sustainability

MDPI

Article

Energy Efficient Deployment of a Service Function Chain for Sustainable Cloud Applications

Jian Sun [1], Yue Chen [2], Miao Dai [1], Wanting Zhang [1], Arun Kumar Sangaiah [3], Gang Sun [1,4,*] and Han Han [2]

1 Key Lab of Optical Fiber Sensing and Communications (Ministry of Education), University of Electronic Science and Technology of China, Chengdu 611731, China; sj@uestc.edu.cn (J.S.); 201822010335@std.uestc.edu.cn (M.D.); 201621010322@std.uestc.edu.cn (W.Z.)
2 National Computer Network Emergency Response Technical Team, Coordination Center of China, Beijing 100029, China; chenyue@cert.org.cn (Y.C.); hanh@cert.org.cn (H.H.)
3 School of Computing Science and Engineering, Vellore Institute of Technology, Tamil Nadu 632014, India; arunkumarsangaiah@gmail.com
4 Center for Cyber Security, University of Electronic Science and Technology of China, Chengdu 611731, China
* Correspondence: gangsun@uestc.edu.cn; Tel.: +86-28-61831578

Received: 15 September 2018; Accepted: 27 September 2018; Published: 29 September 2018

Abstract: With the increasing popularity of the Internet, user requests for cloud applications have dramatically increased. The traditional model of relying on dedicated hardware to implement cloud applications has not kept pace with the rapid growth in demand. Network function virtualization (NFV) architecture emerged at a historic moment. By moving the implementation of functions to software, a separation of functions and hardware was achieved. Therefore, when user demand increases, cloud application providers only need to update the software; the underlying hardware does not change, which can improve network scalability. Although NFV solves the problem of network expansion, deploying service function chains into the underlying network to optimize indicators remains an important research problem that requires consideration of delay, reliability, and power consumption. In this paper, we consider the optimization of power consumption with the premise of guaranteeing a certain virtual function link yield. We propose an efficient algorithm that is based on first-fit and greedy algorithms to solve the problem. The simulation results show that the proposed algorithm substantially improves the path-finding efficiency, achieves a higher request acceptance ratio and reduces power consumption while provisioning the cloud applications. Compared with the baseline algorithm, the service function chain (SFC) acceptance ratio of our proposed algorithms improves by a maximum of approximately 15%, our proposed algorithm reduces the power consumption by a maximum of approximately 15%, the average link load ratio of our proposed algorithm reduces by a maximum of approximately 20%, and the average mapped path length of our proposed algorithm reduces by a maximum of approximately 1.5 hops.

Keywords: energy efficient; network function virtualization; service function chain; deployment; cloud application

1. Introduction

Traditional cloud application providers (CAPs) deploy network function via dedicated hardware [1], meaning network function is, consequently, closely linked to a device. When a user's demands change, the CAP must adjust and upgrade the hardware network to adapt to this change. Even with small changes at the functional level, the underlying equipment may require adjustment for a long time, and the flexibility and scalability of the network are limited.

With the rapidly increasing popularity of cloud computing, the number of cloud application requests has dramatically increased, and traditional operating models have been unable to handle high-frequency device updates and upgrades in cloud computing. However, network function virtualization (NFV) [2] implements the separation of software and hardware; it overcomes the drawbacks that specific functions cannot be quickly upgraded, due to the attachment of the dedicated hardware in the traditional mode and substantially improves the flexibility and scalability of a network. Therefore, NFV is favored by cloud application providers.

A service function chain (SFC) is the main form of NFV to implement cloud applications. Each SFC is composed of a sources node, destination node and multiple virtual network function (VNF) modules, which are connected in order by virtual links. Thus, an arrived cloud application request is abstracted to the corresponding SFCl and deployed to the underlying network of cloud computing.

Previous studies of SFC deployment primarily focused on network delay, network reliability, and network service quality and rarely considered power consumption while provisioning a cloud application. Statistics show that the power consumption of a cloud data center is equivalent to the power consumption of 25,000 homes [3]. Amazon's evaluation of its data center shows that the cost of operation and maintenance accounts for 53% of the total budget of the data center, whereas 42% of the operation and maintenance cost stem from power consumption [4]. The annual electricity cost of the Akamai Company in the United States is approximately 10 million US dollars [5]. China Mobile consumed 13 TWh of energy in 2011 [6]. Telecom Italia consumes 2 TWh of energy each year [7].

High power consumption will shorten device life, create energy waste and carbon emissions, and aggravate the greenhouse effect. However, reducing the power consumption that is used to map the virtual network function can generate lower costs, increase revenue for cloud application providers, and improve commercial competitiveness in industry. Therefore, implementation of an energy-aware management strategy for infrastructure resources is necessary, and the need to perform energy-aware virtual network function deployment research for cloud application provision is urgent. The main contributions of our work are detailed as follows:

- We implement energy-saving SFC deployment in a wavelength division multiplexing (WDM)-based optical cloud network. We briefly describe the energy-efficient next-hop selection (ENSF) algorithm, proposed in Reference [8], and analyze its advantages and disadvantages.
- We propose the new algorithm direction-guided FirstFit (DGFF), which introduces "direction guidance". This algorithm employs efficient and accurate path-finding strategies that significantly improve its performance in cloud application provisioning.
- We introduce the concept of "load balancing" and propose the direction-guided greedy (DGG) algorithm, which is based on the DGFF algorithm. DGG can reduce network bottleneck nodes and links and improve network stability.
- We simulate the proposed algorithm in terms of the SFC total acceptance ratio, SFC average power consumption, average mapped cost, the average running time of the algorithm, average node load ratio and average link load ratio of the network. We make an objective comparison to analyze their performance. A large number of simulation results show that the DGFF algorithm and the DGG algorithm are superior to the ENSF algorithm [8] in cloud application provisioning.

The remainder of this paper is structured as follows: Section 2 introduces related studies; Section 3 introduces and describes the SFC deployment model; Section 4 proposes a heuristic algorithm to implement energy-saving SFC deployment; Section 5 presents and analyzes the simulation results; and Section 6 concludes the paper.

2. Related Work

Numerous studies have focused on the problem of VNF deployment for cloud applications. The authors in References [9–20] provide the method and purpose of early virtual network deployment: Improve Internet service provider (ISP) revenue by mapping more virtual requests to the same

underlying network. We compare the performance of the proposed algorithm with existing algorithms, which as shown in Table 1.

Table 1. Comparison of the related studies.

Main Researched Problems	Proposed Algorithm	Information Exchange	Running Time	Average Revenue	Average Cost	Acceptance Ratio
Using distributed method to achieve better virtual network (VN) mapping [10].	distributed VN mapping	↓ 60%	↓ 90%	Null	Null	Null
Simultaneous node and link mapping to achieve rapid VN mapping [11].	vnmFlib	Null	↓ 80%	↑ 50~60%	↓ 50%	Null
Deploying VN onto a separable network [12].	RW-based, CB-Based	Null	Null	↑ 15~17%	↓ 32~35%	↑ 18~21%
Collaborative consideration of node and link mapping to achieve better VN mapping [13].	D-ViNE, R-ViNE, ViNE-LB, ViNE-SP	Null	Null	↑ 35~40%	↓ 18~20%	↑ 30~35%
Deploying service function chain in multi-domain networks [16].	AVG_D, AVG_LB, WGT_D, WGT_LB	Null	↓ 30~35%	Null	↓ 20~30%	Null

In Reference [21], the authors abstract the cloud application-based SFC deployment as an optimization model and propose the heuristic algorithm. By introducing Markov approximation to calculate the priority deployment node of the SFC, then calculating the node that preferentially accepted the SFC and reducing the solution space of the model, algorithm convergence was accelerated. The algorithm attempts to integrate each SFC into a single or similar physical node and reduce overhead by sharing physical resources.

Yang et al., in Reference [22], proposed an algorithm called merge-RD. They defined a physical quantity, "relational depth" (RD), between VNF modules. They tried to integrate the deep relational modules into the same or similar physical nodes by calculating RD. In this manner, the interaction traffic of the network can be reduced to decrease the power consumption.

W. Racheg et al., in Reference [23], abstracted the SFC deployment problem into an integer linear programming (ILP) model and proposed the extensive search algorithm (ESA). They used the brute force method for each SFC to search for all possible paths between its source and destination, calculated the total revenue for each, and chose the path with the highest income. However, in a large and complex network, algorithm convergence will be very slow, because the possible solution space of an actual operation is too large. Therefore, ESA is not suitable as an online SFC mapping algorithm for cloud applications.

In the studies reported in References [24,25], on virtual network deployment, the authors divided the whole process into two parts: Node mapping and link mapping. In the node mapping process, the authors considered whether the available physical nodes are matched with the requirements of virtual nodes, and whether the selection of the current physical nodes is beneficial to subsequent link mapping. Then, they chose the optimal node as the target. The link mapping process is based on node mapping, where the authors adopted the shortest path strategy to find the optimal path between the mapped nodes to complete the mapping process. The authors in References [26–28] believed that an ordered VNF map (i.e., SFC map) is essentially a special virtual network embedding (VNE) map or an extension of the VNE; however, they have different goals and constraints.

M. Tajiki et al., in Reference [8], proposed a novel resource allocation architecture that enables cloud application-based SFCs to be deployed to SDN-based networks with minimal power consumption. For this reason, they considered the VNF layout, VNF allocation to flows and flow routing for optimization, and modeled the problem. In addition, they modeled traffic rerouting to reduce the impact of resource fragmentation on network utilization. Since then, the authors proposed a

heuristic algorithm for various optimization problems and evaluated the performance of the proposed algorithm in the actual topology under various network traffic patterns. The results confirm that the proposed heuristic algorithm provides a near-optimal solution in an execution time that is applicable to the actual network.

