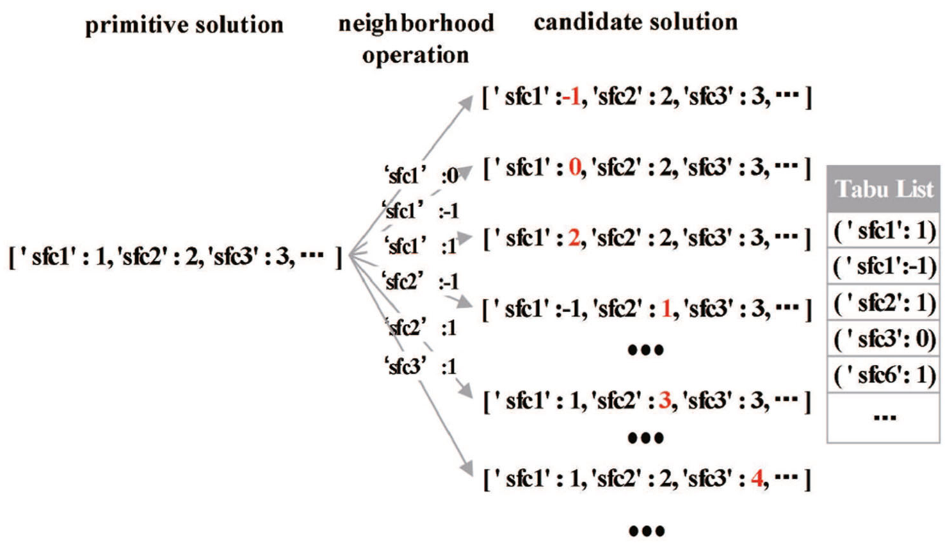

primitive solution
neighborhood operation
candidate solution
['sfc1' : 1,'sfc2' : 2,'sfc3' : 3, ···]
'sfc1' :0
'sfc1' :-1
'sfc1' :1
'sfc2' :-1
'sfc2' :1
'sfc3' :1
['sfc1' :-1,'sfc2' : 2,'sfc3' : 3, ···]
['sfc1' : 0,'sfc2' : 2,'sfc3' : 3, ···]
['sfc1' : 2,'sfc2' : 2,'sfc3' : 3, ···]
['sfc1' :-1,'sfc2' : 1,'sfc3' : 3, ···]
['sfc1' : 1,'sfc2' : 3,'sfc3' : 3, ···]
['sfc1' : 1,'sfc2' : 2,'sfc3' : 4, ···]
Tabu List
('sfc1': 1)
('sfc1':-1)
('sfc2': 1)
('sfc3': 0)
('sfc6': 1)
···

Fig. 9 Neighborhood operation and Tabu List of TS algorithm

Algorithm 1 Tabu Search algorithm

```
Input: solution space: $\Omega$, customer service: SFCs
Output: optimal solution: s, best fitness: g_fit
    Require: g_fit←0, tabu_list←∅, Initialize solution s ←ISG-LBSP (SFCs,$\Omega$)
 1: while not stop do
 2:     S←Generating_candidate_set(s)
 3:     F←Calculated_fitness(S)
 4:     f_best, s_best, n←get max(F) and correspond candidate solution, neighbor
 5:     if n not in tabu_list then
 6:         g_fit←f_best, s←s_best, update n to tabu_list
 7:     else
 8:         if f_best>g_best then
 9:             g_fit←f_best, s←s_best, update n
10:         else
11:             while n in tabu_list do
12:                 f_best, s_best, n←get next max(F) and correspond candidate solution, neighbor
13:             end while
14:             g_fit←f_best, s←s_best, update n to tabu_list
15:         end if
16:     end if
17: end while
18: return s, g_fit ;
```

