Abstract

Network services are an important component of the Internet, which are used to expand network functions for third-party developers. Network function virtualization (NFV) can improve the speed and flexibility of network service deployment. However, with the evolution of the network, network service layout may become inefficient. Regarding this problem, this paper proposes a service chain migration (SCM) method with the framework of "software defined network + network function virtualization" (SDN+NFV), which migrates service chains to adapt to network evolution and improves the efficiency of the network service layout. SCM is modeled as an integer linear programming problem and resolved via particle swarm optimization. An SCM prototype system is designed based on an SDN controller. Experiments demonstrate that SCM could reduce the network traffic cost and energy consumption efficiently.

Introduction

Middleboxes [1], hardware-based network services, are widely deployed in the Internet and being recognized as important components of networks, such as their uses as firewalls, intrusion detection systems/intrusion prevention systems (IDS/IPS), load balancers, agencies, network address translators (NAT), and wide area network (WAN) optimizers etc. Service chains [2] are created by combining these service instances according to network policies and user requirements. Service chains satisfy various needs of users and provide value-added services to networks.

The flexibility and expansibility of the service deployment can be improved by network function virtualization (NFV) [3–5]. In NFV, special hardware middleboxes are replaced by virtual services, thus service chain customization and dynamical creation can be achieved. In addition, software defined networks (SDN) [6–8] are widely used to orchestrate network services according to policies and control traffic to pass special service chains [9–12]. The "SDN +NFV" network paradigm [13] provides a flexible, scalable, and adaptable architecture for deploying virtual services and creates new opportunities for service chain management.

Based on the "SDN+NFV" architecture, there has been extensive researches about service chain deployment [12, 14–17]. In these studies, service chains are deployed with consideration...
of the quality of service, resource allocation, and network security. However, these service chain placement approaches only consider the current network static status, and the changes of the network state are neglected. With network evolution, some old flows may disappear and new flows may arise. Therefore, the status of network changes and the service chain deployment may lose optimality with network evolution, resulting in a waste of network resources and energy. Moreover, the quality of services will decrease due to the degraded service chain layout.

This problem is difficult to resolve in service chain deployment due to the limited ability to anticipate the evolution of network states. However, if the deployment of service chains is dynamic throughout their life cycles, the network resource allocation can be adjusted dynamically to adapt to the changing network status. In this paper, a service chain migration (SCM) framework is proposed to address this problem, in which service chains are migrated dynamically to adapt to network evolution. SCM satisfies policy demands and improves the effectiveness of the network service instance layout. We model SCM as an integer linear programming (ILP) problem, and particle swarm optimization (PSO) [18] is used to solve the SCM problem.

The contributions of this work are as follows.

1. Two scenarios are shown to illustrate that the efficiency of the network resource layout decreases with network evolution and that service chain migration is utilized to improve the situation.

2. An SCM framework is proposed to optimize the network resource layout. We model the SCM as an ILP problem, and a modified PSO algorithm is implemented to provide the optimal solution under certain network resource constraints.

3. An SCM prototype system is designed based on an SDN controller [19, 20], and the efficiency of the SCM is corroborated by several numerical simulations.

Related work

In traditional networks, services are realized by middleboxes, which is a major way for third-party developers to expand network functions. Extensive research has been conducted in middlebox management. StEERING [12], a flexible and extensible middlebox management framework, can efficiently control network traffic to pass middleboxes. SIMPLE [14], a service policy execution layer, can instantiate network service policies into service chains, where load balance and costs of network are considered in the service chain construction. Stratos [9], a network service orchestration framework, considers the changes of network load, and the service chains are constructed efficiently. The above three kinds of frameworks put effort into the service chains to optimize the traffic or load balance. However, the network evolution is not considered, which may limit the effectiveness of the network. Similar to the work above, SCM is also designed to manage network service, but the goal of SCM is to optimize service layouts for network evolution. SCM dynamically adjusts the deployed service chains, and the network resource allocation is optimized according to network state. In addition, more benefits can be obtained in SCM, such as traffic optimization, energy saving and lower network cost.

Virtual machine migration has been proven to have the ability to increase the resource utilization rate and reduce energy consumption. A dynamic virtual machine migration method is proposed in [21], where migration traffic, available bandwidth, and balance of bandwidth capacity are considered during migration. In [22], virtual machine migration and network routing optimization are utilized to reduce the network energy consumption in the data center. Nidhi et al. [23] proposed an energy-aware virtual machine migration method in a cloud
environment to improve the efficiency and decrease the energy consumption of the network. For energy saving, a novel placement selection policy of live virtual machine (VM) migration named PS-ES [24] is proposed. This method combines particle swarm optimization algorithm and simulated annealing to obtain the placement selection policy of the live VM migration. From the point of VM controller, Wen and et al. propose LS-STR [25], an adaptive controller for data center, based on the least square self-tuning regulator. The method can adjust VM resources dynamically and reduce the energy cost. Instead of migrating a single virtual machine or service, as described in the studies above, we migrate service chains to further optimize traffic cost and energy consumption. More specifically, SCM migrates multiple virtual machines periodically and the best migration strategy is determined by solving a ILP problem.

**Motivating scenarios**

For convenience, the abbreviations used throughout this paper are listed in Table 1.

Since the demands of users are dynamic, new policies are added in the network and outdated policies are deleted. With the network evolution, the fixed layout of service instances may lower the utilization efficiency of network resources. However, by migrating service chains, the efficiency of resource layout may be improved. In this paper, two scenarios are shown where service chain migration can reduce traffic cost and energy consumption, respectively.

