FITEE

A virtual service placement approach based on
improved quantum genetic algorithm[*]

Gang XIONG[†1], Yu-xiang HU[1], Le TIAN[2], Ju-long LAN[1], Jun-fei LI[1], Qiao ZHOU[1]

(*¹National Digital Switching System Engineering & Technological Research Center, Zhengzhou 450002, China*)

(*²Department of Mathematics and Computer Science, University of Antwerp, Antwerp 2020, Belgium*)

[†]E-mail: xg1226@126.com

Abstract: Despite the critical role that middleboxes play in introducing new network functionality, management and innovation of them are still severe challenges for network operators, since traditional middleboxes based on hardware lack service flexibility and scalability. Recently, though new networking technologies, such as network function virtualization (NFV) and software-defined networking (SDN), are considered as very promising drivers to design cost-efficient middlebox service architectures, how to guarantee transmission efficiency has drawn little attention under the condition of adding virtual service process for traffic. Therefore, we focus on the service deployment problem to reduce the transport delay in the network with a combination of NFV and SDN. First, a framework is designed for service placement decision, and an integer linear programming model is proposed to resolve the service placement and minimize the network transport delay. Then a heuristic solution is designed based on the improved quantum genetic algorithm. Experimental results show that our proposed method can calculate automatically the optimal placement schemes. Our scheme can achieve lower overall transport delay for a network compared with other schemes and reduce 30% of the average traffic transport delay compared with the random placement scheme.

Key words: Software-defined networking (SDN), Network function virtualization, Quantum genetic algorithm, Middlebox
http://dx.doi.org/10.1631/FITEE.1500494   **CLC number:** TP393

1 Introduction

Current networks rely on rich functionalities, such as improved critical performance (e.g., proxies and load balancers), improved security (e.g., firewalls and the intrusion detection system (IDS)), reduced bandwidth costs (e.g., wide area network (WAN) optimizers), and policy compliance capabilities (e.g., network address translation (NAT) and content filters), which are introduced by a wide spectrum of specialized appliances or middleboxes (Carpenter and Brim,

2002). Sherry *et al.* (2012) showed that the number of middleboxes is on par with the number of routers in a network (e.g., an average very-large network holds 2850 layer-3 routers and 1946 middleboxes). In other words, middleboxes are a critical part of today's networks and it is reasonable to expect that they will remain so in the foreseeable future (Walfish *et al.*, 2004; Joseph and Stoica, 2008).

Though middleboxes are inevitably deployed in networks and are playing a critical role in introducing new network functionality, it is troubling that current middlebox architectures suffer from barriers, such as high cost (Anderson *et al.*, 2012; Anwer *et al.*, 2013), limited flexibility (Rajagopalan *et al.*, 2013), and long development cycles (Sekar *et al.*, 2012). The reason is that today's middleboxes not only are closed and expensive systems with few or no hooks and application programming interfaces (APIs) for extension

ⓘ ORCID: Gang XIONG, http://orcid.org/0000-0002-4249-6820

or experimentation, but also are built on a particular chosen hardware platform that typically supports a narrow range of specialized functions (e.g., IDS). Worse still, middleboxes are acquired from independent vendors and deployed as standalone devices with little uniformity in their management APIs or cohesiveness in how the overall middleboxes are managed (Greenberg *et al.*, 2005).

Given the problems stated above, how to solve these issues of middleboxes has received a significant amount of attention (Hwang *et al.*, 2015). Most recent strategies are built on two kinds of new networking technologies, namely software-defined networking (SDN) (McKeown *et al.*, 2008; ONF, 2012; Nunes *et al.*, 2014) and network function virtualization (NFV) (Chiosi *et al.*, 2012; Li and Chen, 2015). These technologies have emerged aiming at cost reduction, network scalability increase, and service flexibility improvement with the strategies of enabling innovation in network nodes, e.g., standardized APIs and software-centric implementations. NFV proposes to run network functions as software instances on commodity servers or datacenters, while SDN supports a decomposition of the network into control- and data-plane functions. Therefore, these new concepts are considered very promising drivers to design cost-efficient middlebox service architectures (de Turck *et al.*, 2015; Matias *et al.*, 2015).

Although introducing SDN and NFV to the network function has several advantages as mentioned previously, it also brings some challenges for network transmission efficiency (Shen *et al.*, 2015). For example, in the network shown in Fig. 1, an additional traffic transport delay is expected, which requires a thorough planning of the middlebox location within the network. In Fig. 1, two kinds of virtual middleboxes (VMs), i.e., IDS and firewall (FW), operating on general servers 1 and 2, are placed at network nodes R2 and R4. We assume that traffic 1 (red solid curve), which requires both the IDS and FW services, enters the network from border router 1 and exits the network on border router 2, while that traffic 2 (green dashed curve) needing the FW service enters the network from border router 2 and exits the network on border router 3. However, with server 1 supporting only the IDS function and server 2 performing solely the FW function, traffic 1 has to traverse the IDS box in R2 and then the FW box in R4,
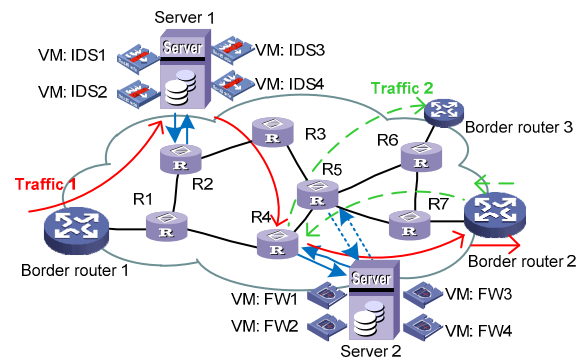

Fig. 1 Motivation of the service deployment problem (References to color refer to the online version of this figure)

and traffic 2 must be steered to the FW box in server 2. Apparently, the placement scheme concerning service locations $\{R2, R4\}$ increases the traffic transport delay, compared with the scheme with locations $\{R2, R5\}$.