4.5.5 Simulated annealing algorithm

The idea of simulated annealing algorithm (SA) is based on the principle of solid annealing. The algorithm starts from a higher temperature , and the solution tends to be stable with the temperature decreasing. SA algorithm will receive the bad solution with a certain probability to avoid falling into the local optimum. Considering the discontinuity of SFC embedding fitness, the algorithm improves the Metropolis criterion of the probability of receiving new solutions, and limits the minimum value of energy difference $E(X_{new}) - E(X_{old})$ between the new solution and the old solution to 1, as shown in (3)

$$p = \begin{cases} 1 \quad if\, E(x_{new}) - E(x_{old}) \leq 0 \\ \exp\left(-\dfrac{1}{T}\right) if\, 0 \ < E(x_{new}) - E(x_{old}) \leq 1 \\ \exp\left(-\dfrac{E(x_{new}) - E(x_{old})}{T}\right) if\, E(x_{new}) - E(x_{old}) \ > 1 \end{cases} \tag{3}$$

The first line indicates that when the new solution is better than the old solution, the new solution is received. The second line indicates that the new solution is not better than the old solution and the fitness difference is less than 1. Accept the new solution with probability $exp(-\frac{1}{T})$, otherwise receive the new solution with probability $exp(-\frac{E(x_{new})-E(x_{old})}{T})$.

The initial solution of SA is generated by LBSP algorithm, which is the same as TS algorithm. When candidate solutions are generated, half of SFCs are randomly selected to perform neighborhood operations to avoid loops. Algorithm 2 describes the main steps of SA. Lines 2-4 describe the candidate solution generation and calculation fitness. Lines 5-15 describe the improved Metropolis criterion. When the temperature T_k is less than $T_{\min}$, the search will exit.

Algorithm 2 Simulated Annealing algorithm

```
Input: solution space: $\Omega$, customer service: SFCs, initial
    temperature: T0
Output: optimal solution: s, best fitness: g_fit
    Require: g_fit←0, Tk←T0, k=0, Initialize solution s
    ←ISG-LBSP (SFCs,$\Omega$)
 1: while Tk>Tmin do
 2:     S←Generating_candidate_set(s)
 3:     F←Calculated_fitness(S)
 4:     f_best, s_best←get max(F) and correspond candidate
    solution
 5:     if f_best - g_fit >0 then
 6:         g_fit←f_best, s←s_best,
 7:     else if f_best-g_fit>-1 then
 8:         if exp(-1/T)>rand() then
 9:             g_fit←f_best, s←s_best,
10:         end if
11:     else
12:         if exp(-(f_best- g_fit)/T)>rand() then
13:             g_fit←f_best, s←s_best
14:         end if
15:     end if
16:     k++
17:     Tk←T0/(1+k)
18: end while
19: return s, g_fit ;
```

4.5.6 Particle swarm optimization algorithm

The particle swarm optimization algorithm (PSO) originates from the predation behavior of birds. By designing particles to simulate the behavior of birds, each particle searches separately in the search space, and adjusts its position and speed according to the individual optimal and group optimal to seek the optimal solution. In the PSO algorithm, each particle contains a random initial embedding scheme generated by all SFCS with KSP-SS algorithm. The combination of these schemes constitutes the unique position of the particles, and the update operation of the particles can be realized in the discrete digital domain. Algorithm 3 describes the process of PSO, the first line calculates the fitness of each particle, and the second line updates the global and local optimal fitness g_fit, p_fit. Line 3 updates the global and local optimal embedding scheme g_best, p_best. Line five updates the speed of particles, C1 and C2 are used to control the tracking speed of particles to the individual optimal and group optimal solution. Line six is to update the particle position.