1. **To optimize service layout and reduce traffic cost**: The network state evolution may lower the efficiency of the service layout. As shown in Fig 1(A), there are two service instances \( e_1 \) and \( e_2 \) in the network, and the traffic cost on each link is shown in the figure. Two policies are deployed successively in the network. Policy 1: The communication between hosts \( h_1 \) and \( h_2 \) needs two service instances \( e_1 \) and \( e_2 \). The traffic cost is given by \( \text{Cost}(h_1, h_2) = 9 \). Policy 2: The hosts \( h_3 \) and \( h_4 \) need service instance \( e_2 \) during communication, and the traffic cost is \( \text{Cost}(h_3, h_4) = 19 \), which is larger. If the service instance \( e_2 \) is migrated to the server connected with \( s_7 \), the service chains will be migrated accordingly. The paths \( h_1 \rightarrow h_2 \) and \( h_3 \rightarrow h_4 \) will be adjusted as shown in Fig 1(B). In this circumstance, \( \text{Cost}(h_1, h_2) = 10 \) and \( \text{Cost}(h_3, h_4) = 11 \), and the total traffic cost will be decreased.

2. **To reduce the number of running servers and save energy**: As the network evolves over time, one server may carry only one or very few running service instances, which leads to a waste of energy. As shown in Fig 2(A), the current routing paths \( h_1 \rightarrow h_2 \) and \( h_1 \rightarrow h_4 \) are marked by dashed arrows, and both policies need the two service instances \( e_1 \) and \( e_2 \). The server \( \text{server}_1 \) carries only one service \( e_1 \), while \( \text{server}_2 \) still has sufficient resource to hold

| Abbreviation | Full name                        |
|--------------|----------------------------------|
| NFV          | Network Function Virtualization  |
| SDN          | Software Defined Network         |
| SCM          | Service Chain Migration          |
| VM           | Virtual Machine                  |
| ILP          | Integer Linear Programming       |
| PSO          | Particle Swarm Optimization      |
| OF           | OpenFlow                         |
| EA           | Evolutionary Algorithm           |

Table 1. Abbreviations.

[https://doi.org/10.1371/journal.pone.0189336.t001](https://doi.org/10.1371/journal.pone.0189336.t001)
one more service instance. Obviously, energy consumption will be reduced if we migrate service $e_1$ to $server_2$ (Fig 2(B)) and turn off $server_1$.

**Modeling**

In order to solve the problem of low efficiency of the resource layout caused by network evolution, service chains are considered to be migrated periodically under constraints. Service chain migration means migrating service instances in server chains from a server to another position without violating the policy demand. We first present relevant definitions, and then the service chain migration model is proposed. Finally, we present an intelligent optimization algorithm to solve the model.
Basic definitions

**Service instance**: A service instance is an instantiated service equipment, which can be represented as a 3-unit set: \( e = <ID, T, V> \), where \( ID \) is the unique identification of a service instance, \( T \) is the type of service, and \( V \) is the amount of data to be transferred for migrating this service instance. Let \( E = \{e_1, e_2, \cdots, e_n\} \) denote the set of all service instances deployed in the network.

**Service chain**: A service chain is a group of sequential service instances, which can satisfy user’s specific demands for communication. Let \( \mathcal{E}_p = \{e_1, \cdots, e_j, \cdots, e_j\} \) denote a service chain.

**Policy**: A policy is defined as \( p \equiv (O_p, \mathcal{E}_p, D_p) \), where \( O_p \) denotes the policy target, i.e., the specific traffic processed by policy \( p \), \( \mathcal{E}_p \) denotes the service chain that processes traffic \( O_p \). The number of service instances in \( \mathcal{E}_p \) is denoted by \( |p| \). And \( D_p \equiv (d_p^0, d_p^1) \) is the demand set of policy \( p \), where \( d_p^0 \) denotes the bandwidth demand, and \( d_p^1 \) denotes the path cost demand.

**Server**: A server is a running platform for virtual service instances, which can provide storage and bandwidth resources for multiple service instances. Let \( Z = \{z_1, z_2, \cdots, z_k\} \) be the set of all servers in the network. For \( z_i \in Z, w_i \) denotes the maximum number of service instances that \( z_i \) can carry and \( t_i \) denotes the maximum bandwidth that \( z_i \) can process.

SCM model

Compared with traditional virtual machine migration, the SCM model is more complex for two reasons. First, policy demands, such as the bandwidth and path cost, are taken into account in SCM. Second, the migration of one service instance in a service chain may affect the migration of other service instances in this service chain because of the correlations between them. We intent to get the best migration strategy under multiple factors to minimize the network cost. We define a utility value as the measurement of network cost of a migration and intent to get the minimum utility value of a migration under certain constraints, therefore, it can be regarded as a minimum value problem. As described below, the objective function and constraints are considered to be linear in our study. In addition, the solutions of SCM belong to the integer domain. Therefore, we modeled SCM as a problem of ILP.

For a migration \( M \), the optimization goal of SCM is shown in Eq (1):

\[
SCM(M) \quad \text{min} \quad Utility = x_1 \beta_1 U + x_2 \beta_2 R + x_3 \beta_3 Q
\]

\[
x_1 + x_2 + x_3 = 1
\]

where \( Utility \) is the utility value of migration \( M \); \( U \) is the traffic cost of the whole network policies; \( R \) is the number of running servers; \( Q \) is the migration cost of the service instance; \( \alpha_1, \alpha_2, \alpha_3 \) are the weights of \( U, R, \) and \( Q \), respectively, whose total sum is 1; and \( \beta_1, \beta_2, \beta_3 \) are the parameters that unify the dimensions of \( U, R \) and \( Q \).

\[
U = \sum_{p \in P} \sum_{j \in [0, |p|]} \sum_{z_i \in Z} y_{i,j} \text{Cost}(s', s)f(p, i, j)
\]

\[
e_r = \mathcal{E}_p[j]; e_{r+1} = \mathcal{E}_p[j + 1]
\]

The whole network traffic cost is calculated by Eq (3). The boolean variable \( y_{i,j} \) indicates whether the service instance \( e_r \) is on server \( z_i \) after migration. \( \text{Cost}(s', s) \) is the minimum path cost between server \( z_j \) and \( z_i \), \( f(p, i, j) \) is the traffic injecting into the subsequent service \( e_r \) from service \( e_r \), \( \mathcal{E}_p[0] \) denotes the traffic source of policy \( p \). The traffic from the source to the first
service is the bandwidth demand $d^o_p$:

$$Q = \sum \sum \sum \ w_i \ x'_i \ m'_i$$  \hspace{1cm} (4)$$

The migration of a service instance consumes network bandwidth resources. Eq (4) shows the bandwidth consumption caused by a service instance migration. The boolean variable $x'_i$ indicates whether the service instance $e_i$ is on server $z_s$ in the original network. The boolean variable $m'_i$ indicates whether $e_i$ is migrating to $z_p$. $w_i$ is the volume of $e_i$, and $l_{i,j}$ is the shortest distance between server $z_s$ and $z_p$.

