IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received October 26, 2018, accepted November 9, 2018, date of publication November 19, 2018, date of current version December 27, 2018.

Digital Object Identifier 10.1109/ACCESS.2018.2882255

MSV: An Algorithm for Coordinated Resource Allocation in Network Function Virtualization

HANG LI[1,2,3], LUHAN WANG[1,2,3], XIANGMING WEN[1,2,3], ZHAOMING LU[1,2,3], AND JINYAN LI[4]

[1]School of Information and Communication Engineering, Beijing University of Posts and Telecommunications, Beijing 100088, China
[2]Beijing Laboratory of Advanced Information Networks, Beijing University of Posts and Telecommunications, Beijing 100088, China
[3]Beijing Key Laboratory of Network System Architecture and Convergence, Beijing 100876, China
[4]China Telecom Technology Innovation Center, Beijing 102209, China

Corresponding author: Luhan Wang (wluhan@bupt.edu.cn)

This work was supported in part by the Fundamental Research Funds for the Central Universities under Project 2018RC06, in part by the 111 Project B17007, in part by the Beijing Municipal Science and Technology Commission Research under Project Z171100005217001, and in part by the National Science and Technology Major Project under Grant 2018ZX03001016.

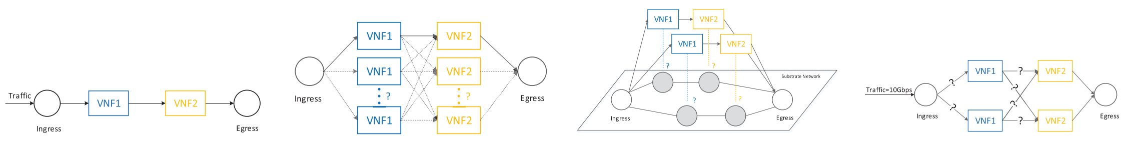
ABSTRACT The proposition of network function virtualization (NFV) aims to solve the difficulty and ossification in current network's management and service provision caused by ever-growing NFs with dedicated hardware. By decoupling the NFs from dedicated hardware to virtualized platform, NFV promises flexible deployment and management of service function chains (SFCs). However, an optimal resource allocation for requested SFC in NFV-based infrastructures should coordinately consider following three stages: virtual network functions (VNFs) chain composing, VNF forwarding graph embedding, and VNFs scheduling, which is a tough task as the decision of these three phases is mutually dependent. In this paper, staring from the challenges in solving coordinated NFV resource allocation (NFV-RA), we first formulate a typical three-stage coordinated NFV-RA model as a mixed integer programming (MIP) and, then, propose a heuristic solution called merge–split viterbi (MSV). MSV can automatically determine the appropriate number of VNF instances without given maximum number threshold, and it does not take the iterative deployment strategy, which is commonly used in current solutions. The main idea of MSV is to first find a global basic solution and, then, to further optimize the basic solution through some improvement procedures, and this makes it not be easily trapped in local optimality and avoid complex anti-local-optimal measures as well. Extensive experiments demonstrate that MSV can get solutions in global range with reasonable execution time and achieves total cost ratio within 115% compared to the MIP implement.

INDEX TERMS NFV, coordinated resource allocation, service function chain.

I. INTRODUCTION

Service function Chain (SFC) [1] is an ordered sequence of network functions (NFs) which should be traversed by a given traffic flow to compose a certain service. Traditionally, these network functions such as firewalls, proxies, Deep Packet Inspections (DPIs), are integrated in specialized hardware called middle-boxes [2], [3]. However, middle-boxes are generally expensive, vendor specific and location fixed which makes flexible and dynamic resource management challenging. To this end, a novel network architecture called network function virtualization (NFV) [4], [5] is proposed to decouple network functions from dedicated hardware. In NFV, NFs are placed within virtual machines (VMs) on commodity servers in the form of software appliance which are referred to as virtual network functions (VNFs). In this way, NFV enables flexible SFC deployment based on geography or customer sets since VNFs can be deployed on any servers in the network.

Despite the much flexibility provided by NFV, a new challenging problem is how to achieve fast, scalable and flexible composition of SFC and resource allocation for VNFs which is called NFV Resource Allocation (NFV-RA) [6]. NFV-RA mainly includes three stages [6], [7], VNFs Chain Composing (VNFs-CC), VNF Forwarding Graph Embedding (VNF-FGE) and VNFs Scheduling (VNFs-SCH). The VNFs-CC is to decide the order of VNFs and the instance amount of each VNF. The VNF-FGE mainly considers placing VNF instances to appropriate locations to optimize certain objectives. And the VNFs-SCH mainly deals with the problem that how to schedule traffic among VNF instances. Fig. 1



(a) Service function chain (b) VNFs-CC (c) VNF-FGE (d) VNFs-SCH

FIGURE 1. An example of three stages in resource allocation. (a) Service function chain. (b) VNFs-CC. (c) VNF-FGE. (d) VNFs-SCH.

shows an example of above three stages. Existing researches about NFV-RA mainly focus on studying one [8], [9] or two stages [10], [11] (including two-stage coordination [12], [13]) of NFV-RA (readers could refer to [6] for more details), however, these three stages are actually mutually dependent. An optimal NFV-RA should consider above three stages coordinately while the related resource and location constraints are satisfied. Therefore, a recent trend in NFV-RA is to study the coordination of NFV-RA's three stages and in this work we call it three-stage coordinated NFV-RA (TSC NFV-RA) problem. Although there are a considerable number of researches and investigations on NFV-RA in recent years, most of works focus on one or two stages and use other stages as input parameters or known conditions. Only a few works study the TSC NFV-RA problem and propose the corresponding solutions. However, there are still some shortcomings in these solutions. Thus, further study on TSC NFV-RA is needed.

To the best of our knowledge, following works study the TSC NFV-RA problem. In [7], a heuristic based algorithm JoraNFV is proposed to solve the VNFs-CC, VNF-FGE and VNFs-SCH in a coordinated way. In [14], the TSC NFV-RA problem is formulated as a joint service-function deployment and traffic scheduling (SUPER) problem and an approximation algorithm based on the Markov approximation technique is proposed. Jang *et al.* [15] jointly optimize the three stages of NFV-RA by proposing a polynomial time algorithm based on linear relaxation and rounding. Ghaznavi *et al.* [16] formulate the TSC NFV-RA problem as a distributed service function chaining (DSFC) problem then develop a local search heuristic called Kariz to solve it. Considering the NP-hardness and high complexity of TSC NFV-RA problem, almost all related works propose heuristic or approximation algorithms to reduce the time complexity in orchestrating resource allocation. However, there are still some challenges in designing the solution of TSC NFV-RA problem as follows.

Challenge in determining the number of VNF instances. In TSC NFV-RA problem, each type of VNF can have multiple instances to be deployed and the designed algorithm should automatically determine the appropriate amount of instances according to actual conditions of network during the deployment. Such uncertainty of instance number increases the difficulty of designing algorithms thus some works [7], [14] set the maximum number of VNF instances. In other words, the number of instances of each VNF cannot exceed a preset threshold during the deployment. However, finding the appropriate threshold is difficult. Some good solutions may be missed when the threshold is set to be small while big threshold will lead to the waste of the execution time and space.

Challenge in iterative deployment strategy. Considering the characteristic of SFC, most of heuristic solutions [7], [16]–[18] of coordinated NFV-RA take the iterative deployment strategy: starting from the first VNF, VNFs in SFC are deployed orderly and all the related NFV-RA stages are coordinately solved during each VNF's deployment process. Here note that such strategy has the risk of being trapped in a local optimal solution thus above heuristic solutions have taken some extra measures to enlarge the search space. However, designing such anti-local-optimal measures is difficult especially in balancing the complexity and the performance. Some designed measures can effectively avoid the local optimality but they may have high complexity in both time and space. Conversely, some measures have low complexity but their performance may be unsatisfactory.

Starting from above two challenges, in this work, we first formulate a typical TSC NFV-RA model then propose a heuristic solution called Merge-Split Viterbi (MSV) to solve it. For each VNF of SFC, MSV can determine the appropriate number of VNF instances without given maximum number threshold. Besides, MSV does not take the iterative deployment strategy thus it avoids complex anti-local-optimal measures and realizes relatively low time complexity. The main idea of MSV is to first find a global basic solution then further to optimize this solution through some improvement procedures.

This paper makes the following contributions specifically: *i*) a typical TSC NFV-RA model is formulated as a mixed integer programming (MIP) which is incorporated to account for VNF deployment cost, bandwidth cost and delay cost. *ii*) a heuristic solution MSV is proposed to solve above TSC NFV-RA model. MSV does not require predefined maximum instance number threshold and does not take the iterative deployment strategy which makes it have relatively low time complexity. To the best of our knowledge, MSV is the first heuristic solution which does not take the iterative deployment strategy in solving the TSC NFV-RA problems. *iii*) extensive simulations are performed to evaluate MSV from cost performance and execution time two perspectives. The experimental result shows that MSV achieves competitive cost performance compared to MIP implement and can

TABLE 1. Notations used in system model.