Algorithm 3 Particle Swarm Optimization algorithm

```
Input: solution space: $\Omega$, customer service: SFCs, tracking
    factor: C1, C2
Output: optimal deployment: g_best, best fitness: g_fit
    Require: g_fit←0, g_best←p_fit←Ø, p_best←[Ø], Ini-
    tializing particle swarm X ←random (SFCs, $\Omega$)
 1: F←Calculated_fitness(X)
 2: g_fit←max(F), p_fit←F
 3: g_best←F.index(g_fit), p_best←X
 4: while not stop do
 5:     V←C1*rand()*(p_best- X )+C2*rand()*(g_best- X))
 6:     X=round(X+V)
 7: end while
 8: return g_best, g_fit;
```

5 Evaluation

All the experiments are carried out on a Hasee Z7-KP7GZ Laptop with an Intel Core i7-8450H CPU, 16 GB RAM. The machine runs a 64-bit Windows 10 operating system with a Python 3.6.1 environment.

5.1 Simulation design

To verify the effectiveness of the simulation platform, we tested the performance of key operations such as instantiation VNF, NF deployment and virtual link deployment of the simulation platform, and compared the performance with the VNE simulation platform based on Java language. After that, this section simulates the static deployment of SFC. The CERNNET2 network topology included in the simulation platform is selected as the substrate network. The CERNNET2 network consists of 21 nodes and 23 links. The network node resources and link delay are uniformly distributed. At the same time, user service includes 46 SFCs, which aim at three scenarios of 5G mMTC, uRLLC and eMBB respectively. The delay, bandwidth and the number of NFs of SFC are uniformly distributed. The substrate network and SFC parameter settings are shown in Table 2.

We don’t show the unit of parameters, only use one value to represent resource and coefficient. The simulation goal is to deploy the 46 service function chains on the underlying network and get the most traffic in the least time. We take the above problem as an open challenge, and test the system performance with the designed TS, SA, PSO, and LBSP algorithms. The detailed process of the algorithm and the source code of the simulation platform is in the https://github.com/SFCSim/SFCSim.

5.2 Simulation result

Figure 10 shows the performance comparison between SFCSim and VNE simulation platform Alevin. Note that both NFV-RA and VNE include two key stages of virtual

Table 2 SIMULATION PARAMETER

Parameter		value
Node resources		U(10, 30)
Link resources		10
Link delay		U(0.5, 1.5)
Access node		random
NF Resource coefficient		U(1, 2)
Traffic demand	mMTC	U(0.1, 0.5)
	uRLLC	U(1, 2)
	eMBB	U(3, 4)
Delay limit	mMTC	U(5, 10)
	uRLLC	U(2, 4)
	eMBB	U(5, 10)
SFC length	mMTC	U(3, 5)
	uRLLC	U(1, 2)
	eMBB	U(3, 5)

U(a,b) is the uniform distribution from a to b. The source dataset and detailed parameters are set in the https://github.com/SFCSim/SFCSim

node and virtual link deployment, but the complexity of NFV-RA is much greater than VNE. Fig. 10 (a) and (b) respectively show the time-consuming of virtual node deployment and virtual link deployment for different times in the two platforms. It can be seen that SFCSim is more than twice faster as Aevin in virtual node deployment. For virtual link deployment, SFCSim is faster than Alevin when the number of operations is small, and the time consumption increases rapidly and exceeds Alevin with the increase of the number of operations. The above rapid growth phenomenon has a great relationship with the underlying programming languages of Alevin and SFCSim. Alevin is a simulation software based on Java language, and the compiler will optimize the code instructions when performing similar operations many times. While SFCSim is based on Python language, the optimization ratio is less, but the development efficiency of Python is higher. The experimental results show that SFCSim can get almost the same performance as the VNE simulation platform Alevin when simulating the NFV-RA problem with greater complexity.

Fig. 11 shows the time-consuming of six key operations in NFV-RA: Instantiating VNF, Creating SFC, Deploying virtual node, Deploying virtual link, Scaling VNF and Node Resource Reservation on the SFCSim platform. Among them, the virtual node deployment and virtual link deployment operations are compared with similar operations of the VNE simulation platform Alevin. It can be seen from the figure that these key operations are completed within 30 microseconds, and the most important deployment operation is equivalent to Alevin, while the complexity of NFV-RA is much greater than that of VNE, which can prove the effectiveness of SFCSim.