While routers/switches process packets at every hop, middleboxes process only packets of a subset of all the hops. Apparently, if the middleboxes are deployed randomly or in some remote nodes, the network traffic may be sent on a detour for the middlebox services, leading to a potential increase in packet latency and bandwidth consumption. Therefore, there is still an orthogonal problem at the network planning stage, i.e., where to place these middlebox services so that this performance penalty is minimized. We denote this problem as the service placement problem.

In this paper, the middlebox service placement problem is discussed, focusing on how to place services in the network with the objective of minimizing the average time it takes for the subscribers' traffic to go through all required services. The following two key contributions are made:

1. We formulate the service placement problem in theory by the integer linear programming model.

2. We propose a heuristic solution based on the improved quantum genetic algorithm (QGA), and evaluate the algorithm performance.

2 Related work

Much current research focuses on the evolution of the middlebox service model. Generally, two complementary approaches are followed. The first

tackles the high building capital expenditures (CAPEX) and limited extensibility by employing a combination of NFV and SDN. It allows operators to decouple the dependence from specialized equipment and operate network functions as virtualized software instances on a standardized platform instead. The second tackles the high operation expenditures (OPEX) and limited flexibility in the service procedure by SDN controlling routing through the specified functional sequence. The main work related to these two approaches is summarized here.

On the one hand, Sherry *et al.* (2012) proposed a practical service framework for outsourcing enterprise middlebox processing to the shared cloud computing platform (Qi *et al.*, 2014). Sekar *et al.* (2011) innovated middlebox deployment with the software-centric middlebox implementations running on general-purpose hardware platforms managed via open and extensible management APIs. Regarding VMs as first-class entities, Gember *et al.* (2012a) presented a framework for immediate application deployment over or under the cloud. Furthermore, Gember *et al.* (2012b) realized a software-defined middlebox networking framework to simplify the management of complex and diverse functionalities. In the scenarios of NFV and SDN, Gember *et al.* (2014) designed a control plane called OpenNF, which could provide efficient, coordinated control of both internal middlebox state and network forwarding state.

On the other hand, Qazi *et al.* (2013) presented the SIMPLE architecture, an SDN-based policy enforcement layer for efficient middlebox-specific traffic steering. Built upon SDN and the OpenFlow protocol, Zhang *et al.* (2013) proposed a scalable framework (called StEERING) for dynamic traffic routing through any sequence of middleboxes. Fayazbakhsh *et al.* (2014) developed the FlowTags architecture, which consists of SDN controllers and FlowTags-enhanced middleboxes, to integrate middleboxes into SDN-capable networks. Gushchin *et al.* (2015) proposed a solution for routing traffic in an SDN-enabled dynamic network environment with consolidated middleboxes implemented using virtual machines. Cheng *et al.* (2015) used the simulating annealing algorithm for the combinational problem of service chains, which could manage network services in an efficient and scalable way.

The above studies either are based on the assumption that the service has been deployed or make only some preliminary exploration on the service problem (Basta *et al.*, 2014; Lange *et al.*, 2015; Mohammadkhan *et al.*, 2015). Few studies have designed a specific deployment strategy, which is our focus in the next section.

3 Proposed solution

Our goal in this section is to address the service placement problem by combining the theory of QGA with the structure of SDN and NFV, which includes today's SDN controller (Gude *et al.*, 2008), OpenFlow switches, and virtual network function components. Our solution is an optimal middlebox service placement policy maker that decides a reasonable high-level deployment policy for the network.

3.1 SDN/NFV-based architecture

Fig. 2 gives an overview of our architecture, where virtual implementations of middlebox applications are consolidated to run on a general-purpose shared hardware platform, managed in a logically centralized manner with uniform APIs for a network-wide view. This SDN and NFV based solution reduces the cost and development cycles to build and deploy new middlebox applications.
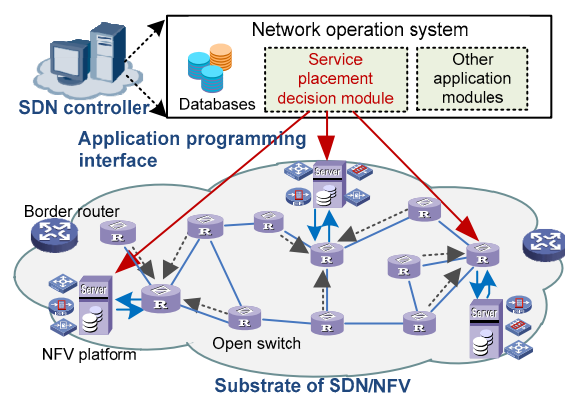

Fig. 2 Overview of middlebox deployment using SDN/NFV

As shown in Fig. 2, the components of the architecture can be classified into three kinds: (1) the control plane components including the network

operation system, decision module, and databases; (2) the data plane components containing the OpenFlow switches and VM platforms; and (3) the interfaces between the control plane and the data plane, such as the OpenFlow protocol. Next, we will describe the roles of the main components and how the proposed solution can be used in the context of SDN and NFV.

The controller is the central administrator of the network and plays a core role in our proposed scheme. The controller periodically collects network state information, including network topology, service function description, network resources (e.g., bandwidth and network-wide traffic workload), and stores them in the databases. The substrate of SDN/NFV contains routers/switches and NFV platforms, which forward the traffic and provide middlebox service. In addition, the APIs are responsible mainly for the communication tasks between the control- and data-planes.

