Algorithms for Fault-Tolerant Placement of Stateful Virtualized Network Functions

Conference Paper · May 2018

CITATIONS
0

READS
105

7 authors, including:

Binxu Yang
University College London
6 PUBLICATIONS **38** CITATIONS
SEE PROFILE

Zichuan Xu
University College London
48 PUBLICATIONS **319** CITATIONS
SEE PROFILE

Wei Koong Chai
Bournemouth University
40 PUBLICATIONS **1,058** CITATIONS
SEE PROFILE

Weifa Liang
Australian National University
183 PUBLICATIONS **2,102** CITATIONS
SEE PROFILE

Algorithms for Fault-Tolerant Placement of Stateful Virtualized Network Functions

Binxu Yang[†], Zichuan Xu[‡], Wei Koong Chai[*†], Weifa Liang[¶], Daphné Tuncer[†], Alex Galis[†], and George Pavlou[†]
[†] Department of Electronic and Electrical Engineering, University College London, London, UK
[‡] School of Software, Dalian University of Technology, Dalian, China, 116621.
[*] Department of Computing & Informatics, Bournemouth University, UK
[¶] Research School of Computer Science, Australian National University, Canberra, ACT 2601, Australia
{binxu.yang.13, d.tuncer, a.galis, g.pavlou}@ucl.ac.uk, z.xu@dlut.edu.cn, wchai@bournemouth.ac.uk, wliang@cs.anu.edu.au

***Abstract*—Traditional network functions (NFs) such as firewalls are implemented in costly dedicated hardware. By decoupling NFs from physical devices, network function virtualization enables virtual network functions (VNF) to run in virtual machines (VMs). However, VNFs are vulnerable to various faults such as software and hardware failures. To enhance VNF fault tolerance, the deployment of backup VNFs in stand-by VM instances is necessary. In case of stateful VNFs, stand-by instances require constant state updates from active instances during its operation. This will guarantee a correct and seamless handover from failed instances to stand-by instances after failures. Nevertheless, such state updates to stand-by instances could consume significant network bandwidth resources and lead to potential admission failures for VNF requests. In this paper, we study the fault-tolerant VNF placement problem with the optimization objective of admitting as many requests as possible. In particular, the VNF placement of active/stand-by instances, the request routing paths to active instances, and state transfer paths to stand-by instances are jointly considered. We devise an efficient heuristic algorithm to solve this problem, and propose a bicriteria approximation algorithm with performance guarantees for a special case of the problem. Simulations with realistic settings show that our algorithms can significantly improve the request admission rate compared to conventional approaches.**

I. Introduction

Cloud service provides exploit different network functions (NFs), such as network address translation (NAT), firewall and deep packet inspection (DPI), to improve network performance and security. These NFs are embedded into dedicated hardware that are costly and difficult to reconfigure. The advent of Network Function Virtualization (NFV) provides a more flexible and inexpensive support of NFs compared to conventional hardware-based approaches [1]. Specifically, NFV decouples NFs from physical devices by implementing NFs as software running in virtual machines (VMs) in the form of virtualized network functions (VNFs). As such, VNFs can be instantiated on any data center (DC) with available computing resources. This flexibility in VNF instantiation further enables advanced VNF placement schemes [2], through which the cost and performance of NFs can be largely improved [3].

Despite the achieved flexibility, moving NFs from hardware to software poses grand concerns especially in terms of reliability. For instance, VNFs are software running in DCs, which are vulnerable to various problems such as misconfiguration, faulty VMs and software malfunctions [4]. In order to enhance VNF fault tolerance, backup VNFs that run in stand-by instances are required [5]. In case of failures, requests to stateless VNFs can be immediately redirected to one of their stand-by instances. In contrast, stateful VNFs generate states during traffic processing [6] that need to be transferred to stand-by instances in order to guarantee seamless request redirection. For instance, a stateful NAT VNF needs to maintain existing user connections to support its correct operation. If a NAT fails, the transient states created by the traffic itself have to be transferred to the backup NAT to avoid NAT disconnection. Given that such state transfers need to be continuously performed while active instances are in operation [7], [8], it could consume considerable network bandwidth resources, and lead to significant network link overheads. Furthermore, if the network path used for state transfers overlaps with the VNF request routing path, the active VNF instance's request admissions may fail due to delay violations caused by link congestion. As such, decisions regarding 1) the placement of active instances, 2) the placement of stand-by instances, 3) request routings, 4) the state transfer paths need to be jointly considered so that the number of admitted user requests can be maximized. In this paper, we study the *fault-tolerant stateful VNF placement problem*, whereby the aforementioned four decisions are jointly determined under DC computing and bandwidth resource constraints.