Table 3 shows the deployment traffic and running time of the LBSP algorithm and three heuristic search algorithms. The LBSP runs the fastest because it has no search strategy, while the other three heuristic algorithms run longer, but can get a better search result and deploy more

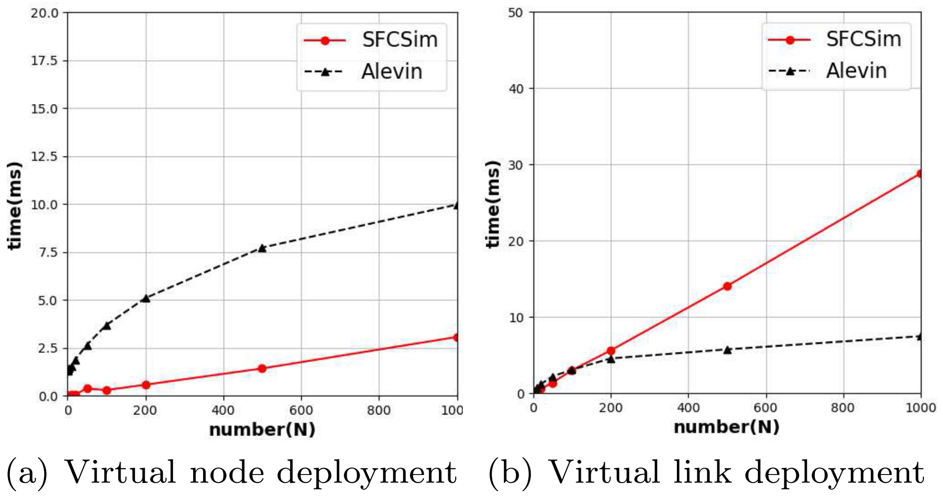


Fig. 10 Comparison of key operational performance between SFCSim and Alevin

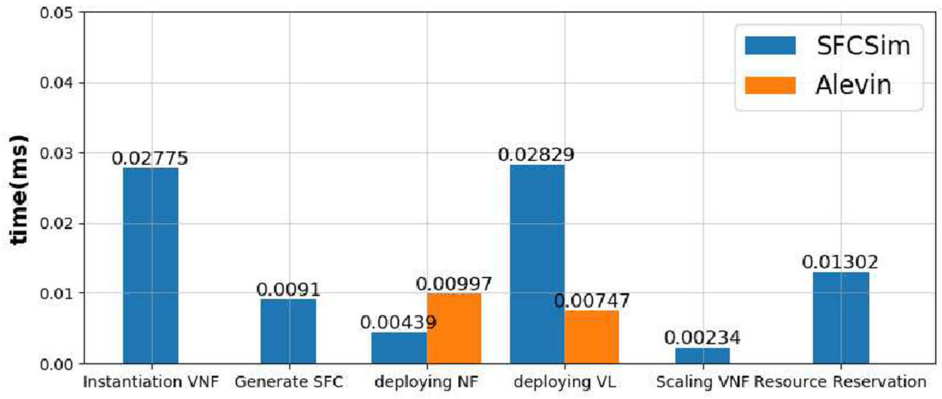


Fig. 11 Execution time of key NFV-RA operations on the SFCSim

Table 3 RESULTS OF DEPLOYMENT

Algorithm	Deployed traffic	time (s)
LBSP	153.55	0.016
PSO	160.84	25.36
TS	165.5	26.76
SA	165.5	18.61