Network Topology	
$G=(N,E)$	Graph G with nodes set N and links set E
N_{fw}	Set of forwarding nodes
N_{sv}	Set of service nodes
$a_r \in A_{res}$	Physical resource type a_r $(r=1,2,...,R)$
$c_m^{a_r}$	Resource capacity for $a_r \in A_{res}$ of node $m \in N_{sv}$
$e_{m,n} \in E$	Link between node $m \in N$ and $n \in N$
$d_{tm}(e_{m,n})$	Transmission delay of link $e_{m,n} \in E$
Service Function Chain (SFC)	
S	Service function chain S
v_x	VNF v_x $(x=1,2,...,w_S)$
m_{in}	Traffic ingress node
m_{out}	Traffic egress node
w_S	Number of VNFs in SFC S
l_S	Length of SFC S
u_x	Component u_x $(x=1,2,...,l_S)$
$u_{x,i}$	The i-th instance of u_x
$p_{u_x}^{a_r}$	Resource demand coefficient of a u_x instance for $a_r \in A_{res}$
$d_{u_x}^{a_r}$	Resource demand of a u_x instance for $a_r \in A_{res}$
η_{u_x}	Data rate scaling ratio of u_x
Homogeneous Link	
H	Set of homogeneous links
$h_x \in H$	Homogeneous link h_x $(x=1,2,...,l_S-1)$
f_S	The request traffic of SFC S
f_{h_x}	The request traffic of h_x
Decision Variable	
σ_{u_x}	The number of u_x instances
$\varsigma_{u_{x,i}}^m \in \{0,1\}$	$\varsigma_{u_{x,i}}^m=1$ if $u_{x,i}$ is deployed on node $m \in N_{sv}$
$\tau_{h_x,i,j}^{e_{m,n}}$	Scheduled data rate from $u_{x,i}$ to $u_{x,j}$ on edge $e_{m,n} \in E$
Cost Factor	
α_{u_x}	Deployment cost of a u_x instance
β	Cost coefficient of transmitting one unit traffic
δ	Delay penalty cost coefficient

get the solution in global range with reasonable execution time. The rest of the paper is organized as follows: The system model is formulated in section II. Our solution is proposed in section III and the evaluation of solution is presented in section IV. Lastly, a short conclusion of the work is given in section V.

II. SYSTEM MODEL

In this section, we introduce the mathematical system model and related definitions. Table 1 shows the notations used in model.

A. NETWORK TOPOLOGY

The substrate network topology is modeled as a graph $G = (N, E)$ where N and E denote the set of nodes and links respectively. Set N consists of the set N_{fw} of forwarding nodes and the set N_{sv} of service nodes set, i.e., $N = N_{fw} \cup N_{sv}$. Forwarding nodes only forward traffic flows to other forwarding nodes or service nodes while service nodes can not only forward flows but also carry the VNFs. Each service node is considered as a commodity server hosting multiple VMs and is endowed a set of available physical resources (e.g., CPU, memory, storage, etc.) denoted by $A_{res} = \{a_r | r = 1, 2, \ldots, R\}$. Meanwhile, we use $c_m^{a_r}$ to represent the resource capacity for type $a_r \in A_{res}$ of node $m \in N_{sv}$. $e_{m,n} \in E$ is the link between node m and node n $(m, n \in N)$ and $d_{tm}(e_{m,n})$ denotes the transmission delay of link $e_{m,n} \in E$. Here note that, for the better expression of intra-node delay, we assume dummy link $e_{m,m} \in E$ $(m \in N)$ and use $d_{tm}(e_{m,m})$ to denote the intra-node delay of node $m \in N$.

B. SERVICE FUNCTION CHAIN

An SFC is denoted as $S = \{m_{in}, v_1, \ldots, v_{w_S}, m_{out}\}$ where $v_1, v_2, \ldots, v_{w_S}$ are required VNFs and $m_{in}, m_{out} \in N_{fw}$ are traffic ingress node and traffic egress node respectively. Traffic flow coming from m_{in} is processed by VNFs in SFC then forwarded to m_{out} finally. Here we assume that there are no duplicate VNFs in an SFC and use w_S and l_S to represent the numbers of VNFs in SFC S and the length of SFC S respectively, i.e., $l_S = w_S + 2$. Both VNFs and m_{in}, m_{out} are regarded as components of SFC and we use u_x to denote the x-th $(x = 1, 2, \ldots, l_S)$ component in SFC S. Thus, the u_x satisfies $u_1 = m_{in}$, $u_{l_S} = m_{out}$ and $u_x = v_{x-1}$ $(x = 2, 3 \ldots, l_S - 1)$. Considering that each type of VNF can be deployed multiple times, we use $u_{x,i}$ to represent the i-th instance of u_x.

As mentioned before, VNFs are deployed in VMs running on the servers, so each VNF would require a certain amount of resources which are often relevant with the traffic volume passing through it. In this work, to reduce the complexity, we assume it is a linear relationship between the traffic volume and required resources and use $p_{u_x}^{a_r}$ to represent the resource demand coefficient of a u_x instance for resource type $a_r \in A_{res}$. So when f denotes the traffic volume handled by a u_x instance, the required resource type of $a_r \in A_{res}$ is $d_{u_x}^{a_r} = p_{u_x}^{a_r} f$ (note that $\forall a_r \in A_{res} : p_{u_1}^{a_r} = p_{u_{l_S}}^{a_r} = 0$).

Note that some VNFs can modify the traversing traffic volume [19] such as a video transcoder can change the encoding of the video which may result in the change of traffic volume. So we define the data rate scaling ratio of u_x as $\eta_{u_x} = \frac{t_{out}}{t_{in}}$, where t_{in} and t_{out} represent input and output traffic volume respectively (note that $\eta_{u_1} = \eta_{u_{l_S}} = 1$).

C. HOMOGENEOUS LINK

For the better description of later mathematical expressions, we define the concept of homogeneous link [20]. In an SFC, a link consisting of two adjacent components is called a homogeneous link. For example, in Fig. 2, the SFC contains four different homogeneous links. Here we use $H = \{h_x | x = 1, 2, \ldots, l_S - 1\}$ to represent the homogeneous link set of SFC S and call the traffic on h_x as "traffic with type h_x". We set f_S is the request traffic volume of SFC S and use f_{h_x} to denote the request traffic volume on homogeneous link h_x (i.e., $f_{h_1} = f_S$, $f_{h_2} = \eta_{u_2} f_S, \ldots, f_{h_{l_S-1}} = \eta_{u_{l_S-1}} f_{h_{l_S-2}}$).

D. DECISION VARIABLES

In this model, we introduce three kinds of decision variables: instance number decision variable $\sigma_{u_x} \in \sigma$, location

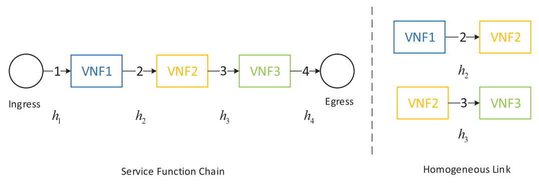


FIGURE 2. **Homogeneous link.**

decision variable $\varsigma^{m}_{u_{x,i}} \in \varsigma$ and traffic volume decision variable $\tau^{e_{m,n}}_{h_x,i,j} \in \tau$. Variable σ_{u_x} denotes the number of u_x instances. Variable $\varsigma^{m}_{u_{x,i}}$ is a binary value, which represents whether $u_{x,i}$ is deployed on node $m \in N$.

$$\varsigma^{m}_{u_{x,i}} = \begin{cases} 1 & \text{if } u_{x,i} \text{ is deployed on node } m, \\ 0 & \text{otherwise.} \end{cases} \tag{1}$$

Variable $\tau^{e_{m,n}}_{h_x,i,j}$ represents the traffic volume scheduled from $u_{x,i}$ to $u_{x+1,j}$ $(x = 1, 2, \ldots, l_S - 1)$ on edge $e_{m,n} \in E$. Here we use $X = (\sigma, \varsigma, \tau)$ to represent a solution of model.

E. CONSTRAINTS

1) BASIC CONSTRAINT

Eq. 2 ensures that each u_x must be deployed at least once while Eq. 3 ensures the relationship between σ_{u_x} and $\varsigma^{m}_{u_{x,i}}$. Eq. 4 makes ensure that each $u_{x,i}, i \in \{1, 2, \ldots, \sigma_{u_x}\}$ can be deployed only once.

$$\forall u_x \in S : \sigma_{u_x} \geq 1 \tag{2}$$

$$\forall u_x \in S : \sum_{m \in N} \sum_{i=1}^{\sigma_{u_x}} \varsigma^{m}_{u_{x,i}} = \sigma_{u_x} \tag{3}$$

$$\forall u_x \in S, \quad i \in \{1, 2, \ldots, \sigma_{u_x}\} : \sum_{m \in N} \varsigma^{m}_{u_{x,i}} = 1 \tag{4}$$

2) LOCATION CONSTRAINT

Eq. 5 ensures that u_1 and u_{l_S} are deployed only on $m_{in} \in N_{fw}$ and $m_{out} \in N_{fw}$ respectively and can be placed only once.

$$\begin{aligned} \sigma_{u_1} &= 1, \quad \varsigma^{m_{in}}_{u_{1,1}} = 1 \\ \sigma_{u_{l_S}} &= 1, \quad \varsigma^{m_{out}}_{u_{l_S,1}} = 1 \end{aligned} \tag{5}$$

3) NODE CAPACITY CONSTRAINT

Eq. 6 guarantees that the resource capacities of nodes are not violated.

$$\forall a_r \in A_{res} : \forall m \in N : \forall e_{n,m} \in E : \\ \sum_{x=2}^{l_S-1} \sum_{i=1}^{\sigma_{u_x}} \sum_{j=1}^{\sigma_{u_{x-1}}} p^{a_r}_{u_x} \varsigma^{m}_{u_{x,i}} \tau^{e_{n,m}}_{h_{x-1},j,i} \leq c^{a_r}_{m} \tag{6}$$

4) TRAFFIC VOLUME DEMAND CONSTRAINT

Eq. 7 ensures that the traffic volume request of each homogeneous link is satisfied.

$$\forall h_x \in H : \forall e_{m,n} \in E : \sum_{m \in N} \sum_{i=1}^{\sigma_{u_x}} \sum_{j=1}^{\sigma_{u_{x+1}}} \varsigma^{m}_{u_{x,i}} \tau^{e_{m,n}}_{h_x,i,j} = f_{h_x} \tag{7}$$

5) FLOW CONSERVATION CONSTRAINT

Eq. 8 makes sure that for each type of traffic, the inflow volume to an unprocessed node is equal to the outflow volume from this node.

$$\forall h_x \in H : \forall e_{m,n}, e_{n,o} \in E : \forall i \in \{1, 2, \ldots, \sigma_{u_x}\} : \\ \forall j \in \{1, 2, \ldots, \sigma_{u_{x+1}}\} : \sum_{m \in N} \tau^{e_{m,n}}_{h_x,i,j} = \sum_{o \in N} \tau^{e_{n,o}}_{h_x,i,j} \tag{8}$$

F. OPTIMIZATION OBJECTIVE

In this model, the final optimization objective is to minimize the total cost consisting of VNF deployment cost, delay cost and bandwidth cost.