A server with no service running on it will be turned off to reduce energy consumption. Eq (5) gives the number of running servers after migration, where $y'_i$ indicates whether $e_i$ is on $z_s$ after migration.

$$R = \sum \sum \left(1 - \prod \left(1 - y'_i\right)\right)$$  \hspace{1cm} (5)$$

Based on the flow conservation principle, Eq (6) declares that the outflow of one service is equal to the inflow of the subsequent service. Network services may affect traffic (for example, a firewall may drop a part of traffic in a flow). In SCM, traffic influence factors are introduced to model the traffic influence caused by services. Let $\gamma'_p$ be the traffic influence on the $j$-th service of policy $p$, which is calculated by $\gamma'_p = \text{outflow/inflow}$. For policy $p$, the traffic from the source to the first service is the bandwidth demand $d^o_p$, as shown in Eq (7).

$$f(p, i, i') = f(p, i', i) \gamma'_p$$  \hspace{1cm} (6)$$

$$j = 1, f(p, \mathcal{E}_p[j-1], \mathcal{E}_p[j]) = d^o_p$$  \hspace{1cm} (7)$$

Eq (8) shows the relationship between $x'_i$, $y'_i$, $m'_i$, where $x'_i \left(1 - \sum_{z_s \in Z \setminus \{s\}} m'_i\right) = 1$ indicates the service instance $e_i$ is running on the server $z_s$ and has not been migrated; $(1 - x'_i)m'_i = 1$ indicates the service instance $e_i$ is not running on $z_s$ originally, but has been migrated to $z_p$. $x'_i$ and $m'_i$ cannot be 1 at the same time. Eq (9) adds the constraint that the service instance $e_i$ cannot be migrated to server $z_s$ from $z_s$ itself.

$$\forall e_i \in E, z_s \in Z, \ y'_i = x'_i \left(1 - \sum_{z_s \in Z \setminus \{s\}} m'_i\right) + (1 - x'_i)m'_i$$  \hspace{1cm} (8)$$

$$\forall e_i \in E, z_s \in Z, \ x'_im'_i = 0$$  \hspace{1cm} (9)$$

Policy $p$ demands that the cost from the source to the destination should not exceed $d^o_p$. Eq (10) indicates that the service chain should satisfy the cost demand after migration.

$$\forall p \in P, \ \sum \sum \sum \ y'_i y'_j \ \text{Cost}(s, s') \leq d^o_p$$  \hspace{1cm} (10)$$

For $\forall z_s \in Z$, its bandwidth is finite. Eq (11) indicates that the traffic injected from the
network to the server $z_i$ should not exceed its maximum bandwidth $t_r$,

$$\forall z_i \in Z, \sum_{p \in P} \sum_{j \in [0,|p|]} y_{j}^{f}(p, i', i) \leq t_r$$  \hspace{1cm} (11)

$$e_i = e_{p[j]}; e_{r} = e_{p[j-1]}$$

For $\forall z_i \in Z$, the number of service instances running on the server $z_i$ is restrained by the capacity of $z_i$, as shown in Eq (12).

$$\forall z_i \in Z, \sum_{i \in E} y_{i}^{f} \leq w_i$$  \hspace{1cm} (12)

For one migration, one service instance can be migrated to no more than one server, which is indicated by Eq (13). Eq (14) shows the range of the variable $m_i^{e}$.

$$\forall e_i \in E, \sum_{z_i \in Z} m_{i}^{e} \leq 1$$  \hspace{1cm} (13)

$$\forall e_i \in E, z_i \in Z, m_{i}^{e} \in \{0, 1\}$$  \hspace{1cm} (14)

**Resolving algorithm**

SCM($M$) is a typical ILP problem, which is called the SCM problem in this paper. An exhaustive algorithm can be used to solve the SCM problem by traversing the solution space. However, the time consumption will be huge due to the large size of this problem. An alternative would be the heuristic search approach, such as the evolutionary algorithm (EA). EA has been studied extensively [26–28] and has shown to be suitable for solving real-world optimization problems with large solution space, containing multiple fields [29–32]. We use PSO algorithm, a particularly interesting algorithm of EA family, to solve the SCM problem due to its advantage in convergence rate. Here, we adopt the binary PSO to solve the SCM problem as the solution space of the SCM problem is binary. The position and velocity parameters of the binary PSO are defined as follows.

**Position ($M$):** The boolean variable $m_{i}^{e}$ denotes whether service instance $e_i$ is migrating to server $z_i$ or not; the vector $(m_{1}^{e}, m_{2}^{e}, \ldots, m_{Z}^{e})$ indicates the migration of $e_i$ ($e_i \in E$). $M_g$ is position vector of the $g$-th particle in a particle swarm, which is defined as one migration of all service instances as shown in Eq (15), where $|Z|$ is the number of servers and $|E|$ is the number of service instances. Let $m_{g,d}$ be the $d$-th component of $M_g$.

$$M_g = (m_{1}^{e}, m_{2}^{e}, \ldots, m_{1}^{Z}, \ldots, m_{1}^{e}, m_{2}^{e}, \ldots, m_{1}^{Z}, \ldots, m_{1}^{e}, m_{2}^{e}, \ldots, m_{1}^{Z})$$  \hspace{1cm} (15)

**Velocity ($V$):** The velocity vector $(v_{1}^{e}, v_{2}^{e}, \ldots, v_{Z}^{e})$ is used to migrate a service instance $e_i$ into a better solution. In Eq (16), the vector $V_g$ is the velocity of the $g$-th particle in the particle swarm. Based on the component $v_{i}^{e}$ in $V_g$, the probability of component $m_{i}^{e}$ in $M_g$ being 1 can be determined. Let $v_{g,d}$ be the $d$-th component of the velocity vector $V_g$.