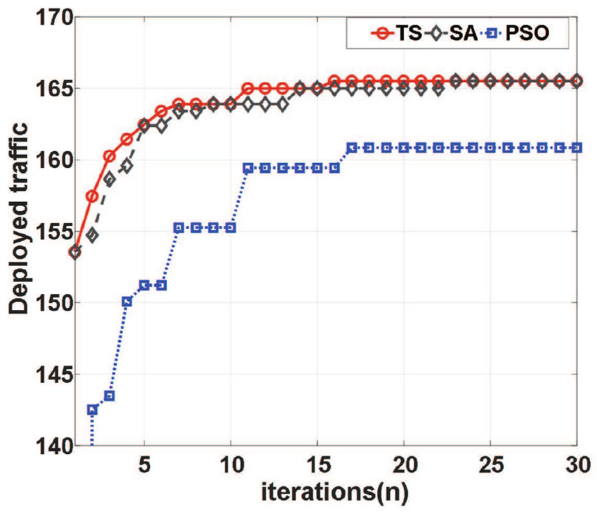


Fig. 12 The iterative results of three heuristic algorithms

traffic. Fig. 12 shows the iterative process of the three heuristic algorithms. The initial solution of the TS algorithm and SA algorithm is based on the LBSP algorithm, the convergence is fast and the deployment traffic is large. The initial solution of PSO is generated randomly, so the iteration speed is slow, but a better solution can be obtained in the end.

The CPU and memory utilization of the above algorithms in the platform are shown in Fig. 13(a) and Fig. 13(b). Since the LBSP algorithm has a short running time and cannot be measured, the other three heuristic algorithms can almost maximize the utilization of a single CPU core. In terms of memory footprint, the memory occupation of the four algorithms running on the platform is less than 90MB, and the simulation of the algorithm can be completed well, which proves the excellent performance of the platform for algorithm simulation.

6 Conclusion

In this paper, a general NFV-RA simulation platform SFCSim is implemented. The simulation platform provides a consistent underlying data structure and a common interface for NFV-RA problems. Meanwhile, the network infrastructure and user services are managed and orchestrated automatically based on the discrete-time event scheduling method to realize the dynamic adjustment of resources and state. Through SFCSim, the scheme designer can quickly complete the simulation and evaluation of the design, and compare it with other schemes conveniently. In the evaluation of the simulation platform, a load-balancing shortest path and three heuristic algorithms are designed, and the performance of the algorithms are simulated on the CERNNET2 network topology to verify the effectiveness of the platform. These algorithms can be used as benchmark algorithms to compare with others.

In future development, we will continue to improve the SFCSim simulation platform. Based on the advantages of the Python language, we will combine the platform with OpenStack, and expand SFCSim based on the Tacker subproject to realize the combination of simulation platform and experimental platform.

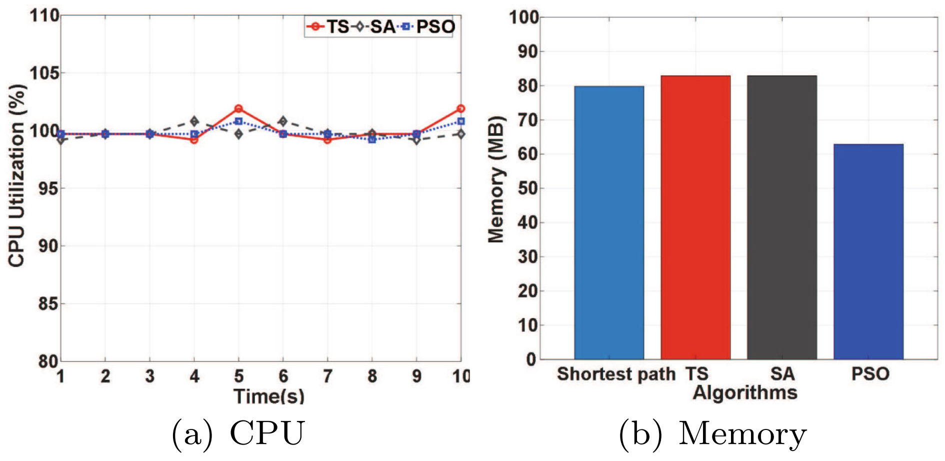


Fig. 13 CPU and Memory Utilization of the algorithms

Acknowledgements This work is supported by the National Natural Science Foundation of China (Nos.61821001, 62090015). The author is very grateful to the scholar of the Institute of Smart Wireless Mobile Information Technology (ISWIT) who provided the design for three heuristic algorithms in SFCSim construction. Thanks to anonymous reviewer for their suggestions to improve the quality of this article.