1) VNF DEPLOYMENT COST

Owing to the license cost and the standby energy cost [21], deploying one VNF instance needs a certain amount of cost and here we take this as VNF deployment cost. Eq. 9 is the VNF deployment cost of SFC S where coefficient α_{u_x} is the deployment cost of one u_x instance (note that $\alpha_{u_1} = \alpha_{u_{l_S}} = 0$).

$$C_{deployment} = \sum_{u_x \in S} \alpha_{u_x} \sigma_{u_x} \tag{9}$$

2) BANDWIDTH COST

The network provider always expects to minimize the total bandwidth consumption in order to reduce the chance of congestion and related operational cost. So we consider the bandwidth consumption as one of the optimization objectives in the form of bandwidth cost. Eq. 10 is the bandwidth cost of SFC S where coefficient β is the cost coefficient of transmitting one unit traffic.

$$C_{bandwidth} = \sum_{h_x \in H} \sum_{e_{m,n} \in E} \sum_{i=1}^{\sigma_{u_x}} \sum_{j=1}^{\sigma_{u_{x+1}}} \beta \tau^{e_{m,n}}_{h_x,i,j} \tag{10}$$

3) DELAY COST

The increase of end-to-end delay may hurt the service's performance and the users' experience. So we also take the end-to-end delay as one of the optimization objectives. Considering that high end-to-end delay will lead to low service revenue, here we treat delay as a penalty to service revenue. Thus, the delay cost means a kind of penalty cost and we consider the end-to-end delay in the form of delay cost. Although the precise relationship between end-to-end delay and delay penalty cost should be modeled by investigating historic marketing statistics, we assume that it is a linear relationship for simplicity. Eq. 11 is the delay cost of SFC S where coefficient δ is the delay penalty cost coefficient.

$$C_{delay} = \sum_{h_x \in H} \sum_{e_{m,n} \in E} \sum_{i=1}^{\sigma_{u_x}} \sum_{j=1}^{\sigma_{u_{x+1}}} \delta d_{tm}(e_{m,n}) \tau^{e_{m,n}}_{h_x,i,j} \tag{11}$$

In addition, we call the sum cost of bandwidth cost and delay cost as link cost. It can be observed that the link cost is proportional to the traffic volume and here we call the link

Algorithm 1 MSV Algorithm

```
1: $(X, A_{res}^{left}) \leftarrow SplitChainViterbi(G, A_{res}, S, f_S, \lambda)$;
2: for $x = 2; x \le l_S - 1; x++$ do
3:     $Classify(G, A_{res}^{left}, f_S, \lambda, \varepsilon, u_x)$;
4: end for
5: for $x = 2; x \le l_S - 1; x++$ do
6:     if $u_x$ is not $BigVNF$ then
7:         $(X, A_{res}^{left}) \leftarrow Merge(G, A_{res}^{left}, S, f_S, \lambda, X, u_x)$;
8:     end if
9: end for
10: for $x = 2; x \le l_S - 1; x++$ do
11:     if $u_x$ is $BigVNF$ then
12:         $(X, A_{res}^{left}) \leftarrow Split(G, A_{res}^{left}, S, f_S, \lambda, X, u_x)$;
13:     end if
14: end for
15: $checkround \leftarrow 0$;
16: while $checkround < \Omega$ do
17:     for $x = 2; x \le l_S - 1; x++$ do
18:         if $u_x$ is $BigVNF$ then
19:             $(X, A_{res}^{left}) \leftarrow Update(G, A_{res}^{left}, S, f_S, X, u_x)$;
20:         end if
21:     end for
22:     $checkround \leftarrow checkround + 1$;
23: end while
24: End
```

Algorithm 2 Procedure *SplitChainViterbi*(.)

```
1: procedure $SplitChainViterbi(G, A_{res}, S, f_S, \lambda)$
2: $A_{res}^{left} \leftarrow A_{res}$;
3: for $k = 1; k \le \lambda; k++$ do
4:     $\forall x \in \{2, \ldots, l_S - 1\}, m_i \in N_{sv}$;
5:     $\xi_{path}(u_x, m_i) \leftarrow Null$;
6:     $\forall m_i \in N_{sv}$:
7:     if $c_{m_i}^{left} > p_{u_2}\frac{f_S}{\lambda}$ then
8:         $c_{cum}(u_2, m_i) \leftarrow LinkCost(m_{in}, m_i, f_S/\lambda)$;
9:         $\xi_{path}(u_2, m_i) \leftarrow m_i$;
10:     end if
11:     for $x = 3; x \le l_S - 1; x++$ do
12:         $\forall m_i \in N_{sv}, m_j \in N_{sv}$;
13:         if $c_{m_j}^{left} > p_{u_x}\frac{f_S}{\lambda}\prod_{i=1}^{x-1}\eta_{u_i}$ then
14:             $c_{cum}(u_x, m_j) \leftarrow \min\{c_{cum}(u_{x-1}, m_i) + LinkCost(m_i, m_j, \frac{f_S}{\lambda}\prod_{i=1}^{x-1}\eta_{u_i})\}$;
15:             $\xi_{path}(u_x, m_j) \leftarrow m_i$ which makes
16:             minimum $c_{cum}(u_x, m_j)$;
17:         end if
18:     end for
19:     $\forall m_i \in N_{sv}$:
20:     $c_{cum}^{final} \leftarrow \min\{c_{cum}(u_{l_S-1}, m_i) + LinkCost(m_i, m_{out}, \frac{f_S}{\lambda}\prod_{i=1}^{l_S-1}\eta_{u_i})\}$;
21:     $\xi_{path}^{final} \leftarrow m_i$ which makes
22:     minimum $c_{cum}^{final}$;
23:     Construct path $\rho = (\xi_{path}^{final}, \xi_{path}(u_{l_S-1}, \xi_{path}^{final}), \ldots)$;
24:     Get split chain deployment $\pi_k$
25:     according to path $reverse(\rho)$;
26:     $A_{res}^{left} \leftarrow refresh(A_{res}^{left})$;
27: end for
28: Get basic solution $X$ according to $\pi_1, \ldots, \pi_\lambda$;
29: return $X$ and $A_{res}^{left}$;
30: end procedure
```

cost under unit traffic volume as unit link cost. For each link $e_{m,n} \in E$, we can calculate its unit link cost and thus we can further obtain each path's unit link cost. We define the unit link cost between two nodes as the unit link cost of path who has the smallest unit link cost between them.

4) OBJECTIVE FUNCTION

So the TSC NFV-RA model can be formulated as following MIP model.

$$\begin{aligned} &\min\ (c_1 C_{deployment} + c_2 C_{bandwidth} + c_3 C_{delay}) \\ &subject\ to\ (2) - (8) \end{aligned} \tag{12}$$

Here c_1, c_2 and c_3 are relative importance coefficients.

III. MERGE-SPLIT VITERBI

In this section, we propose a heuristic solution called Merge-Split Viterbi (MSV) to address above TSC NFV-RA model. As mentioned before, MSV can automatically determine the appropriate instance number of each VNF without given the maximum threshold and MSV does not take the iterative deployment strategy. The main idea of MSV is first to find a global basic solution then further to optimize this solution through some improvement procedures. The basic steps of MSV are given in Alg. 1. MSV first gets the basic solution through *SplitChainViterbi*(.) procedure (line 1). Then all the VNFs in SFC are classified by *Classify*(.) procedure (lines 2-4). After that, MSV starts improvement procedures. MSV first processes all the VNFs which meet the trigger condition in SFC with *Merge*(.) (line 5-9). Then MSV continues to use *Split*(.) to process eligible VNFs in SFC (line 10-14). At last, MSV uses *Update*(.) procedure to further update the solution (line 15-23).

Note that MSV does not explicitly differentiate the types of resources. For each service node $m \in N_{sv}$, MSV calculates its abstract total resource capacity $c_m = \sum_{r=1}^{R} \chi_r c_m^{a_r}$ in advance. The χ_r $(r = 1, 2, \ldots R)$ can be set according to the importance of each resource. Similarly, MSV uses $p_{u_x} = \sum_{r=1}^{R} \chi_r p_{u_x}^{a_r}$ to represent the abstract total resource demand coefficient of a u_x instance and thus the required abstract total resource of a u_x instance is $d_{u_x} = p_{u_x} f$. In the following part of introducing MSV, the resource indicates above abstract total resource.