In Reference [29], the authors used the GWATT tool from Bell Labs to estimate the possible energy-saving effects of three major NFV use cases: (1) Virtualized evolved packet core (VEPC); (2) virtualized customer premises equipment (VCPE); and (3) virtualized radio access network (VRAN). They determined that the mobile network with the highest energy utilization prospect is the evolved packet core (EPC); the virtualization produced a 22% reduction in power consumption and a 32% increase in energy efficiency. The authors also showed that power consumption savings will be even higher if traffic on the network is increased.

3. Preliminaries and Modeling

3.1. SFC Mapping

We use an undirected weight graph to describe the cloud application-based SFC model: $G_v = (V, E, R^v, R^E)$, where V and E denote the VNF module set and the virtual link set, respectively. The resource requirements of the VNF modules and links are denoted as R^v and R^E, respectively. As shown in Figure 1a, s indicates the SFC source physical node, t indicates the destination physical node, v_i indicates the VNF module, c_i is the corresponding central processing unit (CPU) resources requirement, d_i indicates the storage resource requirement, and b_i indicates the virtual link bandwidth requirement for a cloud application. We know the resource requirements of the VNF module are mainly CPU resources (computing resources) and storage resource requirements. The virtual link resource requirements are mainly link bandwidth requirements.

There are fewer CPU computing resources in the physical node resources, and the hardware storage space is usually large enough [30]. To increase the convenience of the subsequent simulation, we re-abstract the cloud application-based SFC model. As shown in Figure 1b, we only consider the CPU resource requirements of the VNF module, and the bandwidth requirement of the virtual link is set to the maximum of all the bandwidth requirements.

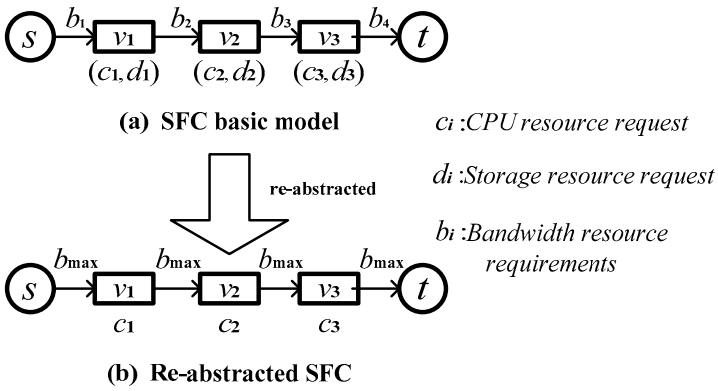


Figure 1. Service function chain (SFC) model.

The main problem of cloud application-based SFC deployment is identifying a physical path between a source node and destination node on the underlying network that can carry all VNF modules of the SFC. This deployment process needs to satisfy certain constraints.

Figure 2 shows the deployment process of a cloud application-based SFC: The source and destination nodes of the SFC are mapped onto the physical nodes A and D, and the VNFs are mapped in sequence. The green nodes represent the physical nodes that carry the VNF modules. The mapping pairs are as follows: (v_1, A), (v_2, B), and (v_3, E); this follows the one-to-one mapping principle (a physical node can only carry one VNF of the SFC). The hollow nodes represent the forwarding nodes that do

not carry the VNF module, the virtual link is mapped to the physical link, as shown in the green link in Figure 2, and the mapping pairs are as follows: {(v_1, v_2), (A, B)}, {(v_2, v_3), (B, E)}, and {(v_3, D), (E, D)}. The traffic passes through all VNF modules of the SFC in the direction of the green arrow in Figure 2.

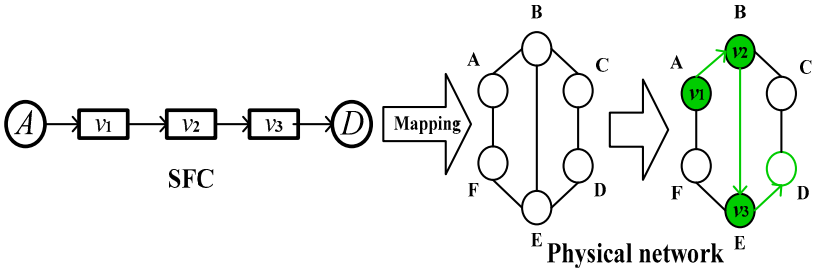


Figure 2. SFC mapping example.

After a cloud application-based SFC is expired, all server resources and communication resources allocated to the SFC are recovered and managed by the underlying network.

3.2. Underlying Network

The underlying physical network topology used in this paper is a WDM-based optical network [31–35]; its communication link medium is optical fiber. There are optical transmitters and optical receivers at the end of the fiber link, and there are optical amplifiers on the link. Thus, the fiber links will introduce more link power consumption than the traditional physical link. Figure 3a shows a more realistic WDM network model.

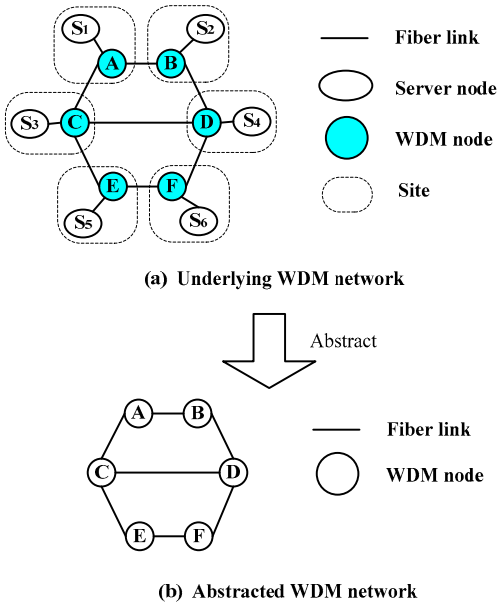


Figure 3. Underlying wavelength division multiplexing (WDM) network, and abstracted WDM model.

In this model, the underlying sites consist of two parts: (1) Server nodes, which provide server resources, such as CPU resources and storage resources; and (2) WDM nodes, which provide communication resources, such as optical signal receiving and sending, data flow forwarding, and optical wavelength conversion. The main resource of the underlying link is wavelength capacity, which can also be understood as bandwidth capacity.

To facilitate the subsequent simulation, we abstract the original WDM network model in this paper, as shown in Figure 3b. Additionally, we abstract the sites consisting of the WDM node and the server node into the integrated nodes, which include all functions and attributes of the sites

under the original model; the link model is almost unchanged. We use an undirected weight graph to represent the abstract model, which is denoted by: $G_s = (N, L, A^N, A^L)$, where N and L represent the physical node set and the fiber link set, respectively, and each node n, $n \in N$ has certain server resources (CPU resources and storage resources) and communication resources (wavelength resources and switching capabilities). Each link l, $l \in L$ has a certain wavelength resource (bandwidth resource). A^N and A^L represent the attributes of the node and the fiber link, respectively; the typical node attributes are the server resource capacity and the communication resource capacity, and the link attribute is the wavelength capacity.

3.3. Power Consumption Model

The power consumption of the cloud application SFC deployment considered in this paper includes two parts: Node power consumption and link power consumption.

(a) Node Power Consumption Model

When physical node n, $n \in N$ is in the standby state, it generates a certain load-independent power consumption, called idle power consumption, which we use P^n_{idle} to denote. In this paper, it is assumed that the node idle power consumption has a linear relationship with the total CPU resources provided by the physical node. When the node carries a certain load, the computational overhead of the VNF module on the node will generate the additional load-related power consumption, which is denoted by P^n_{load} in this paper.

In [36], the authors studied the relationship between the load-related node power consumption and the node load ratio; they found that load-dependent power consumption increases linearly with node load ratio, as shown in Equation (1):

$$P^n_{load}(u) = (P^n_{busy} - P^n_{idle}) \cdot u, \tag{1}$$

where P^n_{busy} represents the power consumption when the node computing resources are fully loaded, and u represents the node computing resource utilization.

In addition, the underlying network of this paper is based on the WDM network. However, it is usually only in a large-scale network that a few physical nodes with the wavelength conversion function are introduced. Because the underlying network is set to be free from the wavelength conversion, our power model of the underlying network node does not include the conversion power consumption $P^n_{waveChange}$.

Finally, when node n forwards user data, it will generate forwarding power consumption $P^n_{forwarding}$. Compared to idle power consumption and load-related power consumption, forwarding power consumption is lower. For the implementation in this paper, a small random number is generated for each node as the forwarding power consumption; each SFC will generate the forwarding power through the physical node.

In summary, the power consumption model of the underlying node used in this paper is shown in Equation (2).

$$P^n_{Node} = P^n_{idle} + P^n_{load} + P^n_{forwarding} \cdot PassedSFC, \tag{2}$$

where *PassedSFC* represents the number of SFCs currently passing on node n.

(b) Link Power Consumption Model

There are three main sources of link power consumption in WDM networks: Optical transmitter power consumption, optical receiver power consumption, and optical amplifier power consumption.

User data is processed at the physical node and then sent to the fiber link; it must go through the optical transmitter at the fiber transmitter to achieve electro-optical conversion. Correspondingly, the optical signal received by the physical node at the opposite end of the link must go through the optical receiver to achieve optical-electro conversion. In addition, to compensate for the loss of

long-distance transmission and continuous reflection of optical signals, we must introduce the optical amplifier on the fiber link to increase the signal strength to prevent the signal from being incorrectly recognized and processed when it is transmitted to the peer end. Therefore, optical transmitter power consumption, optical receiver power consumption and optical amplifier power consumption will be generated on the link.