$$V_g = (v_{1}^{e}, v_{2}^{e}, \ldots, v_{1}^{Z}, \ldots, v_{1}^{e}, v_{2}^{e}, \ldots, v_{1}^{Z}, \ldots, v_{1}^{e}, v_{2}^{e}, \ldots, v_{1}^{Z})$$  \hspace{1cm} (16)

In the traditional binary PSO, the particle velocity and position updating equations are shown in Eqs (17)–(20). In Eq (17), $c_1$ and $c_2$ are constants, $\xi$ and $\eta$ are two independent random variables with uniform distribution on the interval of [0,1], $pBest_{g,d}$ is the $d$-th component in the historical optimal solution $pBest_g$ of particle $g$ (i.e. the local optimal solution), and $gBest_d$ is
the $d$-th component of the optimal solution $g$Best among all particles (i.e. the global optimum solution). $\omega(t)$ is the inertia weight in PSO. $m_{gd}$ is the $d$-th component of $M_g$ and $v_{gd}$ is the $d$-th component of $V_g$. In Eq (18), the random variable random has uniform distribution on the interval of $[0,1]$. In Eq (20), $\omega_{\text{max}}$ and $\omega_{\text{min}}$ are the maximum and minimum values of $\omega(t)$, respectively. $t$ is the current number of iterations, and $MI$ is the maximum number of iterations.

\begin{equation}
 v_{gd} = \omega(t)v_{gd} + c_1\zeta(p\text{Best}_{gd} - m_{gd}) + c_2\eta(g\text{Best}_d - m_{gd})
\end{equation}

\begin{equation}
 m_{gd} = \begin{cases} 
 1, & \text{if random} < \text{Sig}(v_{gd}) \\
 0, & \text{else}
\end{cases}
\end{equation}

\begin{equation}
 \text{Sig}(v_{gd}) = \frac{1}{1 + \exp(-v_{gd})}
\end{equation}

\begin{equation}
 \omega(t) = \omega_{\text{max}} - \frac{\omega_{\text{max}} - \omega_{\text{min}}}{MI} \times t
\end{equation}

From Eq (18), it can be known that the value of boolean variable $m_{gd}$ have randomness. Therefore, the position updating of one particle may cause one service instance to be migrated to multiple servers, which violates the constraint in Eq (13). Thus, it is difficult to find feasible solutions. To solve this problem, the position updating equation Eq (18) is modified with consideration to the constraint in Eq (13).

In the velocity vector $V_g$, the $|Z|$ components $(v_1, v_2, \cdots, v^{|Z|})$ represent the migration of the service instance $e_i$, and $\text{Sig}(v')$ represents the probability that $m_{it}$ is 1 among the $|Z|$ components $(m_{i1}, m_{i2}, \cdots, m_{i|Z|})$ in the position vector $M_g$. Let:

\begin{equation}
 \lambda_i' = \text{Sig}(v_i')
\end{equation}

\begin{equation}
 \lambda_i = (\lambda_{i1}', \lambda_{i2}', \cdots, \lambda_{i|Z|}')
\end{equation}

Eq (22) represents the probabilities that a service instance $e_i$ is migrated to servers in $Z$. Since a service instance $e_i$ can be migrated to only one server, there is at most one component $m_{it}$ in $(m_{i1}, m_{i2}, \cdots, m_{i|Z|})$ with a value of 1 ($1 \leq s \leq |Z|$). If one component $m_{is}$ is 1, its probability is shown Eq (23).

\begin{equation}
 \lambda_{is}^{\text{mig}(s)} = (1 - \lambda_{i1}') \times (1 - \lambda_{i2}') \times \cdots \times (1 - \lambda_{i(s-1)}') \times \lambda_{is}' \times (1 - \lambda_{i(s+1)}') \times (1 - \lambda_{i|Z|}')
\end{equation}

\begin{equation}
 = \frac{\lambda_{is}'}{1 - \lambda_{i|Z|}' \prod_{k=1, |Z|} (1 - \lambda_{ik})}
\end{equation}

If no component in $(m_{i1}', m_{i2}', \cdots, m_{i|Z|}')$ is equal to 1, i.e., all the components are 0, then the probability is marked as $\lambda_{is}^{\text{mig}(0)}$ and calculated by Eq (24).

\begin{equation}
 \lambda_{is}^{\text{mig}(0)} = (1 - \lambda_{i1}') \times (1 - \lambda_{i2}') \times \cdots \times (1 - \lambda_{i(s-1)}') \times (1 - \lambda_{is}') \times (1 - \lambda_{i(s+1)}') \times (1 - \lambda_{i|Z|}')
\end{equation}

\begin{equation}
 = \prod_{k=1, |Z|} (1 - \lambda_{ik}')
\end{equation}

There are $|Z| + 1$ cases for updating the position vector $(m_{i1}', m_{i2}', \cdots, m_{i|Z|}')$ under the constraint in Eq (13): one component is 1 while the other components are zero ($|Z|$ cases) and all
the components are 0 (1 case). The probability of each case can be calculated by Eq (25). Obviously, \( \sum_{k \in [0, |Z|]} \theta_k^i = 1. \)

\[ \forall k \in [0, |Z|], \quad \theta_k^i = \lambda_i^{poly(k)} \left( \sum_{\ell \in [0, |Z|]} \lambda_i^{poly(\ell)} \right)^{-1} \tag{25} \]

A probabilistic search procedure [33] is used to select \( \theta_k^i \). If \( k = 0 \), then all components in \((m_1^i, m_2^i, \cdots, m_Z^i)\) are 0. If \( 1 \leq k \leq |Z| \), then \( m_k^i = 1 \), while the other \(|Z| - 1\) components in \((m_1^i, m_2^i, \cdots, m_Z^i)\) are all 0.