Providing efficient solutions to the fault-tolerant VNF placement problems poses several challenges. On one hand, as stated earlier, a naive solution that separately determines the instance locations and routings may result in network congestion and admission failures. It may also lead to significant network communication costs if the active/stand-by instances are placed with long network distance to the source and destination nodes of requests. On the other hand, the number and placements of stand-by instances directly influence the state update cost for VNFs. At the same time, the number of stand-by instances affects the robustness of the networks. Clearly, a higher number of stand-by instances indicates a higher degree of fault tolerance.

Previous studies on the fault-tolerant VNF placement problem have either focused on backup instances or stateless VNFs [5], [9], [10], [11], [12]. For example, Kanizo *et. al.* [10] investigated the planning-stage VNF backup instances (i.e., do not consider active instances) deployment problem while taking into account the failure probabilities of network nodes. Chantre *et. al.* [11] studied the placement problem of redundant stateless VNFs in LTE networks with a focus on deriving the

optimal number of VNFs to guarantee reliability. Carpio *et. al.* [5] investigated the joint active and backup stateless VNF placement problem, but did not consider request routing and VNF state transfers. To the best of our knowledge, this work is the first study that jointly considers stateful active/stand-by VNF placement, request routing and state transfers.

In the remainder of this paper, we first introduce the considered scenario and the related definitions in Section II. Then, we propose an efficient heuristic based on the joint availability of DC computing resources and the accumulative bandwidth resources of DC's inbound links in Section III. The proposed heuristic jointly computes the placement of both active and stand-by stateful VNF instances. For a special case of our problem without bandwidth constraint, we propose a $(2, 4+\epsilon)$ bicriteria approximation algorithm with proved approximation ratios on the achieved cost and maximum DC utilization in Sections IV. The proposed algorithm exploits an approach based on auxiliary graph that allows active/stand-by instances, request routings and state update paths to be jointly considered. The evaluation results presented in Section V suggest that the proposed algorithms significantly improve the request admission rate while reducing DCs' cost. At the same time, they outperform existing solutions that separately consider placements, routings and update paths. Concluding remarks are finally presented in Section VI.

II. Fault-tolerant VNF Placement Problem

A. System model

We model the network $G = (V \cup \mathcal{DC}, E)$ operated by a cloud service provider with a set V of switches, a set $\mathcal{DC}$ of DCs attached to V, and a set E of network links (see Fig 1). We follow the convention to assume that the number of DCs is far less than the number of switches. Each $DC_i \in \mathcal{DC}$ has computing resources $C(DC_i)$ that can be utilized to instantiate VNFs instances. A sequence of VNFs forms a *service chain*, denoted as SC, and *an instance of a service chain* is defined as an implementation of its specified VNFs in a VM. Given the computing capacity $C(DC_i)$ of DC_i , a limited number of instances of different service chains can be supported in each DC. Similarly, each link $e \in E$ has a capacity $B(e)$ of bandwidth resources that can be allocated to user requests. Without loss of generality, we assume that each DC and the switch node attached to it is connected by a high-speed optical cable with abundant network bandwidth (see Fig 1) so that the delay and communication cost incurred at these links can be considered as negligible. Furthermore, the transmission delay on each link $e \in E$ is denoted as d_e.

B. Requests for VNFs and service chains

We denote as $r_j = (s_j, t_j; SC_j, \rho_j, D_j)$ user request j. Each user request requires to be routed from a source node s_j to a destination node t_j at a given packet rate ρ_j within D_j time, such that its traffic passes through one instance of its required service chain SC_j.

Different user requests have different demands for SC, with each type of service chains having a different sequence of VNFs. Without loss of generality, we assume that the computing resource requested by an instance of service chain SC_j for processing the traffic of r_j is proportional to its packet rate, i.e., $\rho_j \cdot c_b$, where c_b is a given constant representing the amount of computing resources that is needed to process each unit packet rate. The total amount of computing resource allocated to all instances of service chains in a data center DC_i must not exceed its computing capacity $C(DC_i)$.