The optical transmitter and receiver power consumptions of link l are denoted as δ_l^t and δ_l^r, respectively, and the optical amplifier power consumption is denoted as δ_l^a. For δ_l^t and δ_l^r, the implementation in this paper generates a relatively small random number, and δ_l^a is related to the length of link l. In [37], the authors provide the relationship between them, as shown in Equation (3):

$$\delta_l^a = (\frac{d_l}{80} + 2) \times 9, \tag{3}$$

where d_l is the length of fiber link l in kilometers (km).

Currently, each SFC passing through link l will generate three corresponding power consumptions: δ_l^t, δ_l^r and δ_l^a. Therefore, total power consumption P_{Link}^l of link l can be calculated according to Equation (4):

$$P_{Link}^l = (\delta_l^t + \delta_l^r + \delta_l^a) \cdot PassedSFC, \tag{4}$$

where *PassedSFC* represents the number of SFCs currently passing on link l.

3.4. Performance Index

We use the following performance indices to compare the performance of the proposed algorithm to that of the comparison algorithm [8]:

(a) SFC acceptance ratio:

The SFC acceptance ratio R_a is the ratio of the number of accepted SFCs to the total number of SFCs that arrived. The definition is as shown in Equation (5):

$$R_a = \frac{AcceptedSFC}{ArrivedSFC}. \tag{5}$$

(b) Average SFC power consumption:

Average SFC power consumption $P_{average}$ is the ratio of the current total power consumption of the network to the number of SFCs that are currently carried. The definition is as shown in Equation (6).

$$P_{average} = \frac{P_{total}}{AllocatedSFC} \tag{6}$$

The denominator represents the number of SFCs running on the current network, and the numerator represents the current total power consumption of the entire network. P_{total} can be calculated according to Equation (7).

$$P_{total} = \sum_{i \in N} P_{Node_i} + \sum_{j \in L} P_{Link_j} \tag{7}$$

The first part of Equation (7) represents the power consumption of all nodes of the underlying physical network, and the second part represents the power consumption of all links of the underlying physical network.

(c) Average node load ratio:

Average node load ratio U_{AN} indicates the load ratio that each SFC in the bearer state imposes on the underlying physical network node. The definition is shown in Equation (8).

$$U_{AN} = \frac{\sum_{i \in N} U_{Node_i}}{AllocatedSFC} \tag{8}$$

The numerator represents the sum of the load ratios of the underlying physical nodes, and the denominator represents the number of SFCs in the receiving and running states on the current underlying network.

(d) Average link load ratio:

Average link load ratio U_{AL} indicates the load ratio that each SFC in the bearer state imposes on the underlying physical network link. The definition is shown in Equation (9). The numerator represents the sum of the load ratios of the underlying physical links, and the denominator represents the number of SFCs that are running on the current underlying network.

$$U_{AL} = \frac{\sum_{j \in L} U_{Link_j}}{AllocatedSFC} \tag{9}$$

3.5. Optimization Objective and Constraint Condition

(a) Optimization Objective

In this paper, we study the power consumption overhead reduction of the SFC mapping in the underlying network under the guarantee of a certain SFC request acceptance ratio. The problem is abstracted as an optimization mathematical model, and the optimization objective function definition is as shown in Equation (10):

$$Minmize\{P_{Network} = \sum_{n \in N} P^n_{Node} + \sum_{l \in L} P^l_{Link}\} \tag{10}$$

Objective Equation (10) attempts to reduce the total power consumption of the underlying network, where the first part represents the power consumption of all nodes of the underlying network and the second part represents the power consumption of all links of the underlying network.

(b) Node Constraints

The node constraints that must be met to complete the mapping of service function chains are as follows:

$$\sum_{n \in N} h_{vn} = 1, \ \forall v \in V, \tag{11}$$

$$\sum_{v \in V} h_{vn} \leq 1, \ \forall n \in N, \tag{12}$$

$$t(n) \leq 1, \ \forall n \in N, \tag{13}$$

$$st(n) = t(n), \forall n \in N, \tag{14}$$

$$rC(n) \leq avaC(n) \ , \quad \forall n \in N \ , \tag{15}$$

$$rC(n) = \sum_{\forall v \in V} req(v) \cdot h_{vn} \ , \quad \forall n \in N \ , \tag{16}$$

where h_{vn} indicates whether VNF module $v, v \in V$ is deployed onto underlying physical node $n, n \in N$ (if v is deployed onto n, h_{vn} is 1; otherwise, h_{vn} is 0). $t(n)$ represents the number of times the SFC passes through node n. $st(n)$ represents whether physical node n is in the running state (if n is in the running state, b is 1; otherwise, b is 0). $rC(n)$ and $avaC(n)$ represent the number of computing resources provided by node n for the SFC and the number of computing resources available at present, respectively. $req(v)$ represents the computing resource requirement of VNF module v.

Equation (11) indicates that each VNF module of the SFC can only be deployed to one node on the underlying network. Equation (12) indicates that each node of the underlay network supports at most one VNF module of the SFC. These two conditions ensure that each VNF module requested by the SFC is mapped into the underlying physical network in a one-to-one mode. Equation (13) indicates that the SFC data flows at most once through each physical node, which ensures that the mapping of SFCs on the underlying network does not create loops. Equation (14) ensures that underlying physical node n will be powered on only if there is an SFC passing through, which avoids introducing unnecessary idle power. Equation (15) is the resource constraint of physical node n, which ensures that the computing resources provided by node n to the SFC must not exceed its own currently available computing resources. Equation (16) indicates that the computing resource provided by physical node n to the SFC is equal to the computing resource requirement of the VNF module deployed onto the node.

(c) Link Constraints

The link constraints that must be met to complete the mapping of service function chains are as follows:

$$\sum_{l \in L} h_{el} \geq 1, \forall e \in E, \tag{17}$$

$$\sum_{e \in E} h_{el} \leq 1, \forall l \in L, \tag{18}$$

$$rB(l) \leq avaB(l), \forall l \in L, \tag{19}$$

$$rB(l) = \sum_{e \in E} req(e) \cdot h_{el}, \forall l \in L, \tag{20}$$

$$\sum_{j \in N_s^{in}} b_{j,s} = \sum_{j \in N_t^{out}} b_{t,j} = 0, \tag{21}$$

$$\sum_{j \in N_s^{out}} b_{s,j} = \sum_{j \in N_t^{in}} b_{j,t} = B, \tag{22}$$

$$\sum_{j \in N_i^{out}} b_{i,j} = \sum_{j \in N_i^{in}} b_{j,i}, \forall i \in N, i \neq s, i \neq t, \tag{23}$$

where h_{el} indicates whether virtual link $e, e \in E$ is mapped into physical link l (if e is mapped into l, h_{el} is 1; otherwise, h_{el} is 0). $rB(l)$ indicates the bandwidth resource provided by link l to the SFC, and $avaB(l)$ indicates the available bandwidth resource of link l at present. $req(e)$ indicates the bandwidth requirement of virtual link e. $b_{i,j}$ denotes the directed SFC traffic on physical link <i, j>, N_i^{in} represents the set of the adjacency nodes of physical node i, N_i^{out} represents the set of the adjacency nodes of physical node i, and B denotes the bandwidth requirement of the SFC.

Equation (17) indicates that virtual link e is mapped into at least one physical link or a physical path that is formed by multiple physical links. Equation (18) indicates that physical link l supports at most one virtual link. Equation (19) is the physical link resource constraint, which indicates that the bandwidth resources provided by physical link l to the SFC do not exceed the available bandwidth resources at present. Equation (20) indicates that the bandwidth resource provided by underlying link l to the SFC is equal to the virtual link bandwidth resource required by the SFC mapped into physical link l. Equation (21) indicates that the inflow at source node s and the outflow at destination node t are 0. Equation (22) indicates that the outflow of source node s and the inflow of destination node t are equal to the bandwidth requirements of the SFC. Equation (23) indicates that the intermediate physical node carrying the VNF module (excluding the source and destination nodes) meets the traffic conservation constraint.

4. Algorithm Design

Because of the disadvantages of the greedy algorithm, if we want to use the local-to-global algorithm to achieve efficient and accurate path-finding, we must introduce auxiliary measures to change the essential characteristics of blind path-finding. In this way, the acceptance bottleneck of SFC can be broken and the path-finding success ratio can be improved. Therefore, in this section, we propose the direction-guided FirstFit (DGFF) algorithm based on "direction guidance."

As shown in Figure 4, when an SFC reaches the underlying network, the algorithm first extracts its destination node and then uses the Bellman-Ford algorithm or Dijkstra algorithm to update the minimum hops from other nodes to the destination node in the underlying network. Using this auxiliary parameter, the program can roughly determine the relationship between the number of nodes in the current remaining path and the number of VNFs to be mapped.

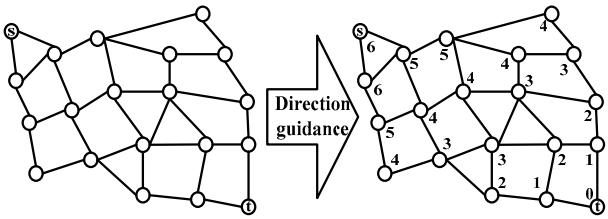


Figure 4. Direction guidance.