For \( i \in [1, |E|] \), \( m_k^i = \begin{cases} 1, & \text{if } \sum_{\ell \in [0, k-1]} \theta_{\ell i} \leq rd \leq \sum_{\ell \in [0, k]} \theta_{\ell i}, \quad 1 \leq k \leq |Z| \\ 0, & \text{otherwise} \end{cases} \tag{26} \]

The position updating of particle \( g \) is conducted in groups. First, the components of the position vector \( M_g \) are divided into \(|E|\) groups. Each group \((m_1^i, m_2^i, \cdots, m_Z^i)\) corresponds to the migration of one service instance \( e_i \). \((m_1^i, m_2^i, \cdots, m_Z^i)\) is updated based on Eq (26) (\( rd \) has uniform distribution on the interval of [0, 1]). With this method, Eq (13) can be solved after the particle position vector is updated and the optimizing speed is improved by reducing the number of infeasible solutions.

To reduce the number of running servers, the service instances in small-load servers should be migrated with a high probability. Thus, Eq (21) is modified into Eq (27) by adding an encouragement factor, where Threshold is the server load threshold, \( Enc(t) \) is an encouragement function, as showed in Eq (28).

\[ \lambda_i^t = \begin{cases} \text{Sig}(v_i^t) + Enc(t)(1 - \text{Sig}(v_i^t)), & \text{when load of the server which } e_i \\
\text{Sig}(v_i^t), & \text{is located is lower than Threshold}, \\
\text{Sig}(v_i^t), & \text{and load of } z_i \text{ is more than Threshold} \\
\text{Enc}(t) = \frac{1}{\theta e - 1} \left( (\theta e)^m - 1 \right), t \in [1, MI] \end{cases} \tag{27} \]

In Eq (28), \( Enc(t) \) changes with the number of iterations \( t \) and the parameter \( \theta \), as shown in Fig 3. We set \( \theta = 1000 \). As shown in the figure, when the number of iterations is small, the instance migration is not encouraged, which enables PSO to escape from local optima and converge to the global optimum. When the number of iterations is large, it intends to migrate the instances in small-load servers to reduce the number of running servers.

To solve the SCM problem, Eq (1) is used as the fitness function in the PSO, marked as \( \text{fit}(M) \), where \( M \) indicates a possible migration. The constraints will be checked before computing the fitness function. If a migration \( M \) satisfies the constraints, \( \text{fit}(M) \) is computed; otherwise, \( \text{fit}(M) = +\infty \).

**Design of SCM Framework and Experiments**

In this section, a design of the SCM framework is elaborated. Then, three metrics are proposed to evaluate the performance and cost of SCM. Finally, the performance and cost of SCM are compared and analyzed under different parameters.
SCM framework

We designed an SCM prototype system FlowMover based on the SDN controller. As shown in Fig 4, FlowMover consists of three planes: a policy plane, a control plane, and a data plane. The policy plane and the control plane are constructed based on the SDN controller. The Policy manager of the policy plane sends policies to the network. The control plane deploys the service instances according to the service chain policies and network state, and installs flows through the Fow manager (via the southbound API). The FlowMover in the control plane migrates the service chains periodically through service chain interface API (SC API).

In FlowMover, the traffic tag method proposed in [14] is applied. Specific traffic is tagged to pass a certain service chain based on the policy. There are many methods available for online virtual machine migration without interrupting services [34–36] and the virtual machine migration method proposed in [34] is adopted in SCM.

Evaluation metrics

To evaluate the effectiveness and cost of SCM, we define three evaluation metrics: optimized rate of traffic cost $OE_U$, optimized rate of number of running servers $OE_R$, and migration rate of service instance $OE_Q$, which can be calculated according to Eqs (29), (30) and (31), respectively. $U$ and $U'$ denote the whole network traffic cost before and after service chain migration, $R$ and $R'$ denote the numbers of running servers in the network before and after service chain migration, and $Q$ and $Q'$ denote the numbers of all and migrated service instances in one
Performance and cost assessment

Simulations of network evolution and service chain migration are implemented in MATLAB. BRITE [37], a network topology generator, is used in the experiments to generate a random topology with 100 nodes of Waxman model ($\alpha = 0.2$, $\beta = 0.15$). In the experiment topology, the delay of each link in is considered as the cost of the link. Each network node is connected with a server, and all servers have the same capacity $w_s$ and bandwidth resource $t_s$. $t_s$ is set to 50 unit traffic per unit time, and the threshold $\text{Threshold}$ is set to $[w_s/3]$. The length of the service chain of policies has uniform distribution on the interval of [1,15], while the bandwidth demand of the policies has uniform distribution on the interval of [1,5]. The policy request follows a Poisson distribution with an average of 4 requests in 100 time units. The policy life cycle follows an exponential distribution with an expectation of 1000 time units. In the optimization goal of Eq (1), $\beta_1 = 1/U'$, $\beta_2 = 1/R'$, $\beta_3 = 1/Q'$, where $U'$ and $R'$ denote the traffic cost and the number of running servers before service chain migration. $Q'$ represents the summation of service instance volumes multiplied by the average link length. For the service $p[j]$ in the policy $p$, we set $\gamma_p^{p[j]} = 1$.

To simulate the evolution of the network state, the service chain deployment algorithm proposed in [38] is used in the experiments, and the service chains are deleted from the network.

\[
OE_U = \frac{U'}{U} \tag{29}
\]

\[
OE_R = \frac{R'}{R} \tag{30}
\]

\[
OE_Q = \frac{Q'}{Q} \tag{31}
\]
when policies are outdated. The minimum cost between service instances is calculated by Dijkstra algorithm. The parameters setted in PSO are listed in Table 2. The simulation program runs on a computing platform with Intel Core i7 @3.10 GHz and 8G RAM.