The procedure of our placement scheme is as follows: First, with the input databases in the controller and the requirement of placement, the SDN controller runs the service placement decision module, which solves an optimization problem of minimizing network-wide transport delay. Second, the results of the decision module are output of the configuration policy to guide service placement operation by the uniform APIs. Finally, NFV platforms are placed on network nodes with optimal locations, which can provide middlebox services for the traffic from the nodes without the service placed (as illustrated by dashed lines in Fig. 2).

3.2 Integer programming model for service placement

We formulate the service deployment problem as an optimization problem that aims at minimizing the transport delay or distance to be traversed by all subscribers' traffic (Lu et al., 2013). Assume that the network topology is defined as an undirected graph $G=(V, E)$, with node set V representing switches and edge set E representing the links. For example, in the topology of Fig. 2, graph G is a symmetric graph with weighted edges, and each edge is associated with a transport delay value $d(l)$ ($l \in E$).

The objective is to find a subset V_S of the locations among all candidates V ($1 \le |V_S| \le |V| = N$, $|\cdot|$ is the cardinality of a set), and place the services in these selected locations so that the total delay for all the users is minimized. The optimization problem can be considered as an integer linear programming (ILP) model, whose feasible solution defines a scheme that satisfies our objectives. The problem is formulated as follows:

First, given two nodes v_i and v_j (v_i, $v_j \in V$, $v_i \ne v_j$), the minimum transport delay between v_i and v_j is calculated by

$$d(v_i, v_j) = \arg\min_{p_k \in P(v_i, v_j)} \sum_{l \in p_k} d(l), \qquad (1)$$

where $P(v_i, v_j)$ denotes the set of paths from node v_i to node v_j, and p_k is one path element of set $P(v_i, v_j)$.

Let $\boldsymbol{D}=[d(v_i, v_j)]_{N \times N}$ ($i, j=1, 2, \ldots, N$) denote the shortest path matrix of graph G, and V^E ($V^E \subset V$) the egress node set. The linear optimization model is shown as

$$\min\left(\sum_{i=1}^{N} x_i d(v_i, v_i^E) + \sum_{i=1}^{N}(1-x_i)d(v_i, v_i^S)\right) \qquad (2)$$

$$\text{s.t. } \forall v_i \in V, \ v_i^S = \{v_n \mid v_n \in V, x_n = 1, \\ d(v_i, v_n) = \arg\min_{j=1,2,\ldots,N, \ x_j=1} d(v_i, v_j)\}, \qquad (3)$$

$$\forall v_i \in V, \ v_i^E = \{v_k \mid v_k \in V^E, \\ d(v_i, v_k) = \arg\min_{v_j \in V^E} d(v_i, v_j)\}, \qquad (4)$$

$$x_i = \{0, 1\}, \quad i = 1, 2, \ldots, N, \qquad (5)$$

where $x_i \in \{0, 1\}$ ($i=1, 2, \ldots, N$) are the variables of the optimization ($x_i=0$ means that node v_i is not selected for service placement; otherwise, v_i is the location of the network service), v_i^S is the service node corresponding to node v_i, and v_i^E is the egress point corresponding to node v_i. Expression (2) defines the total transport delay, which is calculated as the sum of the transport delay between the ingress points and the service points and the transport delay between the service points and the egress points. Constraint (3) means assigning the service node with the minimum transport delay to v_i as v_i^S. Constraint (4) indicates that the egress node with the minimum transport delay to v_i is selected as v_i^E.

3.3 Solution based on the improved quantum genetic algorithm

Malossini *et al.* (2008) and Mohammed *et al.* (2012) have shown that the quantum genetic algorithm (QGA) has a good performance in dealing with integer programming. In this study, we extend the basic QGA with some improvement methods, such as dynamic rotation angle mechanism, quantum mutation, and population catastrophe. Then we propose a algorithm (called SP-IQGA) based on the improved quantum genetic algorithm (IQGA) for the ILP model to obtain the optimal service placement (SP).

3.3.1 Introduction to QGA

QGA is based on the concepts of quantum bit and quantum superposition state. The basic unit of information in quantum computation is the qubit. A qubit is a two-level quantum system with basis states $|0\rangle$ and $|1\rangle$, which can be represented by a superposition of the basis states:

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle, \quad \alpha^2 + \beta^2 = 1, \qquad (6)$$

where α and β are complex numbers, denoting the probability amplitudes of the basis states.

QGA operates on a population composed of multiple feasible solutions. Each feasible solution of the QGA, which is made up of multiple qubits, is the element chromosome of the population. If the probability amplitudes of a qubit are $[\alpha, \beta]^T$, a chromosome containing N qubits is described by

$$C = \begin{array}{cccc} q_1 & q_2 & \cdots & q_N \\ \left[\begin{array}{c|c|c|c} \alpha_1 & \alpha_2 & \cdots & \alpha_N \\ \beta_1 & \beta_2 & \cdots & \beta_N \end{array}\right], \end{array} \qquad (7)$$

where each qubit q_i (i=1, 2, …, N) of the chromosome C can be one of the state $|0\rangle$, state $|1\rangle$, and superposition of states $|0\rangle$ and $|1\rangle$, and will collapse into a certain state ($|0\rangle$ or $|1\rangle$) in the observation of chromosome. So, this operation endows the QGA with better population diversity than the basic genetic algorithm.

In our SP-IQGA algorithm, the ith qubit state of chromosome C represents the service information of node v_i; i.e., q_i with state $|1\rangle$ means that node v_i is chosen as the service placement location; otherwise,

q_i with state $|0\rangle$ means that v_i is not placed in the service.