Still to introduce each main part of MSV in detail.

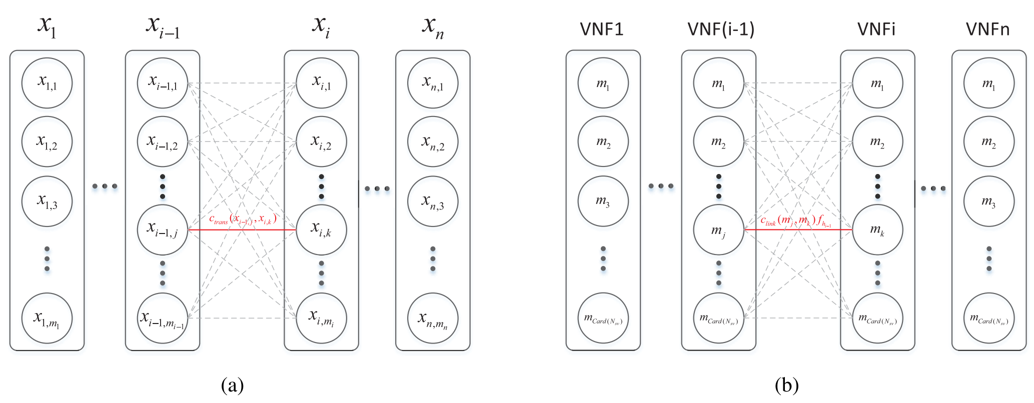

x_1
x_{i-1}
x_i
x_n
$c_{trans}(x_{i-1,j}, x_{i,k})$
VNF1
VNF(i-1)
VNFi
VNFn
$c_{link}(m_j, m_k) f_{h_{i-1}}$
(a)
(b)

FIGURE 3. Modeling with multi-stage graph.

A. FIND BASIC SOLUTION

The procedure *SplitChainViterbi*(.) shown in Alg. 2 is used for finding the global basic solution. The *SplitChainViterbi*(.) is based on Viterbi algorithm and here we first introduce the Viterbi algorithm and how it is applied in our solution.

Viterbi algorithm [22] is a dynamic programming algorithm for finding the most likely sequence of states from a set of observed states. The main process of Viterbi algorithm is as follow: Viterbi algorithm first constructs a multi-stage graph consisting of the states and their relationships as shown in Fig. 3(a). Each stage includes all possible states and there is a transition cost $c_{trans}(x_{i-1,j}, x_{i,k})$ between all pairs of states in successive stages. Then Viterbi algorithm computes per state cumulative cost $c_{cum}(x_{i,k})$ for all x_{i-1} in the previous stage to x_i's stage which is computed by $c_{cum}(x_{i,k}) = min\{c_{cum}(x_{i-1,j}) + c_{trans}(x_{i-1,j}, x_{i,k}) | j = 1, \ldots, m_{i-1}\}$ (here $i \geq 2$ and $\forall \varpi : c_{cum}(x_{1,\varpi}) = 0$). Above computation continuously proceeds in the increasing order of stage until finishing the final stage's computation. Finally, by tracing from the final stage back to the first stage, the most likely sequence of states is constructed by the path which accumulates the minimum cost. Viterbi algorithm can find the most likely sequence of states in $\Theta\left(\sum_{i=1}^{n-1} m_i m_{i+1}\right)$ time.

Inspired by [23], under our model, if we assume the request traffic volume f_S is fully small (or the resources of each service node are infinite), thus each VNF of SFC is only need to be deployed once (i.e., each VNF has only one instance), we can establish this reduction model's multi-stage graph and solve it with Viterbi algorithm as shown in Fig. 3(b). Here each stage is each type of VNF, each state is available location of VNF (i.e., each service node) and the transition cost between states is the link cost between nodes under the matched traffic volume f_{h_x}. Now finding the most likely sequence of states is to find the deployment path whose total cost is minimum and this path is also the optimal solution of this reduction model. Thus, Viterbi algorithm can solve this reduction model optimally (lines 4-25 in Alg. 2) in $\Theta\left(w_S N^2\right)$ time where $N = Card(N_{sv})$.

However, in our model, the request traffic volume f_S may be large and the resources are limited. So if we do not consider multiple instances, the resource demands for VNFs are large that none of service nodes can carry them. Thus, extra instances may have to be added to split the traffic flow to reduce the resource occupation of per VNF. But the Viterbi algorithm requests that each VNF should be deployed only once and thus above case may lead Viterbi algorithm to find an empty solution. Although Viterbi algorithm cannot solve our model directly, considering its global optimality, we manage to apply it to find the global basic solution.

As discussed above, under the resource constraint, the Viterbi algorithm is applicable only when the request traffic volume is small. So here we divide the original SFC into λ same split chains as shown in Fig. 4(b). Each split chain is identical with original SFC in construction but its request traffic volume is one λ-th of original request traffic volume, i.e., f_S/λ. Then we use the Viterbi algorithm to deploy the split chains one by one. Here note that the size of λ must ensure all the split chains can be deployed successfully by the Viterbi algorithm. Here the success means for each of split chain, Viterbi algorithm can find a nonempty solution. In this work, the λ is regard as an input parameter and it can be any reasonable value which satisfies above condition. As all the split chains have the same ingress nodes and egress nodes, after deploying all the split chains, we actually obtain a solution for our model as shown in Fig. 4(c). Now we take this solution as the basic solution (or start solution).

Above is the main idea of how the procedure *SplitChainViterbi*(.) finds the basic solution. As each split chain is deployed by Viterbi algorithm with a global perspective, the obtained solution is also a global relatively high-quality solution and we start from this solution, further optimize it through some improvement procedures.

B. IMPROVEMENT PROCEDURE

1) PROCEDURE OF CLASSIFYING

Before taking the improvement procedures, we first classify all the VNFs through *Classify*(.) as shown in Alg. 3. The *Classify*(.) classifies the VNFs mainly according to

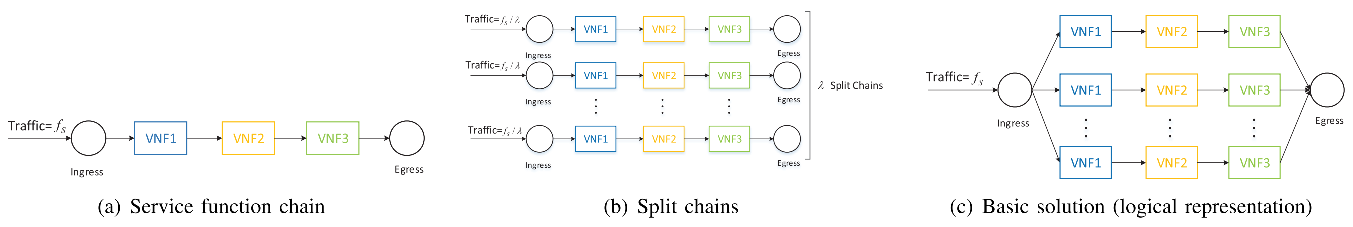


FIGURE 4. Find basic solution. (a) Service function chain. (b) Split chains. (c) Basic solution (logical representation).

Algorithm 3 Procedure *Classify*(.)

```
1:  procedure $Classify(G, A_{res}^{left}, f_S, \lambda, \varepsilon, u_x)$
2:  $count \leftarrow 0$;
3:  for each $m \in N_{sv}$ do
4:      if $c_m^{left} < p_{u_x}\frac{f_S}{\lambda}\cdot\prod_{i=1}^{x-1}\eta_{u_i}$ then
5:          $count \leftarrow count + 1$;
6:      end if
7:  end for
8:  if $count > \lfloor \varepsilon \cdot Card(N_{sv}) \rfloor$ then
9:      this VNF is $BigVNF$;
10: end if
11: end procedure
```

each VNF's p_{u_x} and the detailed steps are as follow. The *Classify*(.) procedure first calculates each instance's resource occupation d_{u_x} of VNF in basic solution. Here note that all the instances' resource occupations of this VNF are same because all the split chains have same traffic volume. Then *Classify*(.) counts the number of service nodes whose remaining resource is smaller than d_{u_x} and if this number is larger than $\lfloor \varepsilon \cdot Card(N_{sv}) \rfloor$, the *Classify*(.) classifies this VNF as "Big VNF". Here ε is the predefined precision parameter and $0 < \varepsilon < 1$. The *Classify*(.) procedure terminates when all the VNFs in SFC have been classified.

After the classification, for the VNFs which are not "Big VNFs", we first use the improvement procedure *Merge*(.) to process their instances then we continue to use the improvement procedure *Split*(.) to process the instances of "Big VNFs". Here note that after these procedures, some split chains will be changed and in the following part of introducing improvement procedures, the "split chain" represents the original split chain in the basic solution of the hop or branch that is currently being processed.

2) PROCEDURE OF MERGING

It can be observed in obtained basic solution that the instance number of each VNF is same. However, for some VNFs, especially which have the small size of resource demand coefficient p_{u_x}, the instance number in basic solution may be superfluous. So here we introduce the first improvement procedure *Merge*(.) as shown in Alg. 4. If one VNF meets the trigger condition of *Merge*(.), the *Merge*(.) starts from the first split chain in current solution and tries to continuously merge the instance in next following split chain, as shown in Fig. 5. Here the "merge" indicates using one instance to replace the spare instances. The detailed merging process is as follow. We first assume that the *Merge*(.) procedure has now successfully merged k instances of u_x (the initial value of k is one), in other words, these k instances have been replaced

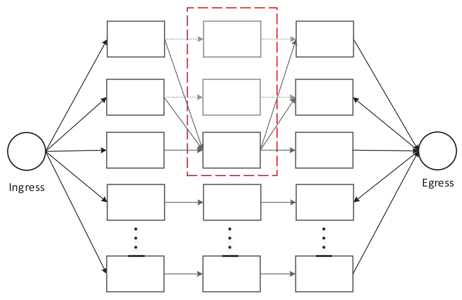


FIGURE 5. Procedure of merging.