For example, if the hop count of the current node is 4 (i.e., there are at least 4 nodes to the destination point without the current node), the number of VNFs to be mapped in the SFC is 3. The algorithm first searches for nodes less than 4 hops from adjacent nodes to see whether they can carry the next VNF module that can quickly approach the destination point. If no node is found that meets the requirements, the algorithm will look for nodes that have the same number hops and can carry the next VNF module in adjacent nodes. If a suitable node still cannot be found, then it will find nodes with hops more than 4 to see whether they meet the requirements. If the number of VNFs to be mapped is 6, then the algorithm first searches for nodes more than 4 hops from the adjacent nodes. The program temporarily seeks the direction away from the destination point in case the number of subsequent nodes is insufficient to accommodate the VNF modules.

The advantage of the DGFF algorithm is that, no matter which node the program is currently on, and which node it is going to next, the program can know whether this step is approaching or moving away from the destination node. In addition, the algorithm also provides a basis for guiding the direction of its own path, which is the relationship between the length of the remaining path and the number of VNFs to be mapped. When the remaining path is short, the path goes away from the destination node; when the remaining path is long, it goes to the destination node. The program achieves efficient path-finding by repeating dynamic adjustments.

The main body of the algorithm is divided into two processes: The downlink process (process = 1), which refers to finding a path to lower nodes (nodes with smaller hops), and the uplink process (process = 2), which refers to finding a path to higher nodes (nodes with larger hops). The algorithm pseudocode is given in Algorithm 1.

Algorithm 1. DGFF Algorithm

Input: Underlying network $G_s = (N, L, A^N, A^L)$;
Service function requests $G_v = (V, E, R^v, R^E)$

Output: node, link mapping pair of SFC on the underlying network

1: Extract destination node t of the SFC and use the Bellman-Ford algorithm to update the shortest number of hops from other nodes to this node in the underlying network.

2: According to the SFC, do not initialize the underlying physical node; initialize the VNF queue to be mapped, the current node as the SFC source node, and the process value according to the current node hop count, and map the number of VNFs;
3: **while** VNF queue to be mapped is not empty, **do**
4: **if** process = 1 **then**
5: **if** the current node can map the current VNF, **then**
6: Establish the mapping relationship between the current node and VNF;
7: **if** the current VNF is not the last VNF, **then**
8: Find a lower node to carry the next VNF; otherwise find a peer node; otherwise find a forwarding node in peer nodes; otherwise find a forwarding node in lower nodes; otherwise find a higher node; otherwise find a forwarding node in higher nodes to carry next VNF;
9: If a node satisfying conditions to carry the next VNF is not found, the SFC is rejected and the program ends; otherwise, the selected node is set as the next current node. If its number of hops is smaller than the number of VNFs to be mapped, process = 2, otherwise process = 1;
10: **end if**
11: **if** the current VNF is the last VNF, **then**
12: Find the shortest path from the current node to the destination node;
13: If path-finding is not successful, reject the SFC and the program ends;
14: **end if**
15: **end if**
16: **if** current node cannot map current VNF, **then**
17: Find a lower node to carry the current VNF; otherwise find a peer node; otherwise find a forwarding node in peer nodes; otherwise find a forwarding node in lower nodes; otherwise find a higher node; otherwise find a forwarding node in higher nodes to carry the current VNF;
18: If not, find a node satisfying the conditions to carry the current VNF, the SFC is rejected and the program ends; otherwise, the selected node is set as the next current node. If its number of hops is smaller than the number of VNFs to be mapped, process = 2; otherwise process = 1;
19: **end if**
20: **end if**
21: **if** process = 2 **then**
22: **if** the current node can map the current VNF, **then**
23: Establish the mapping relationship between the current node and VNF;
24: Find a higher node to carry the next VNF; otherwise find a forwarding node in higher nodes; otherwise find a peer node; otherwise find a forwarding node in peer nodes; otherwise find a lower node; otherwise find a forwarding node in lower nodes to carry the next VNF;
25: If not, find a node satisfying the conditions to carry the next VNF, the SFC is rejected and the program ends; otherwise, the selected node is set as the next current node. If its number of hops is smaller than the number of VNFs to be mapped, process = 2, otherwise process = 1;
26: **end if**
27: **if** current node cannot map current VNF, **then**
28: Find a higher node to carry the current VNF; otherwise find a forwarding node in higher nodes; otherwise find a peer node; otherwise find a forwarding node in peer nodes; otherwise find a lower node; otherwise find a forwarding node in lower nodes to carry current VNF;
29: If not, find a node satisfying the conditions to carry the current VNF, the SFC is rejected and the program ends; otherwise, the selected node is set as the next current node. If its number of hops is smaller than the number of VNFs to be mapped, process = 2, otherwise process = 1;
30: **end if**
31: **end if**
32: **end while**

The DGFF algorithm introduces the "direction-directed" strategy based on the traditional local path-finding algorithm, which alleviates the problem of blind path-finding for the traditional local

algorithms. The algorithm improves the efficiency and success ratio of path-finding, and significantly improves the overall performance of the algorithm. However, this algorithm still has its disadvantages. Each time the algorithm looks for bearer nodes and forwarding nodes, it does not select the optimal node among all the selectable objects. In addition, it does not consider the load balance.

In this section, we propose an improved direction-guided greedy (DGG) algorithm. The DGG algorithm will compare many optional objects when selecting the bearer and forwarding nodes and choose the one with the best optimization objective. In addition, the DGG algorithm has a load balancing strategy. The specific operation is as follows.

We assume that the current path is trying to extend to node n and go through link l, thus introducing new power consumption costs P_n^{old}, P_l^{old}. When load balancing is not considered, the total new power consumption P_{add} introduced is the sum of the two. After load balancing is introduced, a penalty factor must be added according to the current load ratio of the node and the link, as shown in Equations (24) and (25):

$$P_n^{new} = A_n^{u_n} \cdot P_n^{old}, \tag{24}$$

$$P_l^{new} = A_l^{u_l} \cdot P_l^{old}, \tag{25}$$

where A_n and A_l are constants greater than 1, and u_n and u_l represent the load ratios of nodes and links, respectively. The total new power consumption introduced is calculated according to Equation (26):

$$P_{add} = P_n^{new} + P_l^{new} = A_n^{u_n} \cdot P_n^{old} + A_l^{u_l} \cdot P_l^{old}. \tag{26}$$

A_n and A_l represent the trade-off in the DGG algorithm between load balancing and total network power consumption. In the extreme case, when both are taken as 1, the algorithm degenerates to a version that does not consider load balancing. The larger the values of A_n and A_l, the more emphasis the algorithm has on load balancing. In addition, A_n and A_l can also be used to weigh the impact of the new link power consumption and the new node power consumption on the total new power consumption. When the new power consumption introduced by one side is too large, the corresponding penalty factor can be appropriately reduced, which can avoid its dominance and hinder the adjustment of the load ratio of the other side. Pseudocode for the DGG algorithm is given in Algorithm 2.

Algorithm 2. DGG Algorithm

Input: Underlying network $G_s = (N, L, A^N, A^L)$;
Service function requests $G_v = (V, E, R^v, R^E)$

Output: node, link mapping pair of SFC on the underlying network

Extract destination node t of the SFC and use the Bellman-Ford algorithm to update the shortest hops from other nodes to this node in the underlying network;
According to the SFC, do not initialize the underlying physical node; initialize the VNF queue to be mapped, the current node as SFC source node, and the process value according to the current node hop count, and map the number of VNFs;
while VNF queue to be mapped is not empty, **do**
if process = 1, **then**
if the current node can map the current VNF, **then**
Establish the mapping relationship between the current node and VNF;
if the current VNF is not the last VNF, **then**
Find the lower node with the smallest value of P_{add} to carry the next VNF; otherwise find the peer node with the smallest value of P_{add}; otherwise find the forwarding nodes with the smallest value of P_{add} in peer nodes; otherwise find the forwarding nodes with the smallest value of P_{add} in lower nodes, otherwise find the higher node with the smallest value of P_{add}; otherwise find the forwarding node with the smallest value of P_{add} in higher nodes to carry the next VNF;

9: If not, find a node satisfying conditions to carry the next VNF, the SFC is rejected and the program ends; otherwise, the selected node is set as the next current node. If its number of hops is smaller than the number of VNFs to be mapped, process = 2, otherwise process = 1;
10: **end if**
11: **if** the current VNF is the last VNF, **then**
12: Find the optimal route from the current node to the destination node based on the accumulated value of P_{add};
13: If path-finding is not successful, reject the SFC and the program ends;
14: **end if**
15: **end if**
16: **if** current node cannot map current VNF, **then**
17: Find a lower node with the smallest value of P_{add} to carry the current VNF; otherwise find the peer node with the smallest value of P_{add}; otherwise find the forwarding node with the smallest value of P_{add} in peer nodes; otherwise find the forwarding node with the smallest value of P_{add} in lower nodes; otherwise find the higher node with the smallest value of P_{add}; otherwise ind the forwarding node with the smallest value of P_{add} in higher nodes to carry the current VNF;
18: If not, find a node satisfying conditions to carry the current VNF, the SFC is rejected and the program ends; otherwise, the selected node is set as the next current node. If its number of hops is smaller than the number of VNFs to be mapped, process = 2, otherwise process = 1;
19: **end if**
20: **end if**
21: **if** process = 2; **then**
22: **if** the current node can map the current VNF, **then**
23: Establish the mapping relationship between the current node and VNF;
24: Find the higher node with the smallest value of P_{add} to carry the next VNF; otherwise find the forwarding node with the smallest value of P_{add} in higher nodes; otherwise find the peer node with the smallest value of P_{add}; otherwise find the forwarding node with the smallest value of P_{add} in peer nodes; otherwise find the lower node with the smallest value of P_{add}; otherwise find the forwarding node with the smallest value of P_{add} in lower nodes to carry the next VNF;
25: If not, find a node satisfying conditions to carry the next VNF, the SFC is rejected and the program ends; otherwise, the selected node is set as the next current node. If its number of hops is smaller than the number of VNFs to be mapped, process = 2, otherwise process = 1;
26: **end if**
27: **if** current node cannot map current VNF, **then**
28: Find the higher node with the smallest value of P_{add} to carry the current VNF; otherwise find the forwarding node with the smallest value of P_{add} in higher nodes; otherwise find the peer node with the smallest value of P_{add}; otherwise find the forwarding node with the smallest value of P_{add} in peer nodes; otherwise find the lower node with the smallest value of P_{add}; otherwise find the forwarding node with the smallest value of P_{add} in lower nodes to carry the current VNF;
29: If not, find a node satisfying conditions to carry the current VNF, the SFC is rejected and the program ends; otherwise, the selected node is set as the next current node. If its number of hops is smaller than the number of VNFs to be mapped, process = 2, otherwise process = 1;
30: **end if**
31: **end if**
32: **end while**

The DGG algorithm improves the deficiencies of the DGFF algorithm. When selecting the bearer node and the forwarding node, the algorithm optimizes the selection and reduces the total power consumption. While introducing the load balancing strategy, the DGG algorithm can reasonably channel user traffic to the entire network, which reduces the average load ratio of the network and makes the network more robust.