Author contributions Xu Lingyi performed the experiment, data analyses and wrote the manuscript, Hu Hefei contributed to the conception of the study, Liu yuanan contributed significantly to analysis and manuscript preparation.

Data availability The datasets generated during the current study are available in the https://github.com/SFCSim/SFCSim repository

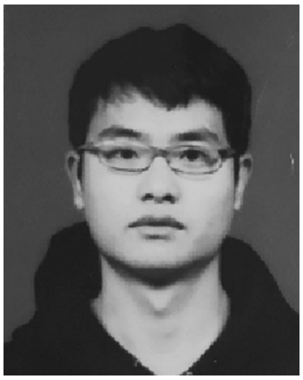
Declarations

Ethical statement No conflict of interest exists in the submission of this manuscript, and the manuscript is approved by all authors for publication. This article does not contain any studies with human participants or animals performed by any of the authors. I would like to declare on behalf of my co-authors that the work described has not been submitted elsewhere for publication, in whole or in part.

References

1. ETSI, Network Functions Virtualization (NFV); Architectural Framework, ETSI GS NFV 002 (V1.2.1) - (12-2014)
2. Halpern, J., Pignataro, C.: Service function chaining (sfc) architecture[M]//RFC 7665 (2015)
3. Herrera, J.G., Botero, J.F.: Resource allocation in NFV: a comprehensive survey. IEEE Trans. Network Serv. Manag. **13**(3), 518–532 (2016)
4. Li, H., Wang, L., Wen, X., et al.: MSV: An algorithm for coordinated resource allocation in network function virtualization[J]. IEEE Access **6**, 76876–76888 (2018)
5. Alameddine, H.A., Sebbah, S., Assi, C.: On the interplay between network function mapping and scheduling in VNF-based networks: a column generation ap-proach[J]. IEEE Trans. Network Serv. Manag. **14**(4), 860–874 (2017)
6. Kuo, T.W., Liou, B.H., Lin, K.C.J., et al.: Deploying chains of virtual network functions: on the relation between link and server usage. IEEE/ACM Trans. Netw. **26**(4), 1562–1576 (2018)
7. Wang, L., Lu, Z., Wen, X., et al.: Joint optimization of service function chaining and resource allocation in network function virtualization. IEEE Access **4**, 8084–8094 (2016)
8. Beck, M.T., Botero, J.F.: Scalable and coordinated allocation of service function chains. Comput. Commun. **102**, 78–88 (2017)
9. NS-2.: The network simulator—-ns-2, http://nsnam.isi.edu/nsnam/index.php/Main_Page. Accessed 23 Aug 2011 (2011)
10. NS-3.: The ns-3 project. http://www.nsnam.org/ (2011)
11. Beck, M.T., Linnhoff-Popien, C., Fischer, A., et al.: A simulation framework for virtual network embedding algorithms. In: 2014 16th International Telecommunications Network Strategy and Planning Symposium (Networks). IEEE, pp. 1–6 (2014)
12. Yu, M., Yi, Y., Rexford, J., Chiang, M.: Rethinking virtual network embedding: Substrate support for path splitting and migration. ACMSIGCOMM CCR **38**(2), 17–29 (2008)
13. Cao, H., Zhang, Y., Yang, L.: Dynamic embedding and scheduling of virtual network service for future networks. In: 2019 IEEE International Conference on Communications Workshops (ICC Workshops). IEEE, pp. 1–6 (2019)
14. Liu, J., Lu, W., Zhou, F., et al.: On dynamic service function chain deployment and readjustment. IEEE Trans. Netw. Serv. Manag. **14**(3), 543–553 (2017)
15. Eramo, V., Miucci, E., Ammar, M., et al.: An approach for service function chain routing and virtual function network instance migration in network function virtualization architectures. IEEE/ACM Trans. Netw. **25**(4), 2008–2025 (2017)
16. Hu, H., Yang, C., Xu, L., et al.: Path load adaptive migration for routing and bandwidth allocation in mobile-aware service function chain. Electronics **11**(1), 57 (2021)
17. Sun, G., Xu, Z., Yu, H., et al.: Low-latency and resource-efficient service function chaining orchestration in network function virtualization. IEEE Internet Things J. **7**(7), 5760–5772 (2019)
18. Pei, J., Hong, P., Pan, M., et al.: Optimal VNF placement via deep reinforcement learning in SDN/NFV-enabled networks. IEEE J. Sel. Areas Commun. **38**(2), 263–278 (2019)
19. Ma, Y., Liang, W., Li, J., et al.: Mobility-aware and delay-sensitive service provisioning in mobile edge-cloud networks. IEEE Trans. Mob. Comput. **21**(1), 196–210 (2020)
20. Liu, Y., Zhang, H., Guan, H., et al.: A method for adaptive resource adjustment of dynamic service function chain. IEEE Access **6**, 69988–70004 (2018)
21. OPNET.: OPNET modeler: Scalable network simulation.http://www.opnet.com/solutions/network_rd/modeler.html (2011)
22. Hagberg, A., Swart, P.S, Chult, D.: Exploring network structure, dynamics, and function using NetworkX. Los Alamos National Lab. (LANL), Los Alamos, NM (2008)