3.3.2 Formulation of the SP-IQGA algorithm

The main steps of the SP-IQGA algorithm can be described as follows:

Step 1: acquisition of the shortest path matrix $\boldsymbol{D}$. $\boldsymbol{D}$ is an important input parameter, and contains all minimum transport delays between any two nodes in graph G. For example, element $d(v_i, v_j)$ of the ith row and jth column in $\boldsymbol{D}$ is the minimum transport delay value between v_i and v_j, which can be calculated by using the Bellman-Ford algorithm.

Step 2: initialization of the QGA. At the initial stage of SP-IQGA, we set the chromosome population size as M and the qubit length of each individual chromosome as N. Denote the tth generation population as $P(t) = \{C_1^{(t)}, C_2^{(t)}, …, C_M^{(t)}\}$, where $C_m^{(t)}$ (m=1, 2, …, M) is as described in Eq. (7). In the initial search of the algorithm, all states appear with the same probability, so we set

$$\alpha_{mi}^{(0)} = \beta_{mi}^{(0)} = \frac{1}{\sqrt{2}}, \quad i=1,2,…,N, \ m=1,2,…M,$$

and obtain

$$C_m^{(0)} = \left[\begin{array}{c|c|c|c} \alpha_{m1}^{(0)} & \alpha_{m2}^{(0)} & \cdots & \alpha_{mN}^{(0)} \\ \beta_{m1}^{(0)} & \beta_{m2}^{(0)} & \cdots & \beta_{mN}^{(0)} \end{array}\right]$$
$$= \left[\begin{array}{c|c|c|c} 1/\sqrt{2} & 1/\sqrt{2} & \cdots & 1/\sqrt{2} \\ 1/\sqrt{2} & 1/\sqrt{2} & \cdots & 1/\sqrt{2} \end{array}\right].$$

Step 3: measurement of the observation value of chromosome C. The chromosome observation is to make each qubit of the chromosome collapse into a certain state. The measurement method is to generate a random number in range [0, 1] for each qubit. If the random number is less than $|\alpha|^2$, the measurement value of the qubit is 0; otherwise, it is 1. After the measurement operation, C is transformed to the observation value $X_C=\{x_1, x_2, …, x_N\}$, where x_i (i=1, 2, …, N) is a binary variable (0 or 1).

Step 4: calculation of the fitness. The fitness is the metric indicating the quality of the individual. The higher the fitness value, the closer the individual to the optimal solution. For individual $X_C=\{x_1, x_2, …, x_N\}$, the fitness function can be obtained by

$$\text{Fit}(X_C) = \left[\sum_{i=1}^{N} x_i d(v_i, v_i^{\text{E}}) + \sum_{i=1}^{N} (1 - x_i) d(v_i, v_i^{\text{S}}) \right]^{-1}. \quad (8)$$

Step 5: adaptive adjustment strategy for the quantum rotation gate. In QGA, the population can be updated by quantum rotation gate $U(\theta)$. Based on the quantum rotation gate, the adjustment operation of the ith qubit in $C_m^{(t)}$ is as follows:

$$\begin{bmatrix} \alpha'_{mi} \\ \beta'_{mi} \end{bmatrix} = U(\theta) \begin{bmatrix} \alpha_{mi} \\ \beta_{mi} \end{bmatrix} = \begin{bmatrix} \cos\theta_i & -\sin\theta_i \\ \sin\theta_i & \cos\theta_i \end{bmatrix} \begin{bmatrix} \alpha_{mi} \\ \beta_{mi} \end{bmatrix}, \quad (9)$$

where α'_{mi} and β'_{mi} represent the probability amplitudes of the ith qubit after adjustment. θ_i denotes the rotation angle of quantum rotation gate, defined by

$$\theta_i = s(\alpha_i, \beta_i) \cdot \Delta\theta_i, \quad (10)$$

where $s(\alpha_i, \beta_i)$ determines the direction of quantum rotation and $\Delta\theta_i$ determines the size of quantum rotation. To reduce the influence of the rotation angle on the algorithm convergence rate, an adaptive method is used to adjust θ_i in this study. Specific adjustment policies are shown in Table 1, where δ is a coefficient related to the convergence rate of the algorithm, and we set it as a variable changing with the number of iterations:

$$\delta = 0.04\pi \left(1 - \sigma \cdot \frac{t}{T+1} \right), \quad (11)$$

where $\sigma \in [0, 1]$ is a constant, t is the current evolution iteration number, and T is the total number of evolution iterations.

Step 6: quantum variation and quantum crossover. The performance of QGA can be improved by quantum variation and quantum crossover. Quantum variation can generate new individuals to prevent QGA from evolving into a local optimal solution. During the variation, we choose a small proportion of individuals from the population, appoint randomly a variable qubit of chromosomes, and swap the probability amplitudes α and β of the appointed qubit. On the other hand, quantum crossover can produce more new models to improve the searching performance of the algorithm. Our specific implementation process is that all individuals in the population are ordered

Table 1 Adjustment policies for the rotation angle

x_i	x_i^{b}	Fit(X)≥Fit(X_b)	$\Delta\theta_i$	$s(\alpha_i, \beta_i)$			
				$\alpha_i\beta_i>0$	$\alpha_i\beta_i<0$	$\alpha_i=0$	$\beta_i=0$
0	1	False	0	0	0	0	0
0	1	True	δ	-1	±1	±1	0
1	0	False	δ	-1	±1	±1	0
1	0	True	δ	1	-1	0	±1
1	1	False	δ	1	-1	0	±1
1	1	True	δ	1	-1	0	±1

X_b is the current optimal solution, and x_i^{b} is the ith qubit of X_b

randomly and then a new population is obtained by cyclically shifting the ith qubit for $i-1$ times in all ordered individuals.