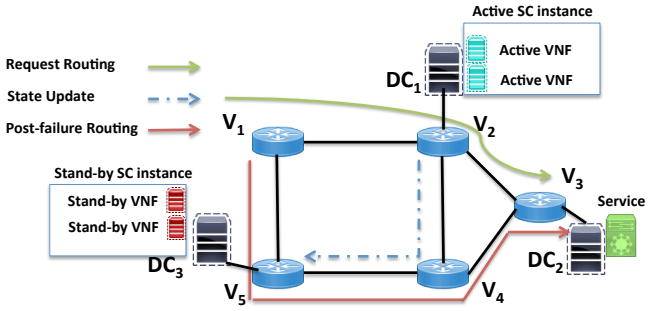


Fig. 1. An example of fault-tolerant placement problem in G with a set $\mathcal{DC} = \{DC_1, DC_2, DC_3\}$ connected by a set $V = \{v_2, v_3, v_5\}$ of switches.

The *end-to-end delay requirement* D_j of each request r_j specifies the maximum tolerable delay experienced by its traffic from its source node, s_j, to its destination node, t_j. It consists of the processing delay of service chain SC_j at a DC and the transfer delay on each link. Specifically, assuming an instance of SC_j at DC_i is assigned to process the traffic of r_j, then its experienced delay consists of the transfer delay $d(s_j, DC_i)$ from s_j to DC_i, the processing delay $d(SC_j, DC_i)$ by an instance of SC_j at DC_i, and the transfer delay $d(DC_i, t_j)$ from DC_i to t_j. The end-to-end delay requirement of r_j is:

$$d(s_j, DC_i) + d(SC_j, DC_i) + d(DC_i, t_j) \leq D_j. \quad (1)$$

C. Stateful Active and Stand-by Instances

Faults can occur anywhere and at anytime in a network due for example to natural disasters in the locations of DCs, software malfunctions in VNFs, and hardware failures in DCs. To avoid service interruption due to such failures, we assume that an *active instance* of service chain of each request is placed into one DC, and a few *stand-by instances* of the service chain are placed into other DCs. For simplicity, the instances are considered at the service chain level which consists of various VNFs. Once the active instance fails (e.g., one of the composite VNFs within a SC fails), its traffic can be seamlessly redirected to one of the stand-by instances for processing. In this work, we consider *stateful* VNFs (i.e., stateful SCs), whereby the states from the active instance need to be constantly transferred to stand-by instances while the active instance is still in operation. Such state transfer plays a vital role in enabling the seamless and correct request redirection from an active stance to a stand-by instance.

We denote as DC_j^a the DC where the active instance of service chain SC_j of user request r_j is placed and denote as $\mathcal{DC}_j^s$ the set of DCs where stand-by instances of SC_j are placed. We assume that the state update rate of each request from its active instance to stand-by instances is proportional to its packet rate, i.e., $\beta \cdot \rho_j$, where β (> 0) is a given constant. We further assume that the computing resource demand of stand-by instances will be allocated only when they are activated (i.e.,

stand-by instances do not consume computing resources in the placement problem). Since the focus of our work is on the pre-failure placement of active/stand-by VNFs, we consider the resource provisioning of back-up instances (after VNFs fail) out of the scope of this paper.

D. Cost model

Minimizing the implementation cost for user requests is usually considered as an effective objective to reduce the operational cost of network service providers. Here, *the implementation cost* of a request $r_j = (s_j, t_j, SC_j, \rho_j, D_j)$ consists of (i) the operational cost of computing resource to process requests, i.e., the use of an active instance of service chain SC_j in DC_j^a, (ii) the communication cost of transferring its traffic from s_j to DC_j^a for processing, (iii) the communication cost of transferring the processed data from DC_j^a to its destination t_j, and (iv) the communication cost of updating status from DC_j^a to DCs in $\mathcal{DC}_j^s$. Let $c(SC_j, DC_i)$ be the cost of implementing an instance of service chain SC_j of r_j in DC_i, and $c(e)$ be the cost of transferring a unit packet rate for request r_j through link $e \in E$. Without loss of generality, we assume that the edge cost $c(e)$ is within the range of $(0, 1]$. Then, the implementation cost $c(r_j)$ of r_j in active DC_j^a and a set $\mathcal{DC}_j^s$ of stand-by DCs of the network is:

$$c(r_j) = \rho_j \Big(c(SC_j, DC_j^a) + \sum_{e \in p_{(s_j, DC_j^a)}} c(e) + \sum_{e \in p_{(DC_j^a, t_j)}} c(e) + \sum_{DC_i \in \mathcal{DC}_j^s} \sum_{e \in p_{(DC_j^a, DC_i)}} c(e) \Big), \tag{2}$$

where $p_{(y,z)}$ is the shortest path in G from node y to node z.