23. Chowdhury, N.M.M.K., Rahman, M.R., Boutaba, R.: Virtual network embedding with coordinated node and link mapping. In: IEEE INFOCOM. IEEE, pp. 783–791 (2009)
24. Michael Till Beck.: NFV-Orchestration. https://github.com/wshwb/NFV-Orchestration. Accessed 14 Dec 2016
25. Srcvirus.: Middlebox-placement. https://github.com/wshwb/middlebox-placement. Accessed 6 Jan 2018
26. Nguyen, T.M., Girard, A., Rosenberg, C., et al.: Routing via functions in virtual networks: the curse of choices. IEEE/ACM Trans. Netw. **27**(3), 1192–1205 (2019)
27. Mechtri, M., Ghribi, C., Zeghlache, D.: A scalable algorithm for the placement of service function chains. IEEE Trans. Netw. Serv. Manag. **13**(3), 533–546 (2016)
28. Chua, F.C., Ward, J., Zhang, Y., et al.: Stringer: balancing latency and resource usage in service function chain provisioning. IEEE Internet Comput. **20**(6), 22–31 (2016)
29. Li, H., Wang, L., Wen, X., et al.: Constructing service function chain test database: an optimal modeling approach for coordinated resource allocation. IEEE Access **6**, 17595–17605 (2017)
30. Leivadeas, A., Kesidis, G., Falkner, M., et al.: A graph partitioning game theoretical approach for the VNF service chaining problem. IEEE Trans. Network Serv. Manag. **14**(4), 890–903 (2017)
31. Jalalitabar, M., Guler, E., Zheng, D., et al.: Embedding dependence-aware service function chains. J. Opt. Commun. Netw. **10**(8), C64–C74 (2018)
32. Chen, X., Zhu, Z., Proietti, R., et al.: On incentive-driven VNF service chaining in inter-datacenter elastic optical networks: a hierarchical game-theoretic mechanism. IEEE Trans. Netw. Serv. Manag. **16**(1), 1–12 (2018)
33. Gupta, A., Habib, M.F., Mandal, U., et al.: On service-chaining strategies using virtual network functions in operator networks. Comput. Netw. **133**, 1–16 (2018)
34. Zhang, J., Wu, W., Lui, J.C.S.: On the theory of function placement and chaining for network function virtualization. In: Proceedings of the Eighteenth ACM International Symposium on Mobile Ad Hoc Networking and Computing, pp. 91–100 (2018)
35. Yi, B., Wang, X., Huang, M., et al.: A multi-criteria decision approach for minimizing the influence of VNF migration. Comput. Netw. **159**, 51–62 (2019)
36. Eramo, V., Ammar, M., Lavacca, F.G.: Migration energy aware reconfigurations of virtual network function instances in NFV architectures. IEEE Access **5**, 4927–4938 (2017)
37. Sun, G., Zhou, R., Sun, J., et al.: Energy-efficient provisioning for service function chains to support delay-sensitive applications in network function virtualization. IEEE Internet Things J. **7**(7), 6116–6131 (2020)
38. Heuristic Strategy of Service Function Chain Deployment Based on N-Base Continuous Digital Coding in Network Function Virtualization Environment
39. Xu, L., Hu, H., Liu, Y.: Heuristic strategy of service function chain deployment based on N-base continuous digital coding in network function virtualization environment. Electronics **11**(3), 331 (2022)
40. Glover, F., Laguna, M.: Tabu search. In: Handbook of combinatorial optimization, pp. 2093–2229. Springer, Boston, MA (1998)
41. Mathew, T.V.: Genetic algorithm. Report submitted at IIT Bombay (2012)
42. Kennedy, J., Eberhart, R.: Particle swarm optimization. In: Proceedings of ICNN'95-International Conference on Neural Networks. IEEE, vol 4, pp 1942–1948 (1995)