Based on the above description, we input all algorithm parameters and call the SP-IQGA algorithm to obtain the service placement scheme. The process flow is shown in Algorithm 1.

Upon the controller, the decision module runs the SP-IQGA algorithm and achieves the result $X_b = \{x_1, x_2, \ldots, x_i, \ldots, x_N\}$, where x_i represents the node v_i being the service location v^{S}.

4 Performance evaluation

4.1 Experimental environment

To evaluate the performance, we set up the experimental environment on a computer with a 2.67 GHz two-core Intel® Core™ i7 CPU and 4 GB RAM. The GT-ITM tool (Zegura et al., 1996) is used for generating different network topologies, and we implement the proposed algorithm with MATLAB.

We evaluate the SP-IQGA algorithm for the service location described in Section 3 using the test topology from the GT-ITM and the simulated real network traffic from the data center network traffic record. The method for generating links is Waxman, with parameters $\alpha=0.3$ and $\beta=0.2$. The link transport delay in the test topology is measured in millisecond, and the transport delay of each link is randomly distributed in range [1, 100]. In SP-IQGA, we set the population size $M=20$ and the variation probability $r=0.1$.

4.2 Evaluation results

4.2.1 Effectiveness of different parameters

With the intelligent decision of QGA, SP-IQGA can calculate automatically the service node number

Algorithm 1 SP-IQGA

Input: network topology graph G, population size M, chromosome length N, variation probability r, number of evolution iterations T

Output: optimal placement location scheme X_b

1   Run the Bellman-Ford algorithm for calculating the shortest path matrix $\boldsymbol{D}$ of G

2   Generate original population $P(0)=\{C_1^{(0)}, C_2^{(0)}, \ldots, C_M^{(0)}\}$, and initialize each chromosome individual

3   Measure $P(t)$ and obtain observation value $O(t)=\{X_1, X_2, \ldots, X_M\}$

4   **for all** $X_i \in O(t)$ **do**

5   **for all** $x_j \in X_i$ **do**

6   **if** $x_j=0$ **then**

7   Query matrix $\boldsymbol{D}$, and assign a minimum transport delay node v_i^S to x_j

8   **end if**

9   **end for**

10   Calculate the fitness value $\mathrm{Fit}(X_i)$

11   **end for**

12   $X_b=\{X_i|\mathrm{Fit}(X_i)=\mathrm{argmax}_{X \in O(t)}\,\mathrm{Fit}(X)\}$

13   **while** $(t<T)$

14   Evolve t to $t+1$ by quantum rotation gate: $t=t+1$

15   Perform steps 3 to 10

16    $B(t)=\{X_i|\mathrm{Fit}(X_i)=\mathrm{argmax}_{X \in O(t)}\,\mathrm{Fit}(X)\}$

17   **if** $\mathrm{Fit}(X_b)<\mathrm{Fit}(B(t))$ **then**

18     $X_b=B(t)$

19   **end if**

20   **if** X_b has not changed after $N/2$ evolution generations **then**

21   Do quantum crossover for $P(t)$

22   Do quantum variation with probability r for $P(t)$

23   **end if**

24   **end while**

Fig. 3 Service node ratio with different network sizes

Fig. 4 Overall transport delay with different number of iterations

(N_S) under different network topologies. For different network sizes (N_V), the optimal service node ratios (N_S/N_V) with different numbers of egress nodes (N_E) are obtained by SP-IQGA (Eqs. (2)–(5)) (Fig. 3). It is intuitive to find that the service node ratio is in range [0.05, 0.25] and increases with N_E. The reason is that with more egress nodes, more network nodes become suitable for the placing service.

Then, under different values of parameter N_E and set $N_V=100$, we show the overall transport delay (Eq. (2)) of network traffic varying with the iteration number in Fig. 4. SP-IQGA reaches rapidly the convergence state after about 100 iterations and searches toward the optimal solution by quantum variation and quantum crossover (i.e., sharp changes in the overall delays).
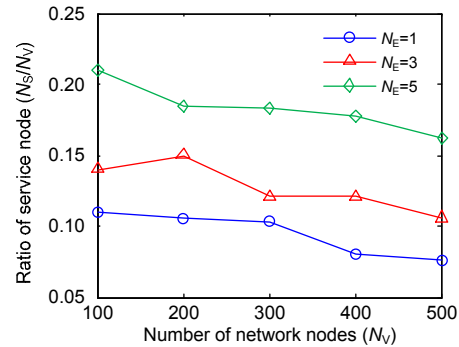
Finally, we test the convergence performance of SP-IQGA by comparison with the genetic algorithm (GA) and the basic quantum genetic algorithm (QGA). Under the conditions $N_E=5$ and $N_V=200$ or 500, we acquire the simulation results of three algorithms. Fig. 5 illustrates that our SP-IQGA converges to the stable states faster. Furthermore, the larger the network size, the more obvious the advantage.
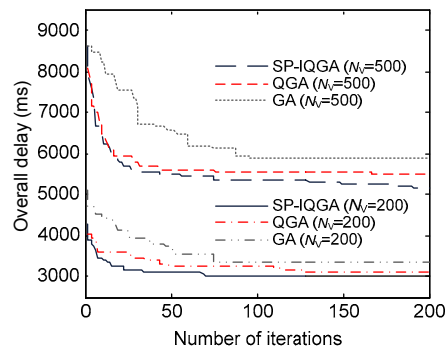