E. Problem definition

Different cloud service providers may have different network performance indicators to optimize the service delivery process of their networks. To cater for the different optimization objectives of different network service providers, we study two different fault-tolerant VNF placement problems that correspond to different operators' needs as follows.

1) Considering that start-up service providers have limited computing and bandwidth resources, their main interest is to admit as many requests as possible, so that their limited resources are perfectly utilized while achieving the least operational cost. Thus, we consider the optimization objective as the maximization of the admitted number of requests. Specifically, the goal of *the fault-tolerant VNF placement problem* is for all user requests r_j in $\mathcal{R}$ to place an active instance of service chain SC_j to a DC_j^a, to place a number of stand-by instances to a set of DC_j^s, to find the routing path for requests from s_j to t_j via DC_j^a and to find the state update path from DC_j^a to DC_j^s, so that as many requests as possible are admitted while the total cost of implementing these admitted requests is minimized, subject to the computing resource capacity constraints $C(DC_i)$, the network bandwidth capacity $B(e)$ for $e \in E$, and the end-to-end delay constraints.

2) Service providers that provide computing-intensive workload processing in distributed DCs may want their DCs to be balanced (e.g., geographical load balancing), such that users in different locations have maximum resource availabilities with guaranteed user experiences. Assuming that links in G have abundant resources to implement all requests in $\mathcal{R}$, let $\mathcal{R}_i$ be the set of instances of service chains that are admitted by DC_i. The goal of *the fault-tolerant VNF placement problem without bandwidth capacity constraint* is the same as the fault-tolerant VNF placement problem except that the objective is to minimize the maximum DC utilization for all DCs, i.e.,

$$\min \max_{DC_i \in \mathcal{DC}} \sum_{r_j \in \mathcal{R}_i} \frac{\rho_j \cdot c_b}{C(DC_i)}, \tag{3}$$

while the cost of implementing all requests is minimized, i.e.,

$$\min \sum_{DC_i \in \mathcal{DC}} \sum_{r_j \in \mathcal{R}_j} c(r_j), \tag{4}$$

subject to the computing resource capacity constraints of DCs in $\mathcal{DC}$ and the end-to-end delay constraints of requests.

Both problems are clearly NP-hard given that a special version of these problems without considering fault-tolerant requirements and (or) bandwidth resource constraints is NP-hard by simple reduction from another NP-hard problem, the unsplittable single-source flow problem [13].

III. A Heuristic for the Fault-Tolerant VNF Placement Problem

Due to the NP-hardness of the problem, in the following, we propose an efficient heuristic to solve it.

A. Algorithm

To avoid poor performance in terms of request admission rate and cost, the placement of active/stand-by instances, request routings and update paths need to be jointly computed. Conventional approaches such as naive *greedy algorithm* select DCs for the active and stand-by instances separately. It first finds the DC with the largest amount of available computing resources to host the active instance for r_j, and then selects a random number of DCs with lowest transfer costs to host stand-by SC instances for r_j. As a result, the separate placement and routing decision may result in situations where no update paths are available from the active instance to one of its stand-by instances due to link congestions.

In contrast, our heuristic jointly selects a DC for the active instance and a number of DCs for its stand-by instances. Specifically, the heuristic first sorts all requests in $\mathcal{R}$ in increasing order of their rates, and then sequentially considers the requests in the sorted list. Next, for the jth request r_j in the sorted list, the algorithm ranks DCs based on the increasing order of the product of the available computing resources and the accumulative available network bandwidth resources of DC's inbound links. Let $NR(DC_i, j)$ be the ranking of DC_i after considering the $(j-1)$th request in the sorted list. Let also denote $A(DC_i, j)$ and $A(e, j)$ as the available computing and bandwidth resources of DC_i and link e after considering the $(j-1)$th request. Then,

$$NR(DC_i, j) = A(DC_i, j) \cdot \sum_{e \in E_{adj}^i} A(e, j), \tag{5}$$

where E^i_{adj} is the set of inbound links of DC_i. The idea of such ranking is to find a set of DCs that not only have enough computing but also network bandwidth resources for both active and stand-by instances.