### 1) Performance

In this section, service chain deployment methods, SCI [38] and Stratos [9], are compared with SCM in terms of traffic cost and number of running servers. In the experiments, the migration period is set to 500 time units, and the policy cost demand \( d^p_p \) is defined as the maximum path cost in the network. We set \( \alpha_1 = 0.5 \), \( \alpha_1 = 0.3 \), and \( \alpha_1 = 0.2 \). The comparison of the traffic cost of the three methods is shown in Fig 5(A). In all three cases, the traffic cost of network increases in the beginning, which can be explained by the increasing number of the policies deployed in the network. We can see that the traffic cost of SCI is the highest among the three methods, since the traffic cost is not considered in SCI. By contrast, Stratos minimizes the traffic cost when server chains are deployed and the traffic cost is decreased effectively. However, in Stratos, the server chains cannot be moved once they are deployed. The layout of server chains will degenerate with the evolution of the network. Yet, SCM moves the server chains along with the evolution of the network and keeps low traffic cost. Therefore, the traffic cost of SCM is the lowest, about 20% lower than SCI and 10% lower than Stratos on average. The number of running servers with time for the three methods are shown in Fig 5(B). SCM has the lowest number of running servers. This is because SCM moves the server chains to reduce the number of running servers for energy saving. As can be seen, almost all the servers are running after 3000 time units when SCI or Stratos is used. The number of running servers is decreased by about 15% in SCM.

### 2) Optimization of traffic cost

The service chains are periodically migrated to optimize the layout of service instances. The migration period is 500 time units. The policy cost demand \( d^p_p \) is defined as the maximum path cost in the network. Fig 6 shows the curves of \( OE_U \) with respect to time and different server capacities. With the network evolution, the number of policies deployed in the network rises at first, and then keeps stable. At the initial period of the network (Fig 6(A)), the ability of the traffic cost optimization is limited, and \( OE_U \) is close to 1 due to the small number of deployed policies. \( OE_U \) decreases with the increase of the number of policies in the network. This is because SCM migrates the service chains along with network evolution to decrease the traffic cost. The higher the weight of traffic cost \( \alpha_1 \) in the utility function (Eq (1)) is, the more significantly SCM optimizes the traffic cost. Therefore, \( OE_U \) is minimized when \( \alpha_1 = 1 \) and maximized when \( \alpha_1 = 0.2 \). As shown in Fig 6(A) and 6(B), when the weights are the same, \( OE_U \) at \( w_s = 10 \) is smaller than that when \( w_s = 5 \). The reason is that when the capacity of the server is enhanced, there are more resources in the network for server chain migration. Thus, better results can be achieved. When \( w_s = 15 \), there are a lot of idle server capacity resources. The layout of the service instances is already optimized when deploying service instances; therefore, \( OE_U \) is not significantly reduced compared with that when \( w_s = 10 \).

### Table 2. PSO parameter setting in the simulation.

| Parameter                  | Value   |
|----------------------------|---------|
| Number of particles \( N_p \) | 30      |
| Cognitive scaling parameter \( c_1 \) | 2       |
| Social scaling parameter \( c_2 \) | 2       |
| \( \xi, \eta \)              | U(0,1)  |
| Maximum inertia weight \( \omega_{max} \) | 0.9     |
| Minimum inertia weight \( \omega_{min} \) | 0.4     |
| Maximum number of iterations \( M \) | 1000    |

https://doi.org/10.1371/journal.pone.0189336.t002
3) Optimization of energy consumption. The optimized rate of number of running servers \(OE_R\) is used to evaluate the optimization of energy consumption. When there is no service instance running on a server, this server will be turned off to save energy. The service chains are migrated with a period of 500 time units and the demand of cost \(d_p\) is defined as the maximum path cost in the network. Fig 7 shows the curves of \(OE_R\) under different server loads. As shown in Fig 7(A), at the initial period, the number of deployed policies is small, and the server resources are sufficient. Therefore, the value of \(OE_R\) is low. With the increase of the number of policies, \(OE_R\) increases, as it becomes difficult to gather all the service instances in a small number of servers. The higher the weight \(\alpha_2\) in the utility function is, the more significantly SCM optimizes the number of running servers. Therefore \(OE_R\) is minimized at \(\alpha_2 = 1\) and maximized at \(\alpha_2 = 0.2\). With the enhancement of server capability, more service instances can be held in each server, thus fewer servers can hold the same number of policies. Therefore, a larger \(w_s\) leads to a smaller \(OE_R\), as shown in Fig 7(A), 7(B) and 7(C).

4) Optimization of service instance migration cost. The migration rate of service instance \(OE_Q\) is used to evaluate the cost of migration in SCM. The curves of \(OE_Q\) are shown in Fig 8 under different values of \(w_s\) and weight parameters. As shown in Fig 8(A), within 0–3000 time units, the value of \(OE_Q\) increases gradually with the increase of the number of deployed policies. After 3000 time units, when the number of policies stabilizes, \(OE_Q\) does as well. With the increase of the number of deployed policies, the amount of idle resources in the server declines. Because of the resource competition, the migration of a single service instance may lead to the re-layout of multiple service instances, leading to the increase of \(OE_Q\). When \(\alpha_3\) in Eq (1) increases, the effect of \(Q\) on \(Utility\) is enhanced. Therefore, \(OE_Q\) tends to decrease with the increase of \(\alpha_3\). Thus, \(OE_Q\) is maximized at \(\alpha_3 = 0.1\) and minimized at \(\alpha_3 = 0.5\) as shown in Fig 8(A), 8(B) and 8(C). When the capabilities of the servers are enhanced, the amount of idle resources of servers increases and the resource competition is less intense, resulting in a decrease in the value of \(OE_Q\). Thus, when the weights are equal, \(OE_Q\) decreases with an increase in the value of \(w_s\), as shown in Fig 8.

5) Efficiency of PSO. In PSO, solutions may exist which do not satisfy the constraints. Eq (26) is used in SCM to update the particles’ positions to decrease the number of infeasible
solutions. Let $N_I$ be the number of infeasible solutions generated in the optimization. The occurrence probability of infeasible solutions can be calculated by Eq (32).

\[ P(N_I) = \frac{N_I}{N_p \times MI} \]  

(32)

The comparison of $P(N_I)$ between the traditional PSO and our modified PSO is shown in Fig 9, where the x-axis represents the network operation time, and y-axis represents $P(N_I)$. The weights in the optimization target are $\alpha_1 = 0.5$, $\alpha_2 = 0.3$, $\alpha_3 = 0.2$. As shown in Fig 9, it is difficult for traditional PSO to find feasible solutions, because it is difficult to obtain solutions meeting the constraint in Eq (13) in traditional PSO due to the randomness in updating particle locations. SCM avoids the occurrence of numerous infeasible solutions using a modified PSO, which improves the optimization efficiency. At the early stage, a small number of policies
are deployed in the network, so that there are enough server resources for deploying server chains. Hence, the constraints can be easily satisfied, and the occurrence probability of infeasible solutions is low. With an increasing number of policies in the network, server resources become tight and the occurrence probability of infeasible solutions increases. The larger $w_s$ is more likely to meet the server capacity constraint. Therefore, the occurrence probability of infeasible solutions decreases when $w_s$ increases.