Algorithm 4 Procedure *Merge*(.)

```
1:  procedure $Merge(G, A_{res}^{left}, S, f_S, \lambda, X, u_x)$
2:  $X_{cur} \leftarrow X$, $count \leftarrow 1$;
3:  for $k = 1; k \leq \lambda - 1; k++$ do
4:      $X_{org} \leftarrow X_{cur}$;
5:      Release $u_{x,k}$, $u_{x,k+1}$;
6:      for each $m \in N_{sv}$ do
7:          if $c_m^{left} > p_{u_x}(count + 1)\frac{f_S}{\lambda}\cdot\prod_{i=1}^{x-1}\eta_{u_i}$ then
8:              Assume $u_{x,k}$ is deployed on $m$;
9:              Get this assumed solution $X_{asu}$;
10:             if $TotalCost(X_{asu}) < TotalCost(X_{cur})$ then
11:                 $X_{cur} \leftarrow X_{asu}$;
12:                 $A_{res}^{left} \leftarrow refresh(A_{res}^{left})$;
13:             end if
14:         end if
15:     end for
16:     if $X_{org} = X_{cur}$ then
17:         $count \leftarrow 1$;
18:     else
19:         $count \leftarrow count + 1$;
20:     end if
21: end for
22: return $X \leftarrow X_{cur}, A_{res}^{left}$;
23: end procedure
```

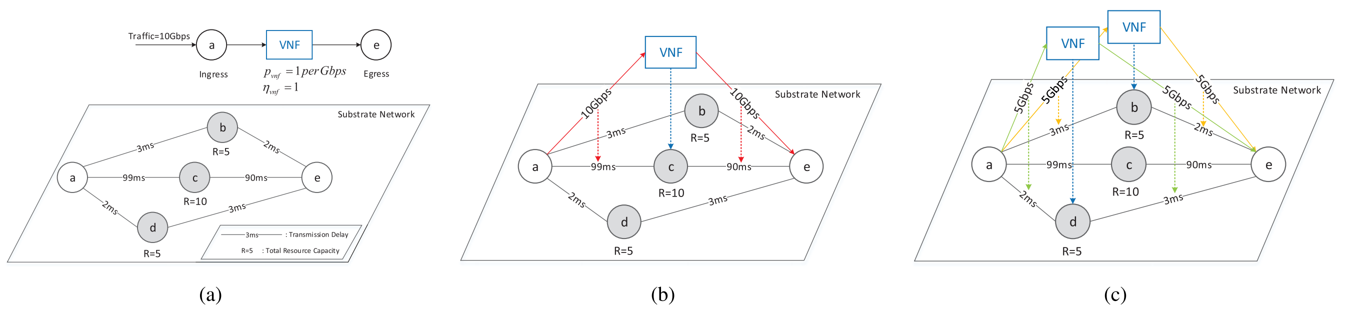


FIGURE 6. An example of SFC deployment.

by a new instance and the traffic volume on this new instance becomes to k times of the original traffic volume reaching to each instance of u_x in basic solution, i.e., $k \cdot (\frac{f_S}{\lambda} \prod_{i=1}^{x-1} \eta_{u_i})$. And we also assume the total cost of current solutions is C_{cur} (the initial value of C_{cur} is the total cost of basic solution). Now the *Merge*(.) procedure continues to try to merge above instance with the instance in next following split chain (assuming this split chain is the l-th chain) with following steps. The *Merge*(.) first eliminates these two instances to release their resource occupations and it searches all the service nodes which have enough resources to carry the traffic with amount of $(k+1)\cdot(\frac{f_S}{\lambda} \prod_{i=1}^{x-1} \eta_{u_i})$. Each node meeting above condition is a candidate node to deploy next new merging instance and the *Merge*(.) calculates each candidate solution's total cost then selects the minimum one. If the total cost of this selected solution is also smaller than C_{cur}, the current solution will be replaced by the selected solution and the *Merge*(.) continues next merging process. However, if no nodes can carry above traffic volume or the minimum cost is larger than C_{cur}, this merging process is considered to be failed. In this case, the value of k is reset to one and the *Merge*(.) procedure restarts from the l-th split chain. The *Merge*(.) procedure terminates when all the instances have been processed.

3) PROCEDURE OF SPLITTING

In the basic solution, each split chain is deployed by Viterbi algorithm. As mentioned above, Viterbi algorithm can obtain optimal solution, however, such optimality is strictly under the condition that each VNF has only one instance. So if we consider the case of multiple instances, there may be better deployment for each split chain. For example, in Fig. 6, we deploy the chain shown in Fig. 6(a) to the substrate network. If we use Viterbi algorithm to deploy this chain, it will deploy the chain as shown in Fig. 6(b). However, the deployment shown in Fig. 6(c) is obviously much better than the former as the links $e_{a,c}$ and $e_{c,e}$ have very large transmission delay. Viterbi algorithm has ignored the links with small transmission delay such like $e_{a,b}$ and $e_{a,d}$ because the integrant nodes b and d do not have enough resource to carry the 10 units traffic volume. Above example indicates

Algorithm 5 Procedure *Split*(.)

```
1: procedure $Split(G, A_{res}^{left}, S, f_S, \lambda, X, u_x)$
2: $X_{cur} \leftarrow X$;
3: for $k = 1; k \le \lambda; k++$ do
4:     $X_{org} \leftarrow X_{cur}$;
5:     Release $u_{x,k}$;
6:     $\forall m_i \in N_{sv}, m_j \in N_{sv}$:
7:     Assume split instances $u_{x,k}^1, u_{x,k}^2$ are
8:     deployed on $m_i, m_j$ respectively;
9:     $\tau \leftarrow HopSCH(N_{src}, F_{src}, N_{cur}, C_{sc}, R_{rem}, N_{des}, C_{cd})$;
10:    Get this assumed solution $X_{asu}$;
11:    if $TotalCost(X_{asu}) < TotalCost(X_{cur})$ then
12:        $X_{cur} \leftarrow X_{asu}$;
13:        $A_{res}^{left} \leftarrow refresh(A_{res}^{left})$;
14:    end if
15: end for
16: return $X \leftarrow X_{cur}, A_{res}^{left}$;
17: end procedure
```

that Viterbi algorithm may miss some good paths whose integrant nodes do not have enough resource. Besides, above case more likely occurs in deploying the VNFs which have the big size of resource demand coefficient p_{u_x} because under the same traffic volume, they request more resources so that the Viterbi algorithm may miss more nodes. So here we introduce the second improvement procedure *Split*(.) as shown in Alg. 5. If one VNF meets the trigger condition of *Split*(.), the *Split*(.) starts from the first split chain in current solution and tries to split each instance of the VNF, as shown in Fig. 7. Here the "split" indicates using two instance to replace the original instance so that the traffic can be split. The detailed splitting process is as follow. We assume the total cost of current solutions is C_{cur}. The *Split*(.) first eliminates the target instance to release its resource occupation then starts to search the appropriate nodes for two new instances through procedure *HopSCH*(.). For every two nodes, the *Split*(.) assumes the two new instances are deployed on these two nodes then the *HopSCH*(.) scheduling the optimal traffic of generated hops by using linear programming which can be solved in polynomial time. The input parameters include nodes set hosting source VNF (i.e. previous type of VNF)

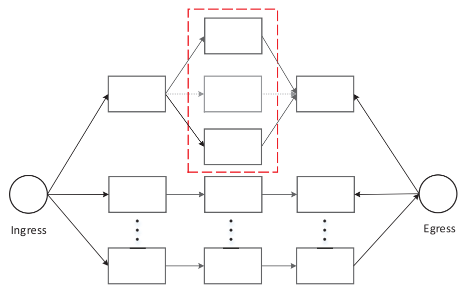


FIGURE 7. procedure of splitting.

instances $N_{src} = \{m_i | i = 1, 2, \ldots, I\}$, traffics from each source VNF instances $F_{src} = \{f_i | i = 1, 2, \ldots, I\}$ (here $I = 1$ or $I = 2$ as previous VNF may also be split), nodes set hosting current VNF (i.e. current type of VNF) instances $N_{cur} = \{m_j | j = 1, 2\}$, unit link cost $C_{sc} = \{c_{ij} | i = 1, 2, \ldots, I; j = 1, 2\}$ between N_{src} and N_{cur}, remaining resource on current nodes $R_{rem} = \{r_j | j = 1, 2\}$, nodes set hosting destination VNF instances $N_{des} = \{m_k | k = 1\}$, unit link cost $C_{cd} = \{c_{jk} | j = 1, 2; k = 1\}$ between N_{cur} and N_{des}. When we use τ_{ij} to denote the traffic allocating between source and current VNF instances, the linear programming can be formulated as:

$$\min \sum_{i=1}^{I}\sum_{j=1}^{2} c_{ij}\tau_{ij} + \sum_{j=1}^{2}\sum_{k=1}^{1} c_{jk}\sum_{i=1}^{I}\eta_u\tau_{ij}$$
$$s.t. \begin{cases} C1: \sum_{j=1}^{2}\tau_{ij} = f_i, & i \in \{1, 2, \ldots, I\} \\ C2: \sum_{i=1}^{I} p_u\tau_{ij} \le r_j, & j \in \{1, 2\} \end{cases} \tag{13}$$

Here η_u and p_u are the data rate scaling ratio and resource demand coefficient of current VNF instance respectively. Constraint C1 ensures all traffics from source VNF instances are allocated while constraint C2 ensures the resources on current nodes can carry the allocated traffic volume. Fig. 8 shows the *HopSCH*(.) procedure when $I = 2$. Note that if above linear programming comes out to be no solution, we set the cost of this generated hops to infinite. After the *HopSCH*(.) procedure, the *Split*(.) calculates this newly generated solution's total cost. For every two nodes, the *Split*(.) repeats above steps then finds the solution with minimum total cost. This solution is considered as a candidate solution by *Split*(.) and if the total cost of it is smaller than C_{cur}, the current solution will be replaced by it and the *Split*(.) continues next split chain's splitting process. If not, the *Split*(.) starts next split chain's splitting process. The *Split*(.) procedure terminates when all the instances have been processed.