Lingyi Xu received the B.S. degree in electrical engineering from Beijing University of Posts and Telecommunications (BUPT), Beijing, China, in 2018. He is currently pursuing the Ph.D. degree at the School of Electronic Engineering, Beijing University of Posts and Telecommunications (BUPT), Beijing, China. His research interests include Bluetooth Low Energy technology in dense Internet of Things environments and service function chain construction in NFV/SDN environments.

Hefei Hu received the B.S. and Ph.D. degrees in Communication Engineering from Beijing University of Posts and Telecommunications in 2000 and 2005, respectively. He is now an associate professor at the School of Information and Communication Engineering at Beijing University of Posts and Telecommunications. His research interests include high - speed wireless access networks, Network Function Virtualization management and orchestration, SDN and spatial networks.

Yuanan Liu (M'93) received the B.E., M.Eng., and Ph.D. degrees in electrical engineering from the University of Electronic Science and Technology of China, Chengdu, China, in 1984, 1989 and 1992, respectively. In 1984, he joined the 26th Institute of the Electronic Ministry of China, to develop the inertia navigating system. In 1992, he began his first post -doctor position with the Electromagnetic Compatibility (EMC) Laboratory, Beijing University of Posts and Telecommunications (BUPT), Beijing, China. In 1995, he started his second post -doctor with the Broadband Mobile Laboratory, Department of System and Computer Engineering, Carleton University, Ottawa, ON, Canada. Since July 1997, he has been a Professor with the Wireless Communication Center, College of Telecommunication Engineering, BUPT, where he is involved in the development of next -generation cellular system, wireless local area networks (LANs), Bluetooth application for data transmission, EMC design strategies for high - speed digital systems, and electromagnetic interference (EMI) and electromagnetic susceptibility (EMS) measuring sites with low cost and high performance. He is interested in smart antennas for high - capacity mobile signal processing techniques in fading environments, EMC for high -speed digital system, ISI suppression, orthogonal frequency division multiplexing (OFDM), and multicarrier system design. Dr. Liu is a Senior Member of the Electronic Institute of China.