Based on the obtained ranking, the algorithm selects the DC with the highest rank, denoted DC_{hr}. Then, the algorithm checks if (1) DC_{hr} has enough computing resources to host an active instance of SC_j for r_j; and (2) if the shortest path from s_j to t_j via DC^a_j has enough bandwidth resources to transfer r_j at rate ρ_j. The algorithm also checks whether (3) DC_{hr} conforms to r_j's delay requirement. If the above three requirements are all satisfied, DC_{hr} is selected as DC^a_j. The algorithm then searches stand-by instances for r_j. To this end, the rest of DCs other than DC_{hr} are sorted in the increasing order of state update costs to DC_{hr}. Each DC in the sorted DC list is further added to $\mathcal{DC}^s_j$ until there is a DC that cannot meet the bandwidth resource requirement for updating states from DC_{hr}. To avoid all the other data centers to be selected to host stand-by instances, we set a threshold K ($1 \leq K \leq |\mathcal{DC}|$) for the number of DCs that can be used for stand-by instances. This prevents a large number of DCs to be selected to place stand-by instances and as such avoids the creation of unnecessary burden for state updates. If no stand-by DC exists after considering the rest of the DCs, request r_j is rejected.

In case constraints (1), (2) and (3) cannot be satisfied, DC_{hr} is added to $\mathcal{DC}^s_j$ as the accumulative bandwidth resources to nearby DCs might make DC_{hr} a promising candidate for stand-by instances. DCs other than DC_{hr} are sorted in a list L_{hr} based on the increasing accumulative communication cost to DC_{hr}. The algorithms then iterates through DCs in L_{hr} until a DC_i that can serve the active service chain instance is found, i.e., a DC that meets constraints (1), (2) and (3). Once such a DC_i is found, it is selected to be the DC that hosts the active instance of r_j. Among the rest DCs in L_{hr}, only the ones that have enough bandwidth resources for state updates rate $\beta \cdot \rho_j$ from DC^a_j are added to $\mathcal{DC}^s_j$ (with $|\mathcal{DC}^s_j| \leq K$). If neither such DC can be found for its active instance nor a set of DCs can be determined for its stand-by instances, r_j is rejected.

The above procedure continues until all requests in $\mathcal{R}$ are considered. The details of the proposed heuristic are shown in Algorithm 1.

Algorithm 1 Heuristic

Input: Network $G(V \cup \mathcal{DC}, E)$; Set of requests $r_j \in R$ where $r_j = (s_j, t_j, SC_j, \rho_j, D_j)$, K.
Output: Assignments of each request in $r_j \in \mathcal{R}$ to DC^a_j for active SC instances and to $\mathcal{DC}^s_j$ for stand-by SC instances.

1: **for** $r_j \in \mathcal{R}_j$ **do**
2: $Sorted_{list} \leftarrow$ SortIncreaseOrder($\mathcal{DC}$) based on Eq. (5)
3: $DC_{hr} \leftarrow Sorted_{list}.getFirst()$;
4: $\mathcal{DC}^s_j \leftarrow \emptyset$ and $DC^a_j \leftarrow NIL$;
5: $A(p_{(s_j,DC_{hr})}) \leftarrow G.shortestPathAvailBandwidth(s_j, DC_{hr})$;
6: $A(p_{(DC_{hr},t_j)}) \leftarrow G.shortestPathAvailBandwidth(DC_{hr}, t_j)$;
7: **if** $\rho_j \leq A(p_{(s_j,DC_{hr})})$ && $\rho_j \leq A(p_{(DC_{hr},t_j)})$ && $D_{hr} \leq D_j$ **then**
8: $DC^a_j \leftarrow DC_{hr}$;
9: $Update_{list} \leftarrow$ SortIncreaseOrder($\mathcal{DC} \setminus DC_{hr}$) based on state update costs to DC_{hr};
10: **for** each $DC_i \in L_{hr}$ **do**
11: $\mathcal{DC}^s_j \leftarrow \mathcal{DC}^s_j \cup \{DC_i\}$
12: **if** $K = |\mathcal{DC}^s_j|$ or $A(p_{(DC_i,DC_{hr})}) \leq \beta \cdot \rho_j$ **then**
13: Break;
14: **else**
15: $\mathcal{DC}^s_j \leftarrow \mathcal{DC}^s_j \cup \{DC_{hr}\}$
16: $L_{hr} \leftarrow$ SortIncreaseOrder($\mathcal{DC} \setminus DC_{hr}$) following state update costs to DC_{hr};
17: **for** each $DC_i \in L_{hr}$ **do**
18: **if** $DC^a_j \neq NIL$ && $A(p_{(DC^a_j,DC_i)}) \geq \beta \cdot \rho_j$ && $|\mathcal{DC}^s_j| \leq K$ **then**
19: $\mathcal{DC}^s_j \leftarrow \mathcal{DC}^s_j \cup \{DC_i\}$;
20: **else**
21: **if** $\rho_j \leq A(p_{(s_j,DC_i)})$ && $\rho_j \leq A(p_{(DC_i,t_j)})$ && $D_i \leq D_j$ **then**
22: $DC^a_j \leftarrow DC_i$;
23: **else**
24: **if** $|\mathcal{DC}^s_j| \leq K$ **then**
25: $\mathcal{DC}^s_j \leftarrow \mathcal{DC}^s_j \cup \{DC_i\}$;
26: Update DCs' available resources and network link resources
27: **return** The assigned DC to place the service chain of each request for the processing of its traffic, and a set of DCs to replicate its service chain.