6) Efficiency of encouragement function. The encouragement function tends to migrate the instances in low-load servers to reduce the number of running servers. Let $R_m$ and $R'_m$ be the number of running servers after migration with and without the encouragement function, respectively. The reduction rate of number of running servers $rRate$ is proposed, which can be

---

**Fig 7. Optimization of the number of running servers by SCM.** (a) $w_s = 5$, (b) $w_s = 10$, (c) $w_s = 15$.

https://doi.org/10.1371/journal.pone.0189336.g007
calculated by Eq (33).

\[ rRate = 1 - \frac{R_m}{R^n} \]  \hfill (33)

In this experiment, we deploy 40 policies in the network. The weights in the utility function are \( \alpha_1 = 0.5, \alpha_2 = 0.3, \alpha_3 = 0.2 \). Fig 10 shows the influence of the encouragement function to the number of running servers, where the x-axis represents server capacity, and y-axis represents rRate. Clearly, the number of running servers decreases when the encouragement function is used (\( rRate > 0 \)). The encouragement function encourages service instances to migrate out of small-load servers. Thus, these service instances tend to be gathered to less servers, and the number of running servers decreases. The larger \( w_s \) is, the number of running servers will be reduced by a larger amount. This can be explained by the fact that the server capacity is the bottleneck.
Fig 9. Comparison of occurrence probability of infeasible solutions between traditional PSO and modified PSO.

https://doi.org/10.1371/journal.pone.0189336.g009

Fig 10. Reduction rate of number of running servers when encouragement function is used with different $w_s$.

https://doi.org/10.1371/journal.pone.0189336.g010
against reducing the number of running servers, when \(w_s\) is small. With the increase of \(w_s\), the optimization of the encouragement function is more significant. When \(w_s \geq 10\), the encouragement function brings about a 20% reduction of the number of running servers.

Conclusions and prospects
In the network deployed with services, the service layout may become inefficient with network evolution, resulting in a waste of network resources and energy. In this paper, we show two typical scenarios to illustrate the inefficiency of network resource deployment with network evolution. Regarding this problem, the SCM method is proposed to optimize the deployment of network resources. SCM migrates service chains and adjusts network resource allocation with the consideration of network performance, resource constraints, and instance migration cost. We model SCM as an ILP problem and solve it via modified PSO. An SCM prototype system, FlowMover, is designed based on the SDN controller. Our experiments show that SCM can reduce the traffic cost and energy consumption effectively. SCM is designed to increase the efficiency of the service instance layout, but the service type is ignored. However, different types of service have different requirements to networks. For example, IPTV services are sensitive to the network delay while file transmission services are sensitive to the bit error rate. In future work, service type will be considered in service chain migration for a better service quality.

Author Contributions
Conceptualization: Qi Zhao, Zheng Zhao.
Data curation: Qi Zhao, Zheng Zhao.
Formal analysis: Qi Zhao, Zheng Zhao.
Funding acquisition: Chuanhao Zhang.
Investigation: Chuanhao Zhang.
Methodology: Qi Zhao, Zheng Zhao.
Resources: Chuanhao Zhang, Zheng Zhao.
Software: Qi Zhao.
Supervision: Zheng Zhao.
Validation: Qi Zhao, Chuanhao Zhang, Zheng Zhao.
Writing – original draft: Qi Zhao, Zheng Zhao.
Writing – review & editing: Qi Zhao, Chuanhao Zhang, Zheng Zhao.

References
1. Sekar V, Egi N, Ratnasamy S, Reiter MK, Shi G, editors. Design and implementation of a consolidated middlebox architecture. Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation; 2011: 24–24.
2. M. Boucadair, C. Jacquenet, R. Parker, D. Lopez, J. Guichard, and C. Pignataro, Service Function Chaining: Framework and Architecture, Active Internet-Draft, IETF Secretariat, Internet-Draft draft-boucadair-sfc-framework-02, Feb. 2014. Available from: https://tools.ietf.org/html/draft-boucadair-sfc-framework-02.
3. Mijumbi R, Serrat J, Gorricho J, Bouten N. Network Function Virtualization: State-of-the-art and Research Challenges. IEEE Communications Surveys & Tutorials. 2015; 18(1):236–262.
4. Han B, Gopalakrishnan V, Ji L, Lee S. Network function virtualization: Challenges and opportunities for innovations. IEEE Communications Magazine. 2015; 53(2):90–97.

5. Chiosi M, Clarke D, Willis P, Reid A, Feger J, Bugenhagen M, et al., editors. Network functions virtualisation introductory white paper. SDN and OpenFlow World Congress; 2012.

6. Kreutz D, Ramos F M V, Verissimo P E, Rothenberg C E, Azodolmolky S, Uhlig S. Software-defined networking: A comprehensive survey. Proceedings of the IEEE, 2015; 103(1): 14–76.

7. Wickboldt J, De Jesus W, Isolani P, Both C. Software-defined networking: management requirements and challenges. IEEE Communications Magazine. 2015; 53(1):278–85.

8. McKeown N. Software-defined networking. INFOCOM keynote talk. 2009; 17(2):30–32.

9. Gember A, Krishnamurthy A, John SS, Grandl R, Gao X, Anand A, et al. Stratos: A network-aware orchestration layer for middleboxes in the cloud. Technical Report, 2013. Available from: http://www.cs.duke.edu/courses/spring14/cps214/Papers/Stratos.pdf.

10. Anwer B, Benson T, Fearnster N, Levin D, Rexford J, editors. A slick control plane for network middleboxes. Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking; 2013: 147–148.