4) PROCEDURE OF UPDATING

The *Split*(.) procedure may generate new instances and some instances may be allocated small traffic volume while some instances may be allocated large traffic volume by *HopSCH*(.). It is highly possible that these

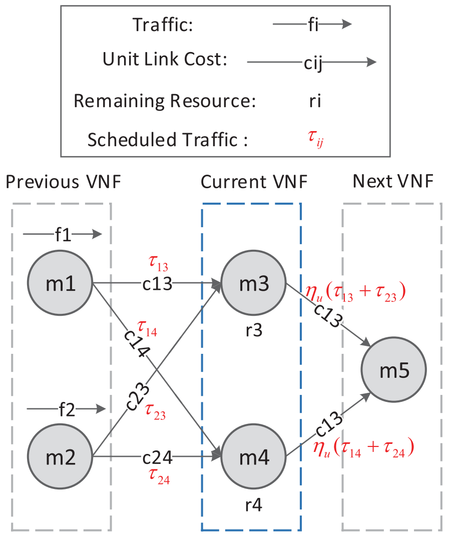


FIGURE 8. Hop scheduling.

small-traffic-allocated instances generated from different split chains can also be merged. On the other hand, these large-traffic-allocated instances may also have the possibility to further be split. So the *Update*(.) procedure as shown in Alg. 6 further updates these newly generated instances. The *Update*(.) first sorts all the instances in ascending order by the amount of allocated traffic volume, then starting from the first instance, the *Update*(.) implements the *Merge*(.) procedure. After the *Merge*(.) procedure, the *Update*(.) continues to implement the *Split*(.) procedure and thus a round of *Update*(.) is completed. Note that the *Split*(.) procedure in *Update*(.) may also generate new instances which means *Update*(.) procedure can be implemented again. So here we set a parameter Ω which represents the total number of *Update*(.) rounds. One can set the Ω to be larger to get better result but it will also spend more time.

Algorithm 6 Procedure *Update*(.)

```
1: procedure $Update(G, A_{res}^{left}, S, f_S, X, u_x)$
2:   $I = \{I_i | i = 1, \ldots, \kappa\} \leftarrow GetNewInstanceSet(X, u_x)$;
3:   $I_{sort} \leftarrow SortByTraffic(I)$;
4:   $(X, A_{res}^{left}) \leftarrow Merge(G, A_{res}^{left}, S, I_{sort}, X, u_x)$;
5:   $(X, A_{res}^{left}) \leftarrow Split(G, A_{res}^{left}, S, I_{sort}, X, u_x)$;
6: end procedure
```

C. TIME COMPLEXITY ANALYSIS

For the *SplitChainViterbi*(.) procedure, as the each split chain is deployed by Viterbi algorithm in $\Theta\left(w_S N^2\right)$ (here $N = Card(N_{sv})$), the *SplitChainViterbi*(.) runs in $T_{scv} = O\left(w_S \lambda N^2\right)$. Obviously, the *Classify*(.) runs in $T_{classify} = O\left(w_S N\right)$. For each type of VNF, the *Merge*(.) processes its instances in $T_{merge} = O\left((\lambda - 1)N\right)$. For the *Split*(.) procedure, let $\phi(L)$ be time complexity of solving the linear programming in Eq. 13. Actually, $\phi(L)$ is very small as there are a few variables and constraints in Eq. 13. Thus, for each type of VNF, the *Split*(.) processes its instances in $T_{split} = O\left(\lambda\phi(L)N^2\right)$. For the *Update*(.) procedure, let $\varphi(G)$ be the time complexity of sort function for sorting G numbers. For the j-th VNF in SFC, when we use $\lambda_1^{i,j}$ and $\lambda_2^{i,j}$ to denotes the number of instances processed by *Merge*(.) and *Split*(.) in the i-th round *Update*(.) procedure respectively, the procedure runs in $T_{update}^{i,j} = O\left(\varphi\left(2\lambda_1^{i,j}\right)+(2\lambda_1^{i,j}-1)N+2\lambda_2^{i,j}\phi(L)N^2\right)$. Finally, in the worst case, considering that the complexity of *Split*(.) is higher than *Merge*(.), we assume all the VNFs in SFC are "Big VNFs" thus the MSV runs in $T_{scv}+T_{classify}+w_S T_{split}+\sum_{i=1}^{\Omega}\sum_{j=1}^{w_S} T_{update}^{i,j}$. Note that this time complexity cannot reflect the typical time complexity of the MSV as it is analyzed under the worst case. In most case, MSV runs much faster than above upper bound.

According to above equation, MSV has the time complexity of $O\left(N^2\right)$ for node number N overall and this is much lower than JoraNFV's [7] and Kariz's [16] (to the best of our knowledge, [7] and [16] are the only works we have known so far which also propose heuristic solutions to solve the TSC NFV-RA problem). In [7], although the time complexity of JoraNFV is not exactly examined, according to the pseudo code of JoraNFV, we can infer that the time complexity of JoraNFV for node number is higher than $O\left(N^2\right)$. On the other hand, the Kariz achieves the time complexity of $O\left(N^3\log N\right)$ for node number. Both JoraNFV and Kariz take the iterative deployment strategy so some extra anti-local-optimal measures are added to them in order to avoid being trapped in local optimality. But these measures also increase their time complexity. On the contrary, MSV does not take the iterative deployment strategy thus it does not require complex anti-local-optimal measures and achieves relatively low time complexity.

IV. PERFORMANCE EVALUATION

A. SIMULATION SETUP

1) SIMULATED NETWORK

The actual characteristic of substrate network topologies are not well understood now as the network virtualization is a new emerging field. So here we use three typical network models: Random Network, Small-World Network [24] and Scale-Free Network [25] as the simulated networks which are generated with principle proposed in [20]. For each simulated network, the node number is one of {10, 15, 20, 45, 50} and 30% nodes are forwarding nodes while the others are service nodes. For the sake of simplicity, we directly use the abstract total resource capacity introduced in §III to represent the resource capacity on each service node. Resource on each service node, transmission delay on each link and intra-node delay of each node are randomly generated following a uniform distribution given in Table 2.

TABLE 2. Network simulation parameters.

Parameters	Distribution	Mean	Var
Node Resources(unit)	*Uniform*	10	8.33
Transmission Delay (ms)	*Uniform*	2.5	0.0833
Intra-node Delay (ms)	*Uniform*	1.5	0.0833

TABLE 3. Virtual network function parameters.

	VNF1	VNF2	VNF3
Resource demand coefficient p_{u_x} (unit/unit)	1	0.1	0.5
Traffic Scale η_{u_x}	1	1	1
Deployment cost α_{u_x}	3	4	5

2) SFCS AND VNFS

The ingress node and egress node in SFC are randomly selected from forwarding nodes. The request traffic volume is one of {20, 30} units and the SFC is one of following.

- Length-3: $\{m_{in}\rightarrow VNF1\rightarrow m_{out}\}$
- Length-4: $\{m_{in}\rightarrow VNF1\rightarrow VNF2\rightarrow m_{out}\}$
- Length-5: $\{m_{in}\rightarrow VNF1\rightarrow VNF2\rightarrow VNF3\rightarrow m_{out}\}$

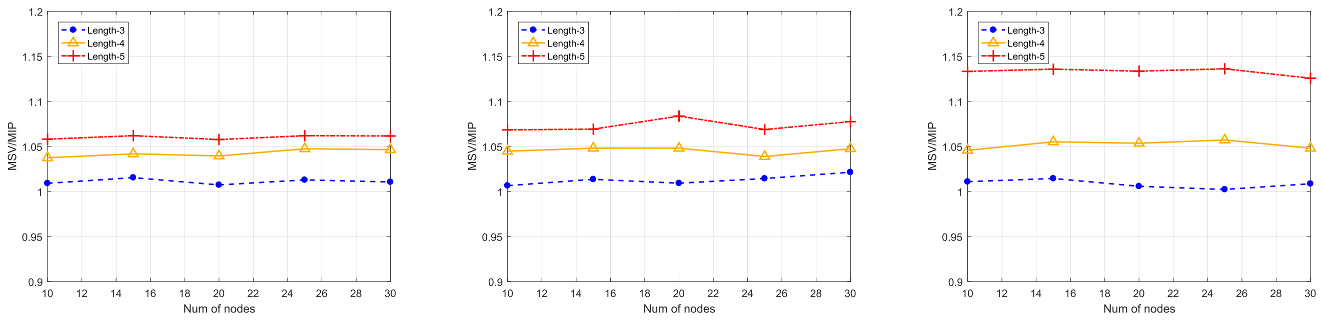
Here Length-i contains all VNFs of Length-(i-1) and the detailed parameters of above VNFs are given in Table 3.