5. Simulation Results and Analysis

The simulation platform in this paper is based on a notebook computer with Windows 64-bit professional system to conduct the simulation. The CPU of the desktop computer is the Intel (R) Core (TM) i5-3230M, and the memory size of the notebook computer is 4G RAM. We use the Visual Studio 2013 and C++ programming language to perform the simulation.

In this paper, we refer to the reference [16] for the setting of the simulation parameters. We set the number of the nodes in underlying network topology to 25 and the node average degree to 3. The node CPU capacity is set to a random number from a uniform distribution between (50, 59). The forwarding power is set to a random number from a uniform distribution between (2, 3) in watts (W). The link bandwidth capacity is set to a random number from a uniform distribution between (100, 119) in Gbps. The link length is set to a random number from a uniform distribution between (100, 119) in km. The power of the optical transceiver on the link is set to a random number from a uniform distribution between (5, 8) W.

The VNF set size is set to 20, and the CPU demand is set to a random number from a uniform distribution between (5, 8). The length of SFC can be 4, 5 or 6. The SFC bandwidth requirement is set to a random number from a uniform distribution between (10, 14) in Gbps. The values of survival time follow the Poisson distributions with mean values of 30, 35, and 40; the unit is the number of SFCs that arrive later.

In [8], the author proposed the heuristic algorithm ENSF for cloud application SFC mapping. The algorithm is based on the greedy algorithm; according to the optimization objective function, the object in the set of selectable objects that make the target optimal is found.

Figure 5 shows the number of cloud application SFCs carried in the underlying network of each SFC for varied circumstances. The results show that both the DGFF algorithm and the DGG algorithm are significantly better than the ENSF algorithm in terms of the number of SFCs and the average power consumption. This gap is due to the "direction guidance", which achieves efficient and accurate path-finding, saving many unnecessary expenses. This gap tends to decrease as network load increases, which has also been observed in some previous studies [38]. When the load is heavy, the higher acceptance ratio of the DGFF and DGG algorithms occupies a larger amount of physical resources, and the subsequent SFC must choose the detour to successfully map. DGFF and DGG do not have much difference in the number of mapped SFCs (the DGG algorithm is slightly better), which is consistent with their similar acceptance ratio.

Figure 6 shows the average power consumption of each SFC for varied circumstances. From the perspective of average power consumption, the DGG algorithm is obviously better than the DGFF algorithm, but this difference is far smaller than the gap with the ENSF algorithm. This is mainly because the sum of the local optimum searched by the DGG algorithm is not necessarily the global optimum. Therefore, from a statistical perspective, although the DGG algorithm is superior to the DGFF algorithm, it is not significantly so. This is primarily because there is no essential difference between the two algorithms. When the network load becomes heavy and path-finding becomes difficult, this difference becomes even smaller.

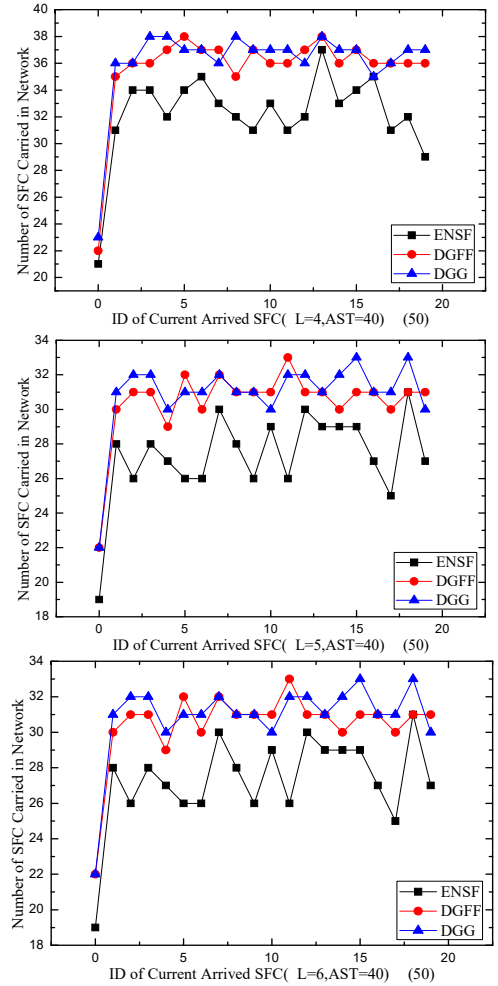


Figure 5. Comparison of the number of SFCs carried in the network. ENSF, energy-efficient next-hop selection; DGFF, direction-guided FirstFit; DGG, direction-guided greedy.

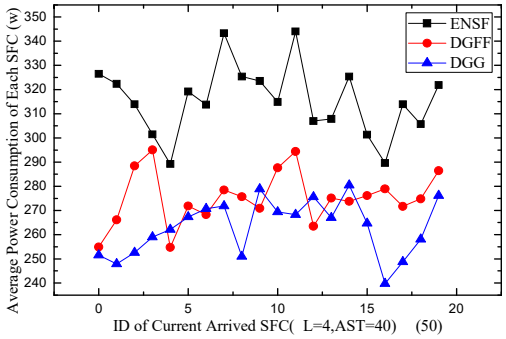


Figure 6. *Cont.*

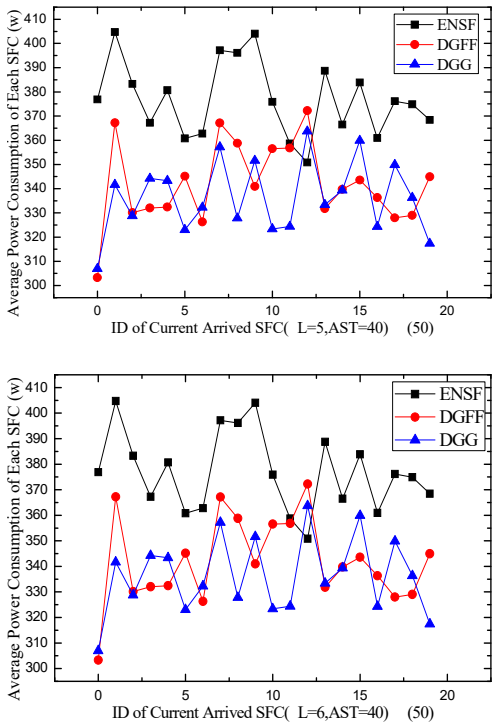


Figure 6. Comparison of the average power consumption of each SFC.

Figure 7 shows the simulation results of the average node load ratio and the average link load ratio of the network for various conditions. From the figure, the average node load ratios of the DGFF algorithm and the DGG algorithm are slightly better than that of the ENSF algorithm, but there is almost no difference.