B. Algorithm Complexity

The performance of the proposed heuristic is given by the following theorem.

Theorem 1. *Given a network $G = (V \cup \mathcal{DC}, E)$, let $\mathcal{R}$ be a set of requests with each represented by $r_j = (s_j, t_j, SC^k, \rho_j, D_j)$. Algorithm 1 delivers a feasible solution to the fault-tolerant VNF placement problem in $O(|\mathcal{R}|(|\mathcal{DC}| \log |\mathcal{DC}|) + (|V| + |\mathcal{DC}|)^3)$ time.*

Proof. To show the feasibility of the algorithm, we need to show that the resource demands of each admitted request and its end-to-end delay requirement are met. Clearly, this is true due to steps 7 and 21.

For the running time of the proposed heuristic, we can see that the most time-consuming phases of Algorithm 1 are (1) finding all pair shortest paths in G, (2) ranking all DCs, and (2) iteratively selecting a number of DCs for each request. Clearly, phase (1) takes $O((|V| + |\mathcal{DC}|)^3)$ time, phase (2) takes $O(|\mathcal{DC}| \log |\mathcal{DC}|)$ time, and phase (3) takes $O(|\mathcal{DC}|)$ time. Since the ranking of DCs is performed every time when a request is admitted, the overall running time of algorithm 1 is $O(|\mathcal{R}|(|\mathcal{DC}| \log |\mathcal{DC}|) + (|V| + |\mathcal{DC}|)^3)$. □

IV. A $(2, 4+\epsilon)$ Bicriteria Approximation Algorithm for the Problem without Bandwidth Constraint

In this section, we consider the fault-tolerant VNF placement problem without bandwidth capacity constraint. We assume that the network G has enough bandwidth resources on its links, and all requests in $\mathcal{R}$ can be admitted. For this problem, we propose a bicriteria approximation algorithm with an approximation ratio of $(2, 4+\epsilon)$. Such a ratio indicates that (1) the implementation cost of all requests is twice the optimal cost, and (2) the minimum maximum utilization of computing resources in a DC is $(4+\epsilon)$ times the optimal one, where ϵ is a constant with $\epsilon > 0$.

A. Overview

Given network G and a set $\mathcal{R}$ of requests, the fault-tolerant VNF placement problem without bandwidth capacity constraint is to balance the workloads among DCs by not only minimizing the maximum resource utilization of DCs but also minimizing the total requests' implementation costs. One challenge is with respect to the tradeoff between the balance of DC

utilizations and the implementation costs of requests. For instance, the active instance of some requests may have to be placed into DCs with high communication costs in order to achieve a balanced workload among DCs. In order to achieve a near optimal solution, we jointly consider the active/stand-by instance placements, request routings and state update paths.

The idea behind the proposed approach is to reduce the fault-tolerant NFV placement problem without the bandwidth capacity constraint in G into a single-source unsplittable flow problem [13] in an auxiliary graph $G' = (V', E')$. Then, a feasible unsplittable flow in G' that minimizes both the implementation cost of requests and the maximum congestion of links in G' is a feasible solution to the original problem in G. Note that the aim of the single-source unsplittable flow problem is, given a network $G = (V, E, u)$, a source vertex s, and a set of M commodities with sinks t_1, ..., t_M and associated real-valued demands σ_1, ..., σ_M, to route the demand σ_m of each commodity m along a single $s - t_m$ flow path so that the congestion, i.e., $max_{e \in E}\{\frac{f_e}{u_e}, 1\}$, and the cost of flow f are minimized, while the edge capacities constraints of G are met. To solve the unsplitable flow problem, Kolliopoulos and Stein [13] gave $(2, 4 + \epsilon)$ approximation algorithm for flow cost and link congestion.