11. Fayazbaksh SK, Chiang L, Sekar V, Yu M, Mogul JC. Enforcing network-wide policies in the presence of dynamic middlebox actions using flowtags. Proceedings of the 11th USENIX Conference on Networked Systems Design and Implementation; Seattle, WA. 2616497: USENIX Association; 2014. p. 533–546.

12. Zhang Y, Beheshti N, Beliveau L, Lefebvre G, Manghirmalani R, Mishra R, et al., editors. Steering: A software-defined networking for inline service chaining. Network Protocols (ICNP), 2013 21st IEEE International Conference on; 2013: 1–10.

13. Matias J, Garay J, Toledo N, Unzilla J, Jacob E. Toward an SDN-enabled NFV architecture. IEEE Communications Magazine. 2015; 53(4):187–93.

14. Qazi ZA, Tu C-C, Chiang L, Miao R, Sekar V, Yu M, editors. SIMPLE-lying middlebox policy enforcement using SDN. Proceedings of the ACM SIGCOMM 2013 conference on SIGCOMM; 2013: 27–38.

15. Gember-Jacobson A, Viswanathan R, Prakash C, Grandi R, Khalid J, Das S, et al., editors. OpenNF: Enabling innovation in network function control. Proceedings of the 2014 ACM Conference on SIGCOMM; 2014: 163–174.

16. Xiong G, Hu Y, Lan J, Cheng G. A Mechanism for Configurable Network Service Chaining and Its Implementation. KSII Transactions on Internet & Information Systems. 2016; 10(8):3701–3727.

17. G X, Y H, L T, a.e. A virtual service placement approach based on improved quantum genetic algorithm. Frontiers of Information Technology & Electronic Engineering. 2016; 17(7):661–671.

18. Kennedy J, Eberhart RC, editors. A discrete binary version of the particle swarm algorithm. Systems, Man, and Cybernetics, 1997 Computational Cybernetics and Simulation, 1997 IEEE International Conference on; 1997: 5:4104–4108.

19. McCauley M. About pox. 2013. Available from: http://www.noxrepo.org/pox/about-pox.

20. Abdelaziz A, Fong AT, Gani A, Garba U, Khan S, Akhunzada A, et al. Distributed controller clustering in software defined networks. PLOS ONE. 2017; 12(4). https://doi.org/10.1371/journal.pone.0174715.

21. Mann V, Gupta A, Dutta P, Vishnoi A, Bhattacharya P, Poddar R, et al. Remedy: Network-aware steady state VM management for data centers. NETWORKING, Springer; 2012. p. 190–204.

22. Wang M, Meng X, Zhang L, editors. Consolilating virtual machines with dynamic bandwidth demand in data centers. Proceedings of INFOCOM, 2011; 2011: 71–75.

23. Kansal NJ, Chana I. Energy-aware Virtual Machine Migration for Cloud Computing—A Firefly Optimization Approach. Journal of Grid Computing. 2016:1–19.

24. Zhao J, Hu L, Ding Y, Xu G, Hu M. A Heuristic Placement Selection of Live Virtual Machine Migration for Energy-Saving in Cloud Computing Environment. PLOS ONE. 2014; 9(8). https://doi.org/10.1371/journal.pone.0068275.

25. Wen C, Long X, Mu Y. Adaptive Controller for Dynamic Power and Performance Management in the Virtualized Computing Systems. PLOS ONE. 2013; 8(2). https://doi.org/10.1371/journal.pone.0057551.

26. Tian Y, Cheng R, Zhang X, Cheng F, Jin Y. An Indicator Based Multi-Objective Evolutionary Algorithm with Reference Point Adaptation for Better Versatility. IEEE Transactions on Evolutionary Computation. 2017; PP(99):1–1.

27. Zhang X, Tian Y, Cheng R, Jin Y. A Decision Variable Clustering-Based Evolutionary Algorithm for Large-scale Many-objective Optimization. IEEE Transactions on Evolutionary Computation. 2017; PP(99):1–1.

28. Xue Y, Jiang J, Zhao B, Ma T. A self-adaptive artificial bee colony algorithm based on global best for global optimization. Soft Computing. 2017(8):1–18.
29. Zhang L, Pan H, Su Y, Zhang X, Niu Y. A Mixed Representation-Based Multiobjective Evolutionary Algorithm for Overlapping Community Detection. IEEE Transactions on Cybernetics. 2017; PP(99):1–14.

30. Liu B, Sun N, Zhang Q, Grout V, Gielen G, editors. A surrogate model assisted evolutionary algorithm for computationally expensive design optimization problems with discrete variables. Evolutionary Computation; 2016: 1650–1657.

31. Buckley JJ, Feuring T. Evolutionary algorithm solution to fuzzy problems: Fuzzy linear programming. Fuzzy Sets & Systems. 2000; 109(1):35–53.

32. Zhang X, Duan F, Zhang L, Cheng F, Jin Y, Tang K. Pattern Recommendation in Task-oriented Applications: A Multi-Objective Perspective [Application Notes]. IEEE Computational Intelligence Magazine. 2017; 12(3):43–53.

33. Goldberg DE. Genetic algorithms in search, optimization, and machine learning. Addison wesley. 1989.

34. Wood T, Ramakrishnan K, Shenoy P, Van der Merwe J, editors. CloudNet: dynamic pooling of cloud resources by live WAN migration of virtual machines. ACM SIGPLAN Notices; 2011; 46(7): 121–132.

35. Wang Y, Keller E, Biskeborn B, Merwe JVD, Rexford J, editors. Virtual routers on the move:live router migration as a network-management primitive. 2008, 38(4): 231–242.

36. Singh A, Korupolu M, Mohapatra D, editors. Server-storage virtualization: integration and load balancing in data centers. Proceedings of the 2008 ACM/IEEE conference on Supercomputing; 2008: 1–12.

37. Medina A, Matta I, Byers J. BRITE: a flexible generator of Internet topologies. 2000. Available from: https://dl.acm.org/citation.cfm?id=864341.

38. Cheng G, Chen H, Hu H, Wang Z, Lan J. Enabling network function combination via service chain instantiation. Computer Networks. 2015, 92: 396–407.