Fig. 5 Convergence comparison of different strategies

Furthermore, we show the time consumptions of operating 200 iterations for the three algorithms in Fig. 6. As shown, with the increase in network size, the computation time of the three methods also increases rapidly. The consumed time of SP-IQGA is more than that of QGA, and both are much higher than that of GA. The reason is that SP-IQGA and QGA execute quantum operation, quantum variation, and quantum crossover, which spend more calculation time. In particular, the dynamic quantum rotation operation in the SP-IQGA algorithm further increases its time consumption.
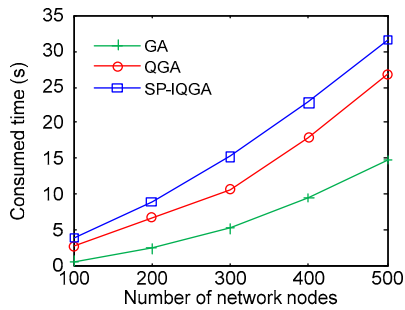

Fig. 6 Comparison of time cost of different strategies

4.2.2 Comparison of different algorithms

To verify the performance of the proposed algorithm, we compare our SP-IQGA with four other placement strategies.

Strategy 1 is based on random placement, denoted as SP-Random; strategy 2 is based on the greedy algorithm, denoted as SP-Greedy (Zhang *et al.*, 2013); strategy 3 is based on a heuristics, denoted as SP-B+COR (Mohammadkhan *et al.*, 2015); strategy 4 is based on another heuristics, denoted as SP-Anneal (Cheng *et al.*, 2015). The notations and descriptions of different algorithms are listed in Table 2.

Fig. 7 shows the simulation results of the five strategies when N_E=5. The overall transport delay of the optimal deployment scheme obtained by different policies increases with the increase of the network size. It is easily seen that the transport delay of SP-IQGA is similar to that of SP-B+COR, and they are both lower than that of the three other strategies under the same network size.

Under the same condition as in Fig. 7, Fig. 8 shows the computational time overhead of the five algorithms. Because the random strategy does not need to solve the optimization problem, its time

Table 2 Comparison of algorithms

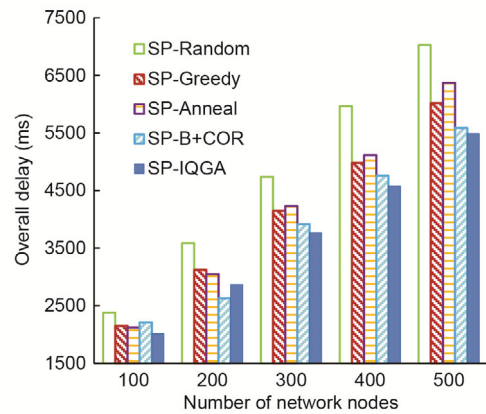| Algorithm	Description
SP-Random	Service placement based on a random number generator
SP-Greedy	Service placement based on a greedy algorithm (Zhang *et al.*, 2013)
SP-B+COR	Service placement based on a heuristic method of partitioning flows (Mohammadkhan *et al.*, 2015)
SP-Anneal	Service placement based on a simulating annealing algorithm (Cheng *et al.*, 2015)
SP-IQGA	Service placement based on an improved quantum genetic algorithm proposed in this study

Fig. 7 Comparison of overall transport delay under different network sizes

overhead is set to zero. The simulation results demonstrate that the calculation time of SP-B+COR is the highest among the algorithms compared and its time consumption increases rapidly with the increase of the network size. Relatively speaking, the calculation time overhead of our proposed SP-IQGA algorithm is significantly lower than that of the three other heuristic algorithms, and it is less affected by the network size.

For the network topology with N_V=100 and N_E=5, we execute service placement using the five algorithms to obtain their respective optimal deployment schemes: X_{random}, X_{greedy}, X_{anneal}, X_{B+COR}, and X_{IQGA}. We simulate 10 types of application traffics and each application contains 5 data flows. Therefore, a total number of 50 flows are investigated. Suppose that every data flow needs to be processed by middlebox services, and these service requests can be satisfied by a service node. Under the five service
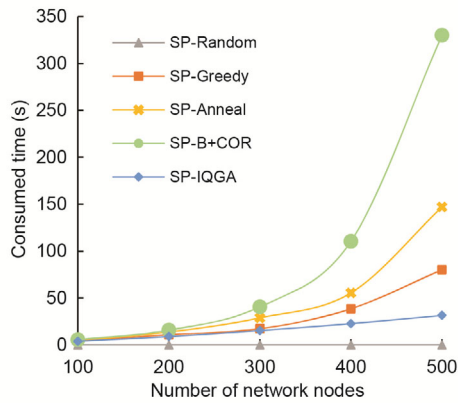
Fig. 8 Comparison of time cost of different algorithms under different network sizes

placement schemes, we let all flows transport the network topology and compute the average transport delay of each traffic for different deployment schemes.