B. Algorithm

The approximation algorithm first constructs the auxiliary graph $G' = (V', E')$. Recall that the traffic of each request r_j is processed by an active instance of its SC_j in a DC, and by one of its stand-by instances in other DCs if the active instance fails. Thus, each DC_i corresponds to a *DC node* (see Fig. 2), and is added into the auxiliary graph G', i.e., $V' \leftarrow \{DC_i \mid 1 \leq i \leq |\mathcal{DC}|\}$. For each DC node DC_i, we further add a *virtual DC node* DC'_i (see Fig. 2) into V', i.e., $V' \leftarrow V' \cup \{DC'_i\}$ so that the DC capacity constraint is converted into a link constraint. Next, for each DC node, we add a few *stand-by set nodes* to G', whereby each stand-by set node represents a candidate set of DCs for stand-by instances (see Fig. 2). Specifically, the stand-by set nodes of DC_i are different combinations of DCs from $\mathcal{DC} \setminus \{DC_i\}$ whereby each stand-by set node has no more than K DCs. For example, DC_1 from Fig. 2 has a 3 stand-by set nodes DC_2, DC_3 and node DC_2,DC_3 where $k = 2$. Note that a stand-by set node will not be added twice (e.g., there is only one DC_1 ins stand-by set nodes). Last, we add a *request node* into V' for each request r_j, and add a common source s_0 for all requests into V'.

An edge from the common source s_0 to each of stand-by set node is added into E'. Its capacity and cost are set to infinity and zero, respectively (i.e., no bandwidth constraint). Also, there is an edge from each stand-by set node to a DC node DC_i if DC_i is not in the set of DCs represented by the stand-by set node (e.g., DC_1 has edges to DC_2, DC_3 and DC_2 & DC_3 in Fig. 2). The capacity of the edge is set to infinity, and its cost is the accumulative cost of state updates from DC_i to the DCs within the set of DCs represented by the stand-by set node. Further, an edge from DC_i to DC'_i is added. Its capacity is the processing capacity of DC_i, and its cost is set to 0. We add an edge from each DC'_i to a request r_j if DC_i provides a total delay (e.g., sum of processing and communication delay) for request r_j smaller than the request delay requirement. The capacity of this edge is set to infinity. Its cost is the total cost of processing costs of DC_i for request r_j (e.g., $\rho_j c(SC_j, DC^a_j)$) plus the communication costs from s_j to DC_i and from DC_i to t_j at packet rate ρ_j. Fig. 2 shows an example of the constructed auxiliary graph G'. Given the constructed auxiliary graph $G'(V', E')$, the original problem is transferred to the problem of single source unsplittable flow problem in G'. To find a feasible flow f in G', the algorithm presented in [13] is invoked. The main steps of the approximation algorithm are shown in Algorithm 2.

Algorithm 2 An $(2, 4+\epsilon)$ Bicriteria Approximation Algorithm for the fault-tolerant VNF Placement Problem without Network Bandwidth Constraint

Input: A network $G(V \cup \mathcal{DC}, E)$, a set requests $r_j \in R$ where $r_j = (s_j, t_j; SC_j, \rho_j, D_j)$.

Output: Assignments of each requests in $r_j \in \mathcal{R}$ to DC^a_j for active SC instances and to $\mathcal{DC}^s_j$ for stand-by SC instances.

1: Construct an auxiliary graph $G' = (V', E')$ from network $G(V \cup \mathcal{DC}, E)$ as exemplified by Fig. 2;
2: Find a single-source unsplittable flow$f$ in the auxiliary graph G' by applying the algorithm presented in [13];
3: The requests that are assigned into DC_i in the flow f will be processed by an instance of a service chain in DC_i, and request will be assigned a set of DCs that are represented by the stand-by set node in f.
4: **return** The assigned DC to place the service chain of each request for the processing of its traffic, a set of DCs to replicate its service chain, the request routings and update paths.