3) PARAMETERS

We evaluate MSV in respect to substrate network type, node number, length of SFC and split chains number. In each experiment, a new network and a new SFC are generated according to above setup and we use MSV to deploy this SFC to this network. We repeat each experiment under the same parameters 10 times and report the average. Note that under above setup (generating principle of resource capacity and setting of request traffic volume), obviously, the SFCs whose request traffic volume is 20 units need to be split to at least two split chains while 30 units SFCs need at least three split chains. So when using MSV to deploy 20 units SFCs, we set $\lambda = 2$ while for 30 units SFCs, we set $\lambda = 3$. Note that in rare cases, especially when the scale of generated network is small, the accompanied generated SFC requires more split chains than above settings. In this case, we abandon above SFC and network then regenerate the SFC and network in order to unify the number of split chains for later time performance evaluation as the number of split chains has influence on the execution time according to previous time complexity analysis. Besides, the precision parameter ε in *Classify*(.) procedure is set to $\varepsilon = 0.4$ and the round parameter Ω is set to $\Omega = 1$.

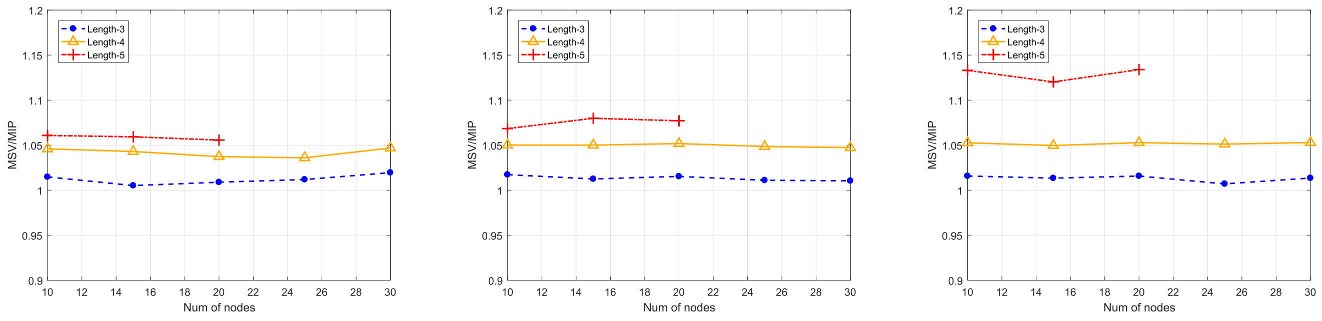
4) EVALUATION METHOD AND ENVIRONMENT

We evaluate the MSV from following two perspectives: cost performance compared to MIP implement by CPLEX (we consider the solutions obtained by MIP implement as optimal solutions) and time performance. We carry out our simulations in a PC with four 3.5GHz CPU cores and 32GB memories using CPLEX 12.6.0.0 (MIP implement) and Matlab 2016b (SFC and network generation, MSV implement).



(a) (b) (c)

FIGURE 9. Cost Performance of MSV with 20 units traffic volume ($\lambda = 2$). (a) Random Network. (b) Small-World Network. (c) Scale-Free Network.



(a) (b) (c)

FIGURE 10. Cost Performance of MSV with 30 units traffic volume ($\lambda = 3$). (a) Random Network. (b) Small-World Network. (c) Scale-Free Network.

TABLE 4. Cost Factors.

Factors	Delay penalty coefficient σ	Bandwidth cost coefficient ς
Value	1	1

Besides, we use the linear programming function using the Dual-Simplex algorithm in Matlab optimization toolbox to solve the linear programming in *HopSCH*(.) procedure and use Matlab's own sort function to sort the branches in *Update*(.) procedure.

B. COST PERFORMANCE

The related cost factors are set as in Table 4 and all the relative importance coefficients are set to one. We compare the MSV's cost performance with MIP's as shown in Fig. 9 and Fig. 10. The reported values are the ratio of MSV's costs and MIP's costs. Here note that as the MIP implement by CPLEX takes too much time when the scale of network becomes larger, we compare MSV with MIP from 10 nodes scale to 30 nodes scale (20 nodes scale network at most for 5 length SFCs with $\lambda = 3$) for each type of network. The result shows that the cost performance of MSV mainly influenced by the length of SFC. With the increase of SFC length, the performance gradually becomes worse. It can be observed that among three kinds of networks, the Scale-Free Network's performance is influenced most by the growth of the SFC's length. On the other hand, the split chains number do not have obvious impact on the performance. Besides, for each different length SFC in each type of network, we can observe that the total cost ratio does not have an obvious change or fluctuation when the number of nodes increase which means MSV does not be affected by the scale of network and can always find the solution in global range. The reason is that the procedures of finding basic solution and improvement are always proceeded from global perspective thus MSV can always get the solution in global range and effectively avoid being trapped in local optimality. Overall, the result shows MSV can usually get the solution whose total cost within 1.15 times of MIP's.

MSV also performs well when compared to JoraNFV [7] and Kariz [16] in total cost performance (both [7] and [16] consider a total cost consisting of various costs as the final optimal objective and take the cost ratio of sub-solution found by algorithms to optimal solution as the main evaluation method of algorithms). When we consider 5-length SFC deployment, MSV performs better than JoraNFV and achieves comparable performance to Kariz. According to the performance evaluation in [7], the ratio to optimal solution of JoraNFV is about 1.15 in average and within 1.25. However, we can observe that the ratio to optimal solution of MSV is about 1.1 in average and within 1.15 from Fig. 9 and Fig. 10. On the other hand, the ratio to optimal solution of Kariz is about 1.05 in average and within 1.1 according to the performance evaluation in [16]. Although MSV performs slightly worse than Kariz on the ratio to optimal solution, it has much lower time complexity than Kariz's.

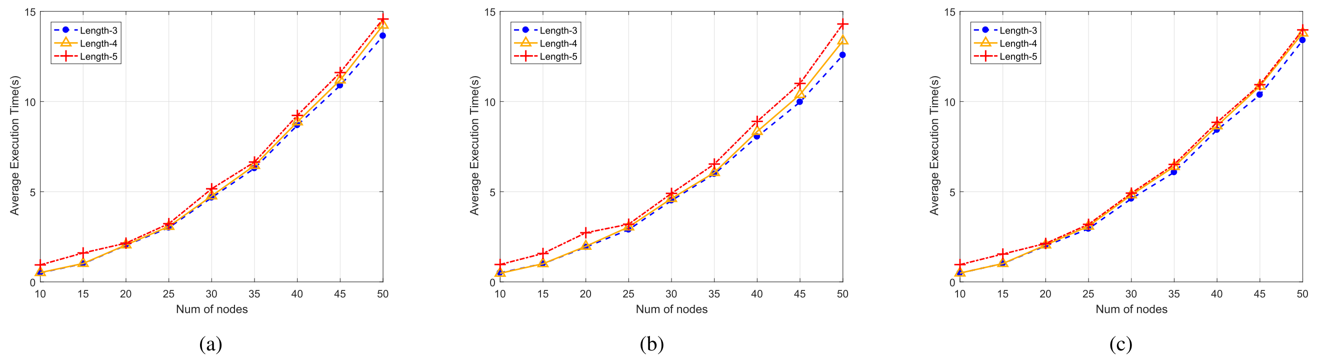


FIGURE 11. Time Performance of MSV with 20 units traffic volume ($\lambda = 2$). (a) Random Network. (b) Small-World Network. (c) Scale-Free Network.

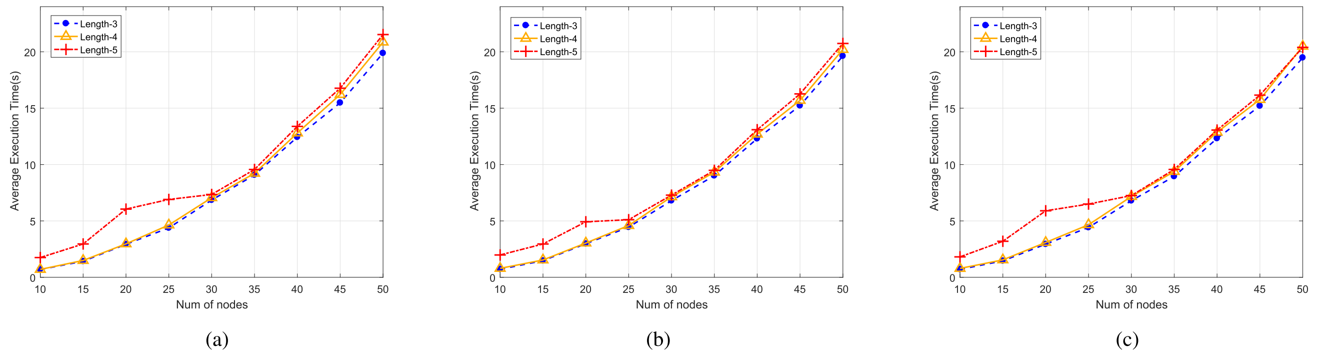


FIGURE 12. Time Performance of MSV with 30 units traffic volume ($\lambda = 3$). (a) Random Network. (b) Small-World Network. (c) Scale-Free Network.