Fig. 9 shows the average transport delay of data flows in each network application under different placement algorithms. The results show that the average transport delay of SP-Random is the highest among all schemes. The delays of SP-Greedy and SP-Anneal are similar and lie in the middle of the algorithms compared. SP-B+COR and SP-IQGA can achieve less transport delay.
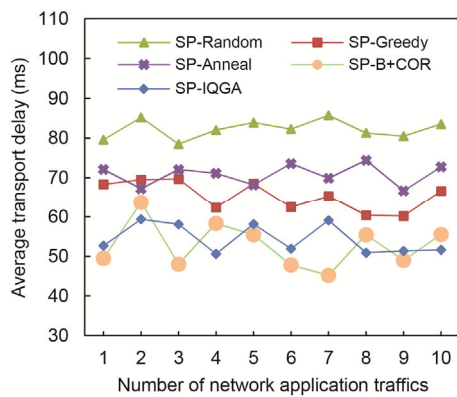

Fig. 9 Comparison of average transport delay under different traffic flows

Fig. 10 shows the transport delay cumulative distribution function (CDF) for all the flows. CDF can show clearly the transmission delay distribution of all applications. The greater the curvature of a CDF curve, the more concentrated the delay value distribution. As shown, the transport delay of each data

flow is distributed in [0, 200]. For 50% of the flows, SP-B+COR and SP-IQGA can achieve about 51 and 55 ms transport delays, respectively, while SP-Greedy, SP-Anneal, and SP-Random need 63, 67, and 80 ms transport delays, respectively. Compared with SP-Random, SP-IQGA can decrease the transport delay by about 30%. Thus, the advantage of SP-IQGA is significant, and it can be used to efficiently solve the service placement problem.
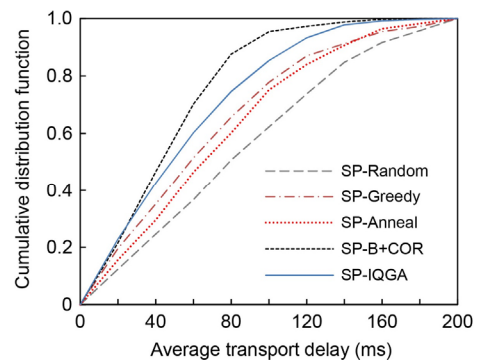

Fig. 10. Distribution of transport delay of different service placement strategies

5 Conclusions

To solve the service deployment problem in the NFV and SDN network environment, we presented a placement approach based on the improved quantum genetic algorithm. Built on top of SDN and the intelligence of QGA, our placement strategy can automatically obtain the optimal schemes for different network topologies. Simulation experiments showed that significant latency reduction can be obtained by our algorithm for placing services in different network topologies. In future work, we will use the proposed approach to help construct network function service chains.

References

Anderson, J.W., Braud, R., Kapoor, R., *et al.*, 2012. xOMB: extensible open middleboxes with commodity servers.

Proc. 8th ACM/IEEE Symp. on Architectures for Networking and Communications Systems, p.49-60. http://dx.doi.org/10.1145/2396556.2396566

Anwer, B., Benson, T., Feamster, N., *et al.*, 2013. A slick control plane for network middleboxes. Proc. 2nd ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking, p.147-148. http://dx.doi.org/10.1145/2491185.2491223

Basta, A., Kellerer, W., Hoffmann, M., *et al.*, 2014. Applying NFV and SDN to LTE mobile core gateways, the functions placement problem. Proc. 4th Workshop on All Things Cellular: Operations, Applications, and Challenges, p.33-38. http://dx.doi.org/10.1145/2627585.2627592

Carpenter, B., Brim, S., 2002. Middleboxes: Taxonomy and Issues, RFC 3234. The Internet Engineering Task Force. Available from http://www.rfc-base.org/rfc-3234.html.

Cheng, G.Z., Chen, H.C., Hu, H.C., *et al.*, 2015. Enabling network function combination via service chain instantiation. *Comput. Netw.*, **92**(Part 2):396-407. http://dx.doi.org/10.1016/j.comnet.2015.09.015

Chiosi, M., Clarke, D., Willis, P., *et al.*, 2012. Network functions virtualisation—introductory white paper. SDN and OpenFlow World Congress. Available from https://portal.etsi.org/NFV/NFV_White_Paper.pdf.

de Turck, F., Boutaba, R., Chemouil, P., *et al.*, 2015. Guest editors' introduction: special issue on efficient management of SDN/NFV-based systems—part I. *IEEE Trans. Netw. Serv. Manag.*, **12**(1):1-3. http://dx.doi.org/10.1109/TNSM.2015.2403775

Fayazbakhsh, S.K., Chaing, L., Sekar, V., *et al.*, 2014. Enforcing network-wide policies in the presence of dynamic middlebox actions using FlowTags. 11th USENIX Symp. on Networked Systems Design and Implementation, p.533-546.

Gember, A., Grandl, R., Anand, A., *et al.*, 2012a. Stratos: virtual middleboxes as first-class entities. Technical Report, No. TR1771, University of Wisconsin-Madison, WI.

Gember, A., Prabhu, P., Ghadiyali, Z., *et al.*, 2012b. Towards software-defined middlebox networking. Proc. 11th ACM Workshop on Hot Topics in Networks, p.7-12. http://dx.doi.org/10.1145/2390231.2390233

Gember, A., Viswanathan, R., Prakash, C., *et al.*, 2014. OpenNF: enabling innovation in network function control. Proc. ACM Conf. on SIGCOMM, p.163-174. http://dx.doi.org/10.1145/2740070.2626313

Greenberg, A., Hjalmtysson, G., Maltz, D.A., *et al.*, 2005. A clean slate 4D approach to network control and management. *ACM SIGCOMM Comput. Commun. Rev.*, **35**(5):41-54. http://dx.doi.org/10.1145/1096536.1096541

Gude, N., Koponen, T., Pettit, J., *et al.*, 2008. NOX: towards an operating system for networks. *ACM SIGCOMM Comput. Commun. Rev.*, **38**(3):105-110. http://dx.doi.org/10.1145/1384609.1384625

Gushchin, A., Walid, A., Tang, A., 2015. Scalable routing in

SDN-enabled networks with consolidated middleboxes. Proc. ACM SIGCOMM Workshop on Hot Topics in Middleboxes and Network Function Virtualization, p.55-60. http://dx.doi.org/10.1145/2785989.2785999