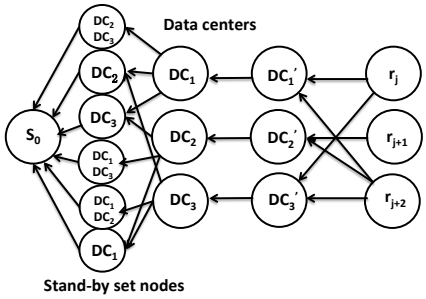


Fig. 2. An example of the auxiliary graph $G' = (V', E')$ constructed from network G with a set $\mathcal{DC} = \{DC_1, DC_2, DC_3\}$ of DCs that are connected by a set $V = \{v_2, v_3, v_5\}$ of switches. $\mathcal{R} = \{r_j, r_{j+1}, r_{j+2}\}$.

C. Algorithm analysis

We now analyze the correctness and performance of the proposed algorithm.

Theorem 2. *Given a network $G = (V \cup \mathcal{DC}, E)$, let $\mathcal{R}$ be a set of requests with each represented by $r_j = (s_j, t_j, SC^k, \rho_j, D_j)$). Algorithm 2 delivers a bicreteria approximate solution with an approximation ratio of $(2, 4 + \epsilon)$ with (1) the implementation cost of all requests twice the optimal cost, and (2) the minimum maximum utilization of computing resource in a DC $(4 + \epsilon)$ times the optimal one, for the fault-tolerant VNF placement problem without bandwidth capacity constraint, in $O(T_2(|\mathcal{R}| + |V| + |\mathcal{DC}|\binom{|\mathcal{DC}|-1}{k}, |\mathcal{R}| \cdot |\mathcal{DC}| + |\mathcal{DC}|\binom{|\mathcal{DC}|-1}{k}))$ time, where $T_2(m, n)$ is the time to solve a fractional minimum-cost flow problem with m edges and n nodes in the flow graph, and ϵ is a constant with $\epsilon > 0$.*

Proof. We first show the feasibility of the proposed algorithm. Given an unsplittable flow f, it starts at a request node r_j and ends at the common source s_0 in G' according to the construction of auxiliary graph G'. Clearly, a DC node DC_i for active instance and a stand-by set node exists in the route. The traffic of request r_j is processed by the placed active instance in DC_i, and it is routed on the paths from r_j's source s_j to DC_i and from DC_i to destination t_j (e.g., represented by edge $\langle DC'_i, r_j \rangle$ in auxiliary graph). Also, since an auxiliary edge between a stand-by node to DC_i denotes the state update path from DC_i to one of the stand-by set nodes, the processing states are then updated to one of the stand-by set nodes following the traversed edge by f. In addition, the delay requirement of r_j is met, as f only exists when there is an edge between r_j and DC'_i (i.e., delay is met).

We then show the approximation ratio of the devised approximation algorithm. It is clear that the solution to the single-source unsplittable flow problem in auxiliary graph G' corresponds to the solution to the VNF placement problem without bandwidth constraint in network G. The approximation ratio obtained for the former problem thus is the approximation ratio for the latter, i.e., $(2, 4+\epsilon)$.

We then show the time complexity of the approximation algorithm, which can be divided into two stages: (1) the construction of the auxiliary graph $G'(V', E')$; and (2) finding an unsplittable flow in the constructed auxiliary graph using the algorithm proposed by Kolliopoulos and Stein [13]. Clearly, the construction of G' takes $(|V'| + |E'|)$ time, where $|V'| = O(|\mathcal{R}| + |V| + |\mathcal{DC}| + \sum_{k=1}^{|\mathcal{DC}|-1} \binom{|\mathcal{DC}|-1}{k}) = O(|\mathcal{R}| + |V| + |\mathcal{DC}|\binom{|\mathcal{DC}|-1}{k})$, and $E' = O(|\mathcal{R}| \cdot |\mathcal{DC}| + |\mathcal{DC}|\binom{|\mathcal{DC}|-1}{k}))$, where $\sum_{k=1}^{|\mathcal{DC}|-1} \binom{|\mathcal{DC}|-1}{k}$ is the maximum number of stand-by set nodes for all DCs. According to [13], finding a unsplittable flow in G' takes $O(T_2(|V'|, |E'|) + |E'| \log(|V'|/\epsilon)) = O(T_2(|\mathcal{R}| + |V| + |\mathcal{DC}|\binom{|\mathcal{DC}|-1}{k}, |\mathcal{R}| \cdot |\mathcal{DC}| + |\mathcal{DC}|\binom{|\mathