Efficient VNF Migration and Recovery in Edge Computing Environments

Yu Chen[†], Sheng Zhang[†*], Yanchao Zhao[‡], Zhuzhong Qian[†], Yu Liang[†], Jidong Ge[†], Sanglu Lu[†]

[†]State Key Lab. for Novel Software Technology, Nanjing University

[‡]Nanjing University of Aeronautics and Astronautics

sheng@nju.edu.cn

Abstract

The combination of network function virtualization (NFV) and software defined networking (SDN) contributes to making the network more efficient, and it promises to guarantee high flexibility and low cost in network operations. Meanwhile, the continuous development of mobile edge computing (MEC) makes it possible for the edge of mobile network to run cloud computing tasks. However, edge computing environments are dynamic, and VNF demands for computing resources usually fluctuate over time. Thus, when the demand of a VNF cannot be satisfied, we need to migrate the VNF so as to meet its demand and retain the network performance. Our paper focuses on efficiently migrating VNFs (MV) and recovering connections (RC) after migration, such that the migration and connection paths can meet the bandwidth requirement for data transmission. We prove that both of MV and RC are NP-complete. We present efficient algorithms to tackle them. Extensive simulations show that the proposed algorithms are efficient and effective.

Motivation

Mobile edge computing (MEC) enables the cloud computing capabilities at the edge of the mobile network, which greatly lowers the delay in data transmission. Virtual network functions (VNFs) can be dynamically deployed, migrated, and removed.

Due to the dynamic nature of edges, the network changes from time to time. For instance, the topology of the network may change, the resource capacity of a specific machine may vary at some point, and the bandwidth between two machines may increase or decrease due to different traffic situations. In the daytime, some VNFs may require more computing resources due to heavy traffic situations, and they need to be migrated to the machines with massive resources in central network nodes; in the night, due to the decreasing demand for computing resources, some VNFs need to be migrated back to edge network nodes with fewer resources, such that the machines with massive resources can be vacated for sudden large tasks. Therefore, we need to efficiently complete the migration of the VNF chains, as well as the task of VNF connection recovery after migration.

In this paper, we use 'to-be-migrated VNFs' to represent the VNFs which we need to migrate. We consider the problem of migrating multiple VNFs to new physical machines, as the to-be-migrated VNFs' demands for computing resources fluctuate due to different traffic situations. As a first step, to initiate a meaningful study, this paper narrows the scope of this problem to a manageable extent: we assume the network node to which each VNF is migrated is known in advance; thus, we focus on selecting the migration paths that meet the bandwidth requirements for all to-be-migrated VNFs, such that the task of migrating VNFs can be completed within the specified time. After migration, we need to recover the connection between the migrated VNFs which are adjacent along a VNF chain. To prevent the traffic congestion in the network, we need to find out the connection paths that minimize bandwidth usage.

Our contributions in the paper are three-fold: (1) We introduce MVRC using graph theory and present a formal formulation. We provide the NP-completeness results of both MV and RC; (2) We propose efficient algorithms for the MVRC problem; (3) Simulations are conducted to confirm the efficiency and effectiveness of the proposed algorithms.

Formulation

Migrating VNFs (MV). We assume that there are c VNF chains to be migrated, and the jth ($j = 1, 2, .., c$) VNF chain has C_j to-be-migrated VNFs. Thus, the total number of to-be-migrated VNFs is denoted as $n = \sum_{j=1}^{c} C_j$. Besides, the ith ($i = 1, 2, ..., n$) to-be-migrated VNF is denoted as f_i, and we migrate it from the current node $S_{M,i}$ to the destination node $D_{M,i}$, which is specified in advance. In the cloud-based edge network, there are $r_{M,i}$ paths between $S_{M,i}$ and $D_{M,i}$, and they are $R^1_{M,i}, R^2_{M,i}, ..., R^{r_{M,i}}_{M,i}$. We need to select one of them, e.g., $R^k_{M,i}$ ($k = 1, 2, ..., r_{M,i}$) as the migration path $P_{M,i}$ for f_i. We define $BW(R^k_{M,i})$ as the bandwidth of $R^k_{M,i}$, which equals the minimum bandwidth of the links along $R^k_{M,i}$. We allocate $B_{M,i}$ units of bandwidth for $P_{M,i}$, and $B_{M,i}$ is no more than $BW(R^k_{M,i})$. If multiple migration paths traverse the same link e, the total bandwidth allocated for them should not exceed B_e.