C. TIME PERFORMANCE

To evaluate the time performance of MSV, we plot the average execution time on each type of network, as shown in Fig. 11 and Fig. 12. The result shows that in each type of network, the average execution time increases as the number of nodes and the length of SFC increase. Besides, by comparing Fig. 11 and Fig. 12, we can also find that MSV takes more time when the number of split chains increases. The type of substrate network does not impact much on execution time as the MSV takes similar time to deploy a SFC on different type network.Here note that in both Fig. 11 and Fig. 12, the 5-length SFCs' average execution time is much larger than other different length SFCs when the scale of network is relatively small. The reason is that, as shown in Table. 3, the resource demand coefficient of VNF1 is set to be relatively large thus in most case, VNF1 is considered as "Big VNF" by MSV and their instances are processed by *Split*(.) procedure. Conversely, VNF2's instances are processed by *Merge*(.) procedure at most case as its resource demand coefficient is relatively small. However, the VNF3's resource demand coefficient is set to be middle thus it is processed differently in different scale of network. In small scale networks, it is usually judged as "Big VNF" by MSV and processed by *Split*(.) procedure. However, with the number of nodes increase, VNF3 is gradually considered as not "Big VNF" then processed by *Merge*(.) procedure instead. As mentioned before, the complexity of *Split*(.) is higher than *Merge*(.), so the 5-length SFCs' average execution time increases rapidly at first then tends to slow.

TABLE 5. Average execution time.

Network Scale	MIP	MSV
10 nodes	About 11h	1.75s
20 nodes	About 45h	6.048s
30 nodes	∞	7.346s

Table. 5 shows the average execution time comparison between MSV and MIP implement when deploying 5-length SFC (here $\lambda = 3$) on random network. It can be observed that the MIP takes about 45 hours to deploy a 5-length SFC on 20 nodes scale random network while MSV just only takes about 6.048 seconds under aforementioned environment. Thus, the performance shows MSV can deploy SFC in relatively short time and can be used for real-time SFC deployment.

V. CONCLUSION

In this work, we propose a heuristic solution MSV to solve a typical three-stage coordinated NFV resource allocation model. MSV can automatically determine the appropriate number of VNF instances without a predefined maximum instance number threshold. Besides, MSV does not take the iterative deployment strategy thus avoids complex anti-local-optimal measures. The main idea of MSV is first to obtain a global basic solution then further to improve it through some improvement procedures. The experimental result shows that MSV can get the solution in global range and achieves total cost ratio within 115% compared to the MIP implement. In addition, MSV can also deploy SFC in relatively short time. In our future work, we would like to continue to investigate some more effective heuristic approaches to solve the three-stage coordinated NFV resource allocation.

REFERENCES

[1] P. Quinn and T. Nadeau, *Problem Statement for Service Function Chaining*, document RFC 7498, 2015.

[2] *C-RAN: The Road Towards Green RAN; White Paper. Version 2.5*, China Mobile Res. Inst., Beijing, China, Oct. 2011.

[3] J. Wu, Z. Zhang, Y. Hong, and Y. Wen, "Cloud radio access network (C-RAN): A primer," *IEEE Netw.*, vol. 29, no. 1, pp. 35–41, Jan. 2015.

[4] N. Operators, "Network functions virtualization, an introduction, benefits, enablers, challenges and call for action," in *Proc. SDN OpenFlow SDN OpenFlow World Congr.*, 2012.

[5] B. Han, V. Gopalakrishnan, L. Ji, and S. Lee, "Network function virtualization: Challenges and opportunities for innovations," *IEEE Commun. Mag.*, vol. 53, no. 2, pp. 90–97, Feb. 2015.

[6] J. G. Herrera and J. F. Botero, "Resource allocation in NFV: A comprehensive survey," *IEEE Trans. Netw. Service Manage.*, vol. 13, no. 3, pp. 518–532, Sep. 2016.

[7] L. Wang, Z. Lu, X. Wen, R. Knopp, and R. Gupta, "Joint optimization of service function chaining and resource allocation in network function virtualization," *IEEE Access*, vol. 4, pp. 8084–8094, 2016.

[8] J. Liu, Y. Li, Y. Zhang, L. Su, and D. Jin, "Improve service chaining performance with optimized middlebox placement," *IEEE Trans. Services Comput.*, vol. 10, no. 4, pp. 560–573, Jul. 2017.

[9] T. Wang and M. Hamdi, "*Presto*: Towards efficient online virtual network embedding in virtualized cloud data centers," *Comput. Netw.*, vol. 106, pp. 196–208, Sep. 2016.

[10] A. Gupta, M. F. Habib, P. Chowdhury, and M. Tornatore, "On service chaining using virtual network functions in network-enabled cloud systems," in *Proc. IEEE Int. Conf. Adv. Netw. Telecommun. Syst.*, Dec. 2016, pp. 1–3.

[11] T. Lukovszki and S. Schmid, *Online Admission Control and Embedding of Service Chains*. Cham, Switzerland: Springer, 2014.

[12] M. T. Beck and J. F. Botero, "Scalable and coordinated allocation of service function chains," *Comput. Commun.*, vol. 102, pp. 78–88, Apr. 2016.

[13] X. You, X. Wang, A. Chen, and G. Luo, "A coordinated algorithm with resource evaluation for service function chain allocation," in *Proc. IEEE Int. Conf. Big Data Cloud Comput.*, Oct. 2016, pp. 45–49.

[14] H. Huang, P. Li, S. Guo, W. Liang, and K. Wang, "Near-optimal deployment of service chains by exploiting correlations between network functions," *IEEE Trans. Cloud Comput.*, to be published, doi: 10.1109/TCC.2017.2780165.

[15] I. Jang, D. Suh, S. Pack, and G. Dán, "Joint optimization of service function placement and flow distribution for service function chaining," *IEEE J. Sel. Areas Commun.*, vol. 35, no. 11, pp. 2532–2541, Nov. 2017.

[16] M. Ghaznavi, N. Shahriar, S. Kamali, R. Ahmed, and R. Boutaba, "Distributed service function chaining," *IEEE J. Sel. Areas Commun.*, vol. 35, no. 11, pp. 2479–2489, Nov. 2017.

[17] M. T. Beck and J. F. Botero, "Coordinated allocation of service function chains," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, Dec. 2015, pp. 1–6.

[18] X. You, X. Wang, A. Chen, and G. Luo, "A coordinated algorithm with resource evaluation for service function chain allocation," in *Proc. IEEE Int. Conf. Big Data Cloud Comput. (BDCloud), Social Comput. Netw. (SocialCom), Sustain. Comput. Commun. (SustainCom) (BDCloud-SocialCom-SustainCom)*, Oct. 2016, pp. 45–49.

[19] S. Mehraghdam, M. Keller, and H. Karl, "Specifying and placing chains of virtual network functions," in *Proc. IEEE 3rd Int. Conf. Cloud Netw. (CloudNet)*, Oct. 2014, pp. 7–13.

[20] H. Li, L. Wang, X. Wen, Z. Lu, and L. Ma, "Constructing service function chain test database: An optimal modeling approach for coordinated resource allocation," *IEEE Access*, vol. 6, pp. 17595–17605, 2018.

[21] M. Bouet, J. Leguay, T. Combe, and V. Conan, "Cost-based placement of vDPI functions in NFV infrastructures," *Int. J. Netw. Manage.*, vol. 25, no. 6, pp. 490–506, 2015.

[22] G. D. Forney, Jr., "The Viterbi algorithm," *Proc. IEEE*, vol. 61, no. 3, pp. 268–278, Mar. 1973.

[23] F. Bari, S. R. Chowdhury, R. Ahmed, R. Boutaba, and O. C. M. B. Duarte, "Orchestrating virtualized network functions," *IEEE Trans. Netw. Service Manage.*, vol. 13, no. 4, pp. 725–739, Dec. 2016.

[24] D. J. Watts and S. H. Strogatz, "Collective dynamics of 'small-world' networks," *Nature*, vol. 393, no. 6684, pp. 440–442, 1998.

[25] A.-L. Barabási and E. Bonabeau, "Scale-free networks," *Sci. Amer.*, vol. 288, no. 5, pp. 60–69, 2003.

HANG LI received the B.S. degree in communication engineering from Jilin University, Changchun, China. He is currently pursuing the M.S. degree in communications engineering with the Beijing University of Posts and Telecommunications, China. His current research interests include network architecture and network function virtualization.

LUHAN WANG received the Ph.D. degree from the Beijing University of Posts and Telecommunications (BUPT) in 2017. In 2017, he joined the School of Information and Communication Engineering, BUPT, as an Assistant Professor. His current research interests include network architecture, network function virtualization, and soft-defined networks.

XIANGMING WEN received the B.E., M.S., and Ph.D. degrees in electrical engineering from the Beijing University of Posts and Telecommunications (BUPT), Beijing, China. He is currently the Vice President of BUPT, where he is also a Professor with the Communication Network Center and the Director of the Beijing Key Laboratory of Network System Architecture and Convergence. He is also the Vice Director of the Organization Committee of the China Telecommunication Association. In the last five years, he has been the author of more than 100 papers published. His current research is focused on broadband mobile communication theory, multimedia communications, and information processing. He is the Principle Investigator of more than 18 projects, including the National Key Project of Hi-Tech Research and Development Program of China (863 program) and the National Natural Science Foundation of China.

ZHAOMING LU received the Ph.D. degree from the Beijing University of Posts and Telecommunications in 2012. He joined the School of Information and Communication Engineering, Beijing University of Posts and Telecommunications, in 2012. His research includes open wireless networks, QoE management in wireless networks, software-defined wireless networks, cross-layer design for mobile video applications, and so on.

JINYAN LI received the master's degree from the Beijing University of Post and Telecommunication. She is currently a Senior Engineer with the China Telecom Technology Innovation Center. Her research focuses on the standardization and technique evolution of mobile networks, especially in the areas of architecture evolution and service deployment.