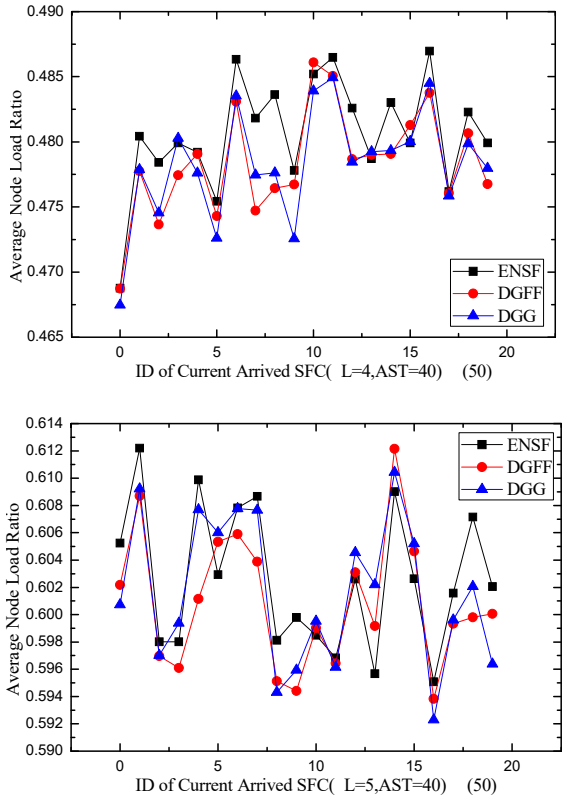


Figure 7. *Cont.*

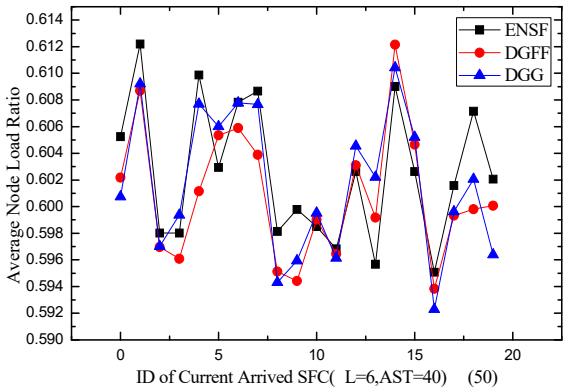
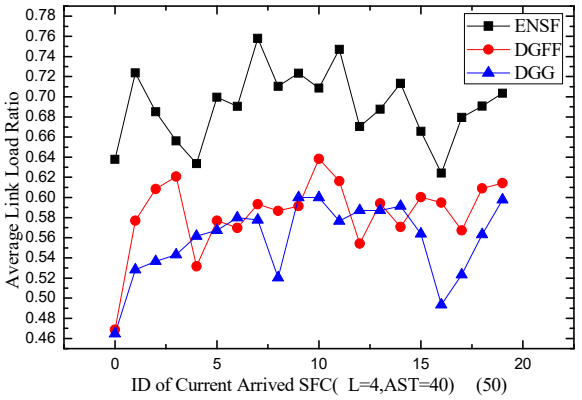


Figure 7. Comparison of average node load ratio in network.

Figure 8 shows the simulation results of the average link load ratio of the network for various conditions. From the perspective of the average link load ratio, DGFF and DGG have a larger improvement than ENSF. This is because the efficient path-finding method saves a large amount of extra link bandwidth overhead. The improvement range decreases as the load increases. In addition, the DGG algorithm has a certain improvement over the DGFF algorithm, and the effect is particularly stable and obvious at light loads. When the load is heavy, the crossover frequency of the two curves increases. However, the average link load ratio of the DGG algorithm is usually still lower than that of the DGFF algorithm, which shows that the load balancing strategy introduced in the DGG algorithm plays a certain role, but its effect will be weakened when the network approaches the bearer limit.



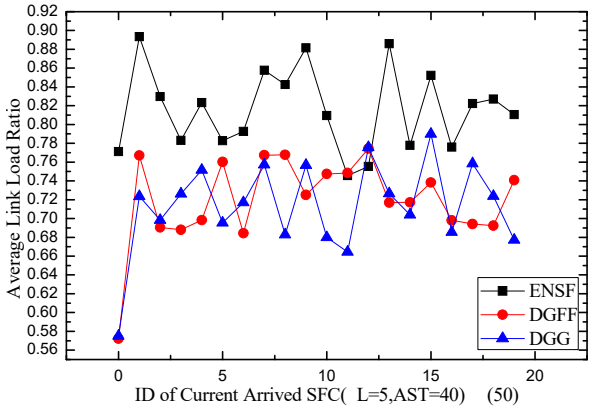


Figure 8. *Cont.*

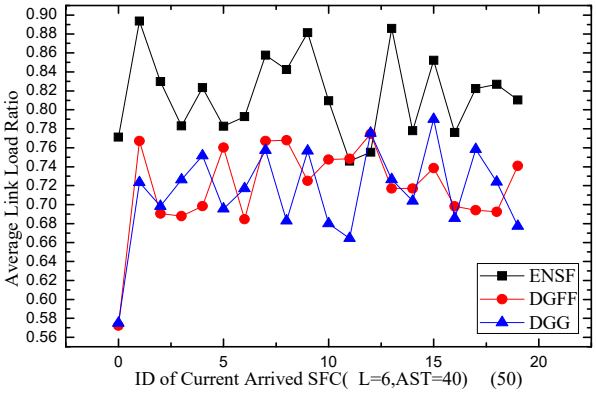


Figure 8. Comparison of average link load ratio in network.

Figure 9 shows the simulation results of the average mapped cost for various conditions. From the perspective of the average mapped cost, DGFF and DGG are superior to ENSF, because the directional guidance strategy substantially improves the efficiency of the mapped path and shortens the length of the mapped path. The strategy conserves the system resources and fully utilizes the resources occupied by the system. However, the cost advantage of the algorithm will decrease with an increase in the network load. When the network is overloaded, the system capacity becomes a major bottleneck, which directly decreases the cost of the optimization space and the performance difference of the algorithm.

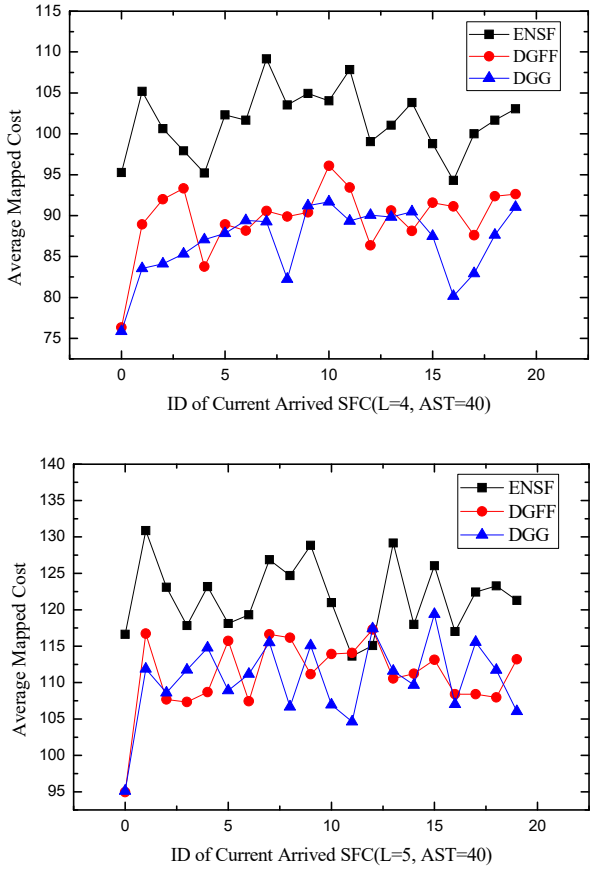


Figure 9. *Cont.*

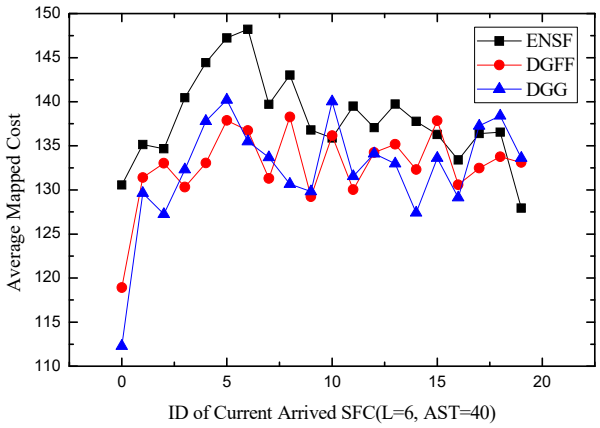


Figure 9. Comparison of average mapped cost.

Figure 10 shows the simulation results of the cloud application SFC acceptance ratio for each algorithm with the same SFC length (L) and a different average survival time (AST). For any circumstance, the SFC acceptance ratio of the DGFF algorithm and the DGG algorithm is significantly higher than that of the ENSF algorithm, and the maximum increase is approximately 15%. In most cases, the DGG algorithm is slightly better. As the network load increases, the acceptance ratio of each algorithm gradually decreases. When {L = 6, AST = 40}, the total acceptance ratio of the ENSF decreases below 60%, whereas the acceptance ratio of the other two algorithms remains 70%.

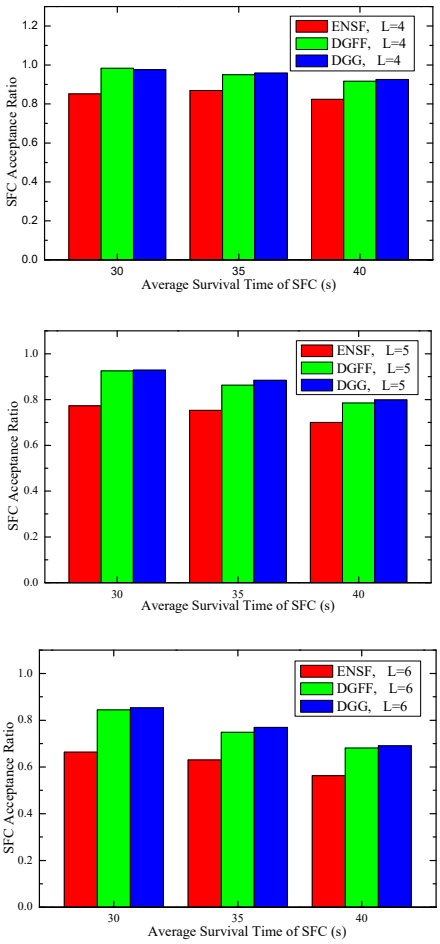


Figure 10. Comparison of SFC acceptance ratio for varied average survival time.

Figure 11 shows the SFC acceptance ratio when the average survival time is the same and L is different. The same features are observed in Figure 10.