We start to migrate the n to-be-migrated VNFs at the same time, and all the migration tasks need to be finished within the specific time T. We assume the amount of the to-be-migrated data of f_i is d_i. Remember that the bandwidth we allocate for f_i along $P_{M,i}$ is $B_{M,i}$, so the time required for f_i's migration is $d_i/B_{M,i}$. All to-be-migrated VNFs need to be migrated to the destinations within time T, so we should select an appropriate migration path for each VNF. There is no motivation for each VNF to finish migration in the time less than T, because it will occupy extra bandwidth that might be utilized by other VNFs. Thus, the bandwidth requirement of $P_{M,i}$ is equal to d_i/T, i.e., $B_{M,i} = d_i/T$.

Recovering Connections (RC). After migration, we need to recover the connection path between each migrated VNF and its neighbor VNFs along the VNF chain. We assume that there are m connection paths to be recovered in total, and they are each selected between node $S_{C,1}$ and $D_{C,1}$, node $S_{C,2}$ and $D_{C,2}$, ..., node $S_{C,m}$ and $D_{C,m}$. In the cloud-based edge network, there are $r_{C,i}$ ($i = 1, 2, ..., m$) paths between $S_{C,i}$ and $D_{C,i}$, and they are $R^1_{C,i}, R^2_{C,i}, ..., R^{r_{C,i}}_{C,i}$. Similar to MV, we should select one of them, e.g., $R^k_{C,i}$ ($k = 1, 2, ..., r_{C,i}$) as the recovered connection path $P_{C,i}$ between node $S_{C,i}$ and $D_{C,i}$. The bandwidth requirement of the connection path between node $S_{C,i}$ and $D_{C,i}$ is $B_{C,i}$. Thus, we need to allocate $B_{C,i}$ units of bandwdith for $P_{C,i}$, and $B_{C,i}$ is no more than $BW(R^k_{C,i})$. If multiple recovered connection paths traverse the same link e, the total bandwidth allocated for them should not exceed B_e.

The total bandwidth allocated for the connection paths traversing the link e_i is denoted as t_{e_i}, and we define the bandwidth occupancy rate after recovering VNF connection as $\alpha = \max_{i=1}^{|E|} \frac{t_{e_i}}{B_{e_i}}$. It is easy to see α can reflect the traffic condition of the network after migration. Our objective is to minimize α and prevent the traffic congestion.

NP-completeness result. By reducing the Widest Pair of Paths Problem to RC and reducing the Widest Pair of Disjoint Paths Decoupled Problem to MV, we can both of MV and RC are NP-complete.

Preliminary Solutions

Heuristics for MV. We need to select n migration paths satisfying bandwidth requirements for all to-be-migrated VNFs in MV. To ensure that as many VNFs as possible can be migrated successfully, we migrate VNFs in the non-decreasing order of the bandwidth requirements and first select the shortest or the minimum wide-edge migration path, such that each migration path will use fewer links and the bandwidth of each link can be made the most use of on the whole.

Smallest path Heuristic (SH) algorithm. SH starts from the migration path $P_{M,i}$ with the smallest bandwidth requirement. SH utilizes the Dijkstra's Algorithm to calculate the shortest path sp between $S_{M,i}$ and $D_{M,i}$.

Minimum Wide-edge Heuristic (MWH) algorithm. MWH differs from SH only in that instead of using Dijkstra we use a similar algorithm that finds the minmum wide-edge path between two given vertices.

Heuristics for RC. To minimize the bandwidth occupancy rate and uniformly use all links' bandwidths, we select the migration paths that have large bandwidth capacity and traverse few links.

Widest Heuristic (WH) algorithm. WH starts from the path with the smallest bandwidth requirement, and checks whether there is any path that satisfies its demand.

Modified Shortest Heuristic (MSH) algorithm. Different from WH, MSH uses the Dijkstra's algorithm to find a suitable for a given vertices pair.