Hwang, J., Ramakrishnan, K.K., Wood, T., 2015. NetVM: high performance and flexible networking using virtualization on commodity platforms. *IEEE Trans. Netw. Serv. Manag.*, **12**(1):34-47. http://dx.doi.org/10.1109/TNSM.2015.2401568

Joseph, D., Stoica, I., 2008. Modeling middleboxes. *IEEE Netw.*, **22**(5):20-25. http://dx.doi.org/10.1109/MNET.2008.4626228

Lange, S., Gebert, S., Zinner, T., *et al.*, 2015. Heuristic approaches to the controller placement problem in large scale SDN networks. *IEEE Trans. Netw. Serv. Manag.*, **12**(1):4-17. http://dx.doi.org/10.1109/TNSM.2015.2402432

Li, Y., Chen, M., 2015. Software-defined network function virtualization: a survey. *IEEE Access*, **3**:2542-2553. http://dx.doi.org/10.1109/ACCESS.2015.2499271

Lu, B., Chen, J.Y., Cui, H.Y., *et al.*, 2013. A virtual network mapping algorithm based on integer programming. *J. Zhejiang Univ.-Sci. C (Comput. & Electron.)*, **14**(12): 899-908. http://dx.doi.org/10.1631/jzus.C1300120

Malossini, A., Blanzieri, E., Calarco., T., 2008. Quantum genetic optimization. *IEEE Trans. Evol. Comput.*, **12**(2): 231-241. http://dx.doi.org/10.1109/TEVC.2007.905006

Matias, J., Garay, J., Toledo, N., *et al.*, 2015. Toward an SDN-enabled NFV architecture. *IEEE Commun. Mag.*, **53**(4):187-193. http://dx.doi.org/10.1109/MCOM.2015.7081093

McKeown, N., Anderson, T., Balakrishnan, H., *et al.*, 2008. OpenFlow: enabling innovation in campus networks. *ACM SIGCOMM Comput. Commun. Rev.*, **38**(2):69-74. http://dx.doi.org/10.1145/1355734.1355746

Mohammadkhan, A., Ghapani, S., Liu, G.Y., *et al.*, 2015. Virtual function placement and traffic steering in flexible and dynamic software defined networks. IEEE Int. Workshop on Local and Metropolitan Area Networks, p.1-6. http://dx.doi.org/10.1109/LANMAN.2015.7114738

Mohammed, A.M., Elhefnawy, N.A., El-Sherbiny, M.M., *et al.*, 2012. Quantum crossover based quantum genetic algorithm for solving non-linear programming. 8th Int. Conf. on Informatics and Systems, p.BIO-145-BIO-153.

Nunes, B.A.A., Mendonca, M., Nguyen, X.N., *et al.*, 2014. A survey of software-defined networking: past, present, and future of programmable networks. *IEEE Commun. Surv. Tutor.*, **16**(3):1617-1634. http://dx.doi.org/10.1109/SURV.2014.012214.00180

Open Networking Foundation (ONF), 2012. Software-Defined Networking: the New Norm for Networks. ONF White Paper.

Qazi, Z.A., Tu, C.C., Chiang, L., *et al.*, 2013. SIMPLE-fying middlebox policy enforcement using SDN. Proc. ACM SIGCOMM Conf., p.27-38. http://dx.doi.org/10.1145/2486001.2486022

Qi, H., Shiraz, M., Liu, J.Y., *et al.*, 2014. Data center network architecture in cloud computing: review, taxonomy, and open research issues. *J. Zhejiang Univ.-Sci. C (Comput. & Electron.)*, **15**(9):776-793.
http://dx.doi.org/10.1631/jzus.C1400013

Rajagopalan, S., Williams, D., Jamjoom, H., *et al.*, 2013. Split/Merge: system support for elastic execution in virtual middleboxes. 10th USENIX Symp. on Networked Systems Design and Implementation, p.227-240.

Sekar, V., Ratnasamy, S., Reiter, M.K., *et al.*, 2011. The middlebox manifesto: enabling innovation in middlebox deployment. Proc. 10th ACM Workshop on Hot Topics in Networks, p.1-6.
http://dx.doi.org/10.1145/2070562.2070583

Sekar, V., Egi, N., Ratnasamy, S., *et al.*, 2012. Design and implementation of a consolidated middlebox architecture. Proc. 9th USENIX Conf. on Networked Systems Design and Implementation, p.323-336.

Shen, J., He, W.B., Liu, X., *et al.*, 2015. End-to-end delay analysis for networked systems. *Front. Inform. Technol. Electron. Eng.*, **16**(9):732-743.
http://dx.doi.org/10.1631/FITEE.1400414

Sherry, J., Hasan, S., Scott, C., *et al.*, 2012. Making middleboxes someone else's problem: network processing as a cloud service. *ACM SIGCOMM Comput. Commun. Rev.*, **42**(4):13-24. http://dx.doi.org/10.1145/2377677.2377680

Walfish, M., Stribling, J., Krohn, M., *et al.*, 2004. Middleboxes no longer considered harmful. Proc. 6th Symp. on Operating Systems Design & Implementation, p.215-230.

Zegura, E.W., Calvert, K.L., Bhattacharjee, S., 1996. How to model an internetwork. 15th Annual Joint Conf. of the IEEE Computer and Communications Societies, p.594-602.
http://dx.doi.org/10.1109/INFCOM.1996.493353

Zhang, Y., Beheshti, N., Beliveau, L., *et al.*, 2013. StEERING: a software-defined networking for inline service chaining. Proc. 21st IEEE Int. Conf. on Network Protocols, p.1-10.
http://dx.doi.org/10.1109/ICNP.2013.6733615