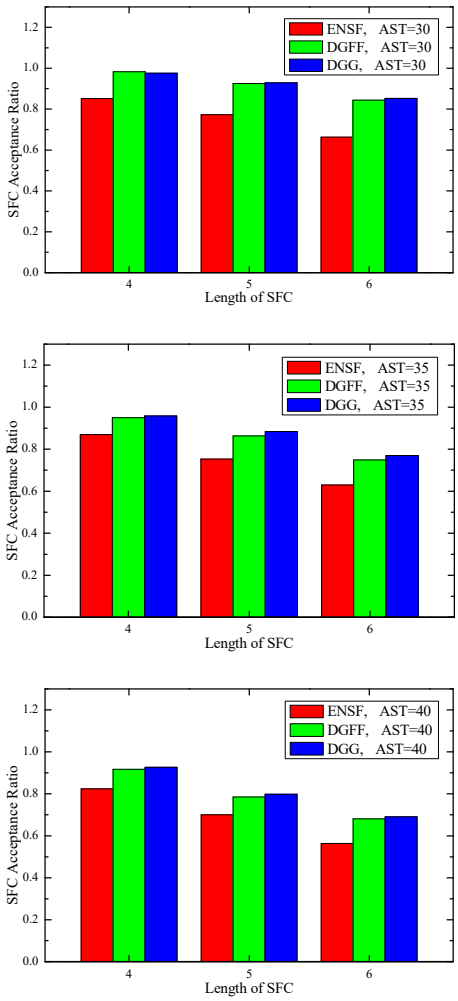


Figure 11. Comparison of SFC acceptance ratio for varied SFC length.

Figure 12 shows the average mapped path length of the SFC for each algorithm for different conditions. When the network load is light, the average mapped path length generated by the ENSF algorithm is approximately 1.5 hops more than that of the DGFF algorithm and the DGG algorithm. This gap tends to decrease with an increase in the network load. When {L = 6, AST = 40}, approximately 0.5 hops exist. Figure 12 shows the accuracy of the SFC mapped path after "direction guidance" is employed and explains why the DGFF and DGG algorithms have a sizable improvement in the SFC acceptance ratio.

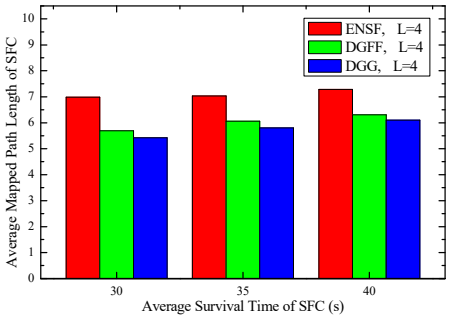


Figure 12. *Cont.*

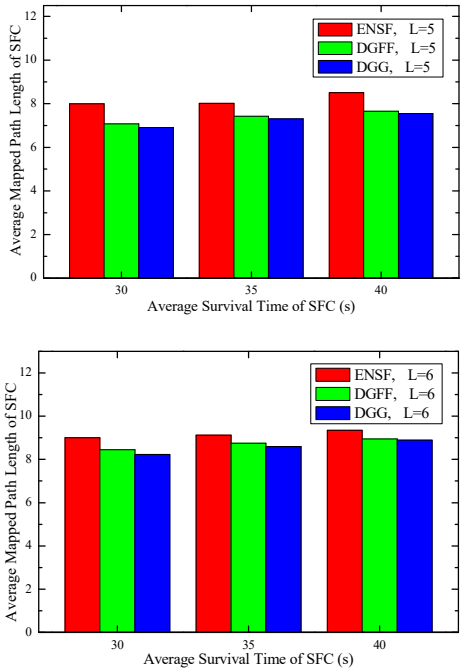


Figure 12. Comparison of the average mapped path length of SFC.

Figure 13 shows the simulation results of the average running time for various conditions. The DGFF algorithm and the DGG algorithm need more running time than the ENSF algorithm, due to the introduction of a directional guidance strategy in the DGFF algorithm and the DGG algorithm. The running time of the DGFF algorithm and the DGG algorithm is greater than the ENSF algorithm by approximately 50–100%.

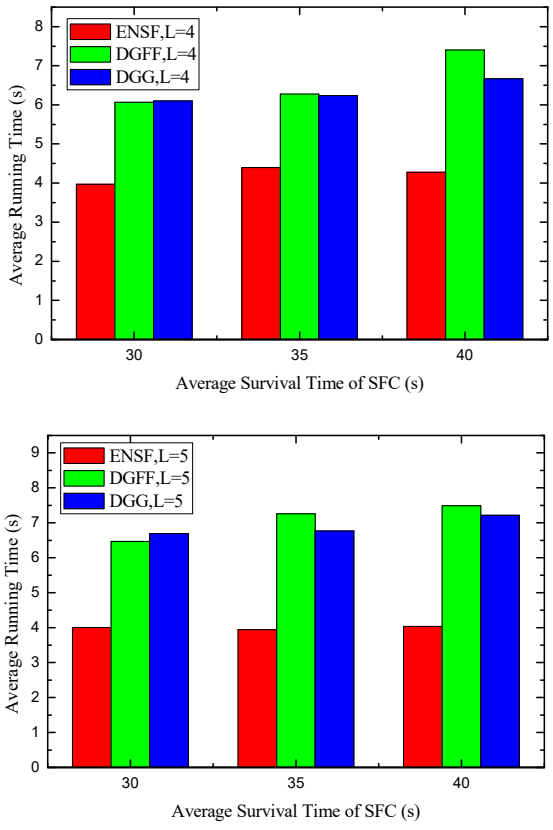


Figure 13. *Cont.*

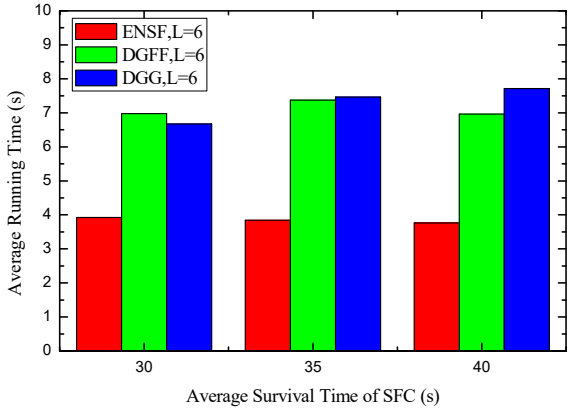


Figure 13. Comparison of average running time.

6. Conclusions and Future Work

This paper focuses on energy optimization of the cloud application SFC deployment in a single-domain network. We first briefly introduced the research background and related concepts of this issue and cited some existing studies. Next, we showed the various models used in the paper. Then, we designed the algorithm for the proposed problem. We first described the ENSF algorithm proposed by previous researchers in detail and analyzed the main problems of this algorithm. For the defects of this algorithm, we introduced "direction guidance" and proposed the DGFF algorithm and the DGG algorithm. The two algorithms introduce an efficient and accurate path-finding strategy, which significantly improves the performance of various aspects of the algorithm. The DGG algorithm also introduces load balancing, which reduces network bottlenecks and links and improves network stability. We simulated each algorithm in terms of the acceptance ratio, the average power consumption, the average mapped cost, the average running time of the algorithm, the average node load ratio and the average link load ratio of the network for provisioning the cloud applications. The simulation results show that the DGFF algorithm and the DGG algorithm achieve a qualitative leap in each performance index compared with the ENSF algorithm. Compared with the ENSF algorithm, the SFC acceptance ratio, the power consumption, the average link load ratio and the average mapped cost of our proposed algorithms improve by approximately 15–20%, and the average mapped path length of SFC reduces by approximately 1.5 hops. However, the running time of the algorithm increases by approximately 50–100%.

In the future, we will investigate objectives that extend beyond power consumption in cloud application SFC deployment. We will combine machine learning with traditional algorithms to design better algorithms and apply them to solve other problems in virtual networking [39–43].

Author Contributions: Conceptualization, J.S.; Methodology, Y.C.; Software, M.D.; Writing-Original Draft Preparation, W.Z.; Writing-Review & Editing, A.K.S.; Supervision, G.S.; Project Administration, H.H.

Funding: This research was funded by National Natural Science Foundation of China grant number [61571098], Fundamental Research Funds for the Central Universities grant number [ZYGX2016J217], and the 111 Project grant number [B14039].

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Sun, G.; Liao, D.; Zhao, D.; Sun, Z.; Chang, V. Towards provisioning hybrid virtual networks in federated cloud data centers. *Future Gener. Comput. Syst.* **2018**, *87*, 457–469. [CrossRef]
2. Sun, G.; Li, Y.; Li, Y.; Liao, D.; Chang, V. Low-Latency Orchestration for Workflow-Oriented Service Function Chain in Edge Computing. *Future Gener. Comput. Syst.* **2018**, *85*, 116–128. [CrossRef]
3. Kaplan, J.; Forrest, W.; Kindler, N. *Revolutionizing Data Center Energy Efficiency*; Technical Report; McKinsey & Company: New York, NY, USA, 2008.