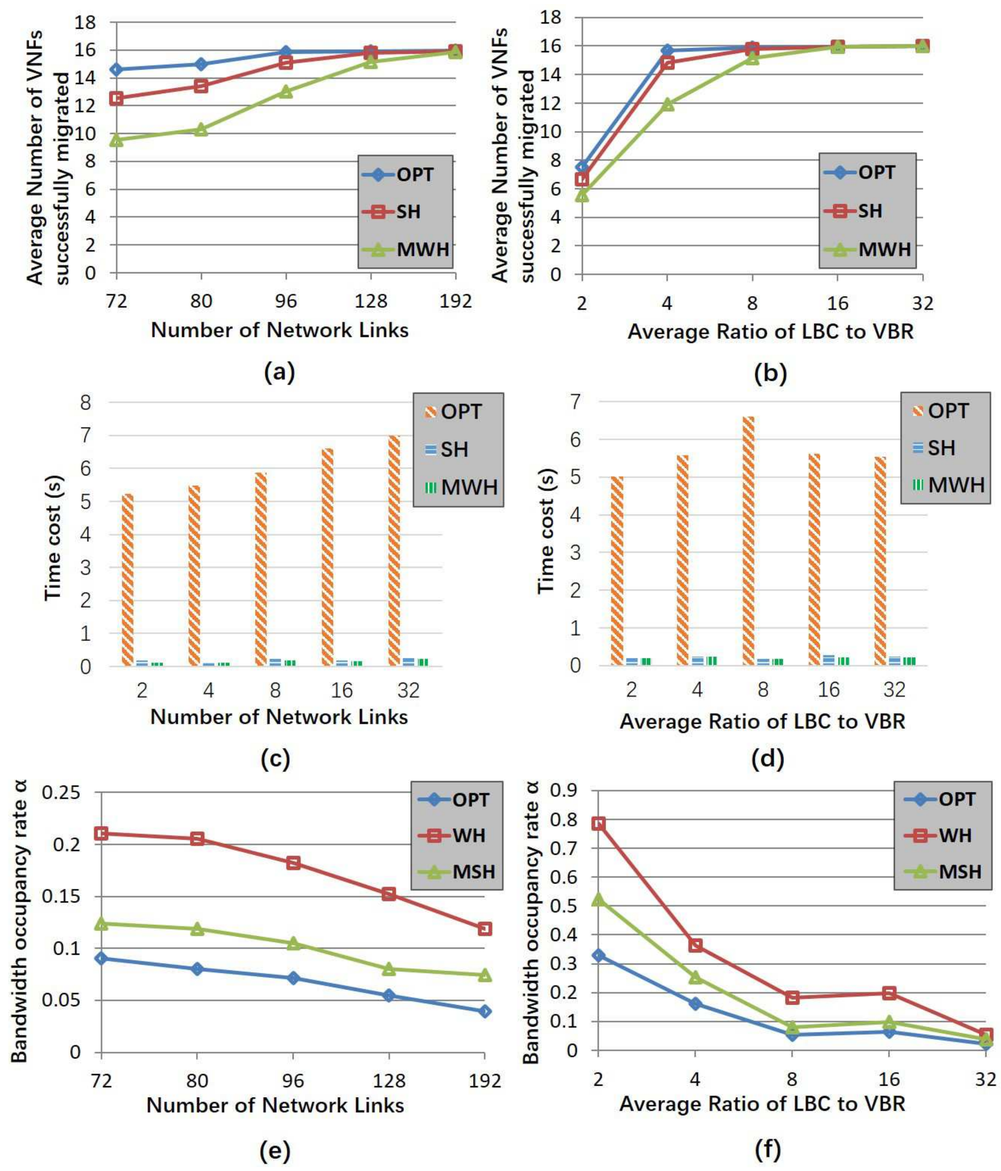

Figure1 Simulation results

Preliminary Results

We simulate the network using the Python package *NetworkX*. In general, the simulation results demonstrate the efficiency and effectiveness of the proposed algorithms, which are nearly as good as the optimal solution. For example, in Fig. 1(a)(b), we can see that the performance of SH and MWH is close to the optimal solution when the number of links is more than twice that of network nodes, i.e., 128. Besides, when the ratio of LBC to VBR in the network is larger than 8, SH and MWH also perform nearly as well as OPT. From Fig. 1(c)(d), we learn that the running time cost of SH and MWH is much less than that of OPT. Thus, when there are a number of links in the network and the ratio of LBC to VBR is not very low, our algorithms for MV can achieve good results.