4. Berl, A.; Gelenbe, E.; Di Girolamo, M.; Giuliani, G.; De Meer, H.; Dang, M.Q.; Pentikousis, K. Energy-efficient cloud computing. *Comput. J.* **2009**, *53*, 1045–1051. [CrossRef]
5. Qureshi, A.; Weber, R.; Balakrishnan, H.; Guttag, J.; Maggs, B. Cutting the electric bill for internet-scale systems. *ACM SIGCOMM Comput. Commun. Rev.* **2009**, *39*, 123–134. [CrossRef]
6. Chih-Lin, I. Green Evolution of Mobile Communications (CMCC Perspective). 2012. Available online: http://www.greentouch.org/uploads/docments/ChihLin_I%20%20TIA%20Green%20from%20a%20Service%20Provider%20Perspective.pdf (accessed on 1 July 2012).
7. Chiaraviglio, L.; Mellia, M.; Neri, F. Minimizing ISP network energy cost: Formulation and solutions. *IEEE ACM Trans. Netw.* **2012**, *20*, 463–476. [CrossRef]
8. Tajiki, M.M.; Salsano, S.; Shojafar, M.; Chiaraviglio, L.; Akbari, B. Energy-efficient Path Allocation Heuristic for Service Function Chaining. In Proceedings of the 2018 21th Conference on Innovations in Clouds, Internet and Networks (ICIN), Paris, France, 19–22 February 2018; pp. 20–22.
9. Yu, M.; Yi, Y.; Rexford, J.; Chiang, M. Rethinking virtual network embedding: Substrate support for path splitting and migration. *ACM SIGCOMM Comput. Commun. Rev.* **2008**, *38*, 17–29. [CrossRef]
10. Houidi, I.; Louati, W.; Zeghlache, D. A distributed virtual network mapping algorithm. In Proceedings of the IEEE International Conference on Communications, Beijing, China, 19–23 May 2008; pp. 5634–5640.
11. Lischka, J.; Karl, H. A virtual network mapping algorithm based on subgraph isomorphism detection. In Proceedings of the 1st ACM Workshop on Virtualized Infrastructure Systems and Architectures, Barcelona, Spain, 17 August 2009; pp. 81–88.
12. Cheng, X.; Su, S.; Zhang, Z.; Wang, H.; Yang, F.; Luo, Y.; Wang, J. Virtual network embedding through topology-aware node ranking. *ACM SIGCOMM Comput. Commun. Rev.* **2011**, *41*, 38–47. [CrossRef]
13. Chowdhury, M.; Rahman, M.R.; Boutaba, R. Vineyard: Virtual network embedding algorithms with coordinated node and link mapping. *IEEE/ACM Trans. Netw.* **2012**, *20*, 206–219. [CrossRef]
14. Cheng, X.; Su, S.; Zhang, Z.; Shuang, K.; Yang, F.; Luo, Y.; Wang, J. Virtual network embedding through topology awareness and optimization. *Comput. Netw.* **2012**, *56*, 1797–1813. [CrossRef]
15. Zhang, Z.; Cheng, X.; Su, S.; Wang, Y.; Shuang, K.; Luo, Y. A unified enhanced particle swarm optimization-based virtual network embedding algorithm. *Int. J. Commun. Syst.* **2013**, *26*, 1054–1073. [CrossRef]
16. Sun, G.; Li, Y.; Liao, D.; Chang, V. Service Function Chain Orchestration across Multiple domains: A Full Mesh Aggregation Approach. *IEEE Trans. Netw. Serv. Manag.* **2018**, *15*, 1175–1191. [CrossRef]
17. Sun, G.; Chang, V.; Yang, G.; Liao, D. The Cost-efficient Deployment of Replica Servers in Virtual Content Distribution Networks for Data Fusion. *Inf. Sci.* **2018**, *432*, 495–515. [CrossRef]
18. Sun, G.; Liao, D.; Zhao, D.; Xu, Z.; Yu, H. Live Migration for Multiple Correlated Virtual Machines in Cloud-based Data Centers. *IEEE Trans. Serv. Comput.* **2018**, *11*, 279–291. [CrossRef]
19. Sun, G.; Liao, D.; Bu, S.; Yu, H.; Sun, Z.; Chang, V. The Efficient Framework and Algorithm for Provisioning Evolving VDC in Federated Data Centers. *Future Gener. Comput. Syst.* **2017**, *73*, 79–89. [CrossRef]
20. Sun, G.; Liao, D.; Anand, V.; Zhao, D.; Yu, H. A New Technique for Efficient Live Migration of Multiple Virtual Machines. *Future Gener. Comput. Syst.* **2016**, *55*, 74–86. [CrossRef]
21. Pham, C.; Tran, N.H.; Ren, S.; Saad, W.; Hong, C.S. Traffic-aware and Energy-efficient vNF Placement for Service Chaining: Joint Sampling and Matching Approach. *IEEE Trans. Serv. Comput.* **2017**, 1–14. [CrossRef]
22. Yang, K.; Zhang, H.; Hong, P. Energy-Aware Service Function Placement for Service Function Chaining in Data Centers. In Proceedings of the 2016 IEEE Global Communications Conference, Washington, DC, USA, 4–8 December 2017; pp. 1–6.
23. Racheg, W.; Ghrada, N.; Zhani, M.F. Profit-driven resource provisioning in NFV-based environments. In Proceedings of the IEEE International Conference on Communications (ICC), Paris, France, 21–25 May 2017.
24. Zhang, Z.; Su, S.; Zhang, J.; Shuang, J.; Xu, P. Energy aware virtual network embedding with dynamic demands. In Proceedings of the IEEE International Conference on Communications, London, UK, 8–12 June 2015; pp. 5386–5391.
25. Zhang, Z.; Liu, A.X.; Cheng, X.; Wang, Y.; Zhao, X. Energy-aware virtual network embedding. *IEEE ACM Trans. Netw.* **2014**, *22*, 1607–1620.
26. Bouet, M.; Leguay, J.; Combe, T. Cost-based placement of vDPI functions in NFV infrastructures. *Int. J. Netw. Manag.* **2015**, *25*, 490–506. [CrossRef]

27. Bagaa, M.; Taleb, T.; Ksentini, A. Service-aware network function placement for efficient traffic handling in carrier cloud. In Proceedings of the 2014 IEEE Wireless Communications and Networking Conference (WCNC), Istanbul, Turkey, 6–9 April 2014; pp. 2402–2407.
28. Mehraghdam, S.; Keller, M.; Karl, H. Specifying and placing chains of virtual network functions. In Proceedings of the 2014 IEEE 3rd International Conference on Cloud Networking (CloudNet), Luxembourg, 8–10 October 2014; pp. 7–13.
29. Mijumbi, R.; Serrat, J.; Gorricho, J.L. On the energy efficiency prospects of network function virtualization. *arXiv* **2015**, arXiv:1512.00215.
30. Zhang, B.; Zhang, P.; Zhao, Y.; Wang, Y.; Luo, X.; Jin, Y. Co-Scaler: Cooperative scaling of software-defined NFV service function chain. In Proceedings of the IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN), Palo Alto, CA, USA, 7–10 November 2016; pp. 33–38.
31. Bao, N.H.; Luo, D.Y.; Chen, J.B. Reliability Threshold Based Service Bandwidth Recovery Scheme for Post-Disaster Telecom Networks. *Opt. Fiber Technol.* **2018**, *45*, 81–88. [CrossRef]
32. Hou, W.; Ning, Z.; Guo, L.; Chen, Z.; Obaidat, M.S. Novel Framework of Risk-Aware Virtual Network Embedding in Optical Data Center Networks. *IEEE Syst. J.* **2017**, *12*, 2473–2482. [CrossRef]
33. Hou, W.; Ning, Z.; Guo, L.; Obaidat, M.S. Service Degradability Supported by Forecasting System in Optical Data Center Networks. *IEEE Syst. J.* **2018**, 1–12. [CrossRef]
34. Hou, W.; Ning, Z.; Guo, L. Green Survivable Collaborative Edge Computing in Smart Cities. *IEEE Trans. Ind. Inform.* **2018**, *14*, 1594–1605. [CrossRef]
35. Hou, W.; Ning, Z.; Guo, L.; Zhang, X. Temporal, Functional and Spatial Big Data Computing Framework for Large-Scale Smart Grid. *IEEE Trans. Emerg. Top. Comput.* **2017**. [CrossRef]
36. Fan, X.; Weber, W.D.; Barroso, L.A. Power provisioning for a warehouse-sized computer. *ACM SIGARCH Comput. Archit. News* **2007**, *35*, 13–23. [CrossRef]
37. Cavdar, C.; Buzluca, F.; Wosinska, L. Energy-Efficient Design of Survivable WDM Networks with Shared Backup. In Proceedings of the IEEE Global Telecommunications Conference, Miami, FL, USA, 6–10 December 2010; pp. 1–5.
38. Botero, J.F.; Hesselbach, X.; Duelli, M.; Schlosser, D.; Fischer, A.; Meer, H.D. Energy efficient virtual network embedding. *IEEE Commun. Lett.* **2012**, *16*, 756–759. [CrossRef]
39. Chen, X.; Li, Z.; Zhang, Y.; Long, R.; Yu, H.; Du, X.; Guizani, M. Reinforcement Learning based QoS/QoE-aware Service Function Chaining in Software-Driven 5G Slices. *Trans. Emerg. Telecommun. Technol.* **2018**, 1–14. [CrossRef]
40. Sun, G.; Anand, V.; Liao, D.; Lu, C.; Zhang, X.; Bao, N.H. Power-efficient provisioning for online virtual network requests in cloud-based data centers. *IEEE Syst. J.* **2015**, *9*, 427–441. [CrossRef]
41. Chen, X.; Wu, T.; Xie, L. The Declarative and Reusable Path Composition for Semantic Web-Driven SDN. *IEICE Trans. Commun.* **2018**, *101*, 816–824. [CrossRef]
42. Sun, G.; Li, Y.; Yu, H.; Vasilakos, A.V.; Du, X.; Guizani, M. Energy-efficient and Traffic-aware Service Function Chaining Orchestration in Multi-Domain Networks. *Future Gener. Comput. Syst.* **2018**, *91*, 347–360. [CrossRef]
43. Chen, X.; Wu, J.; Wu, T. The Top-K QoS-aware Paths Discovery for Source Routing in SDN. *KSII Trans. Internet Inf. Syst.* **2018**, *12*, 2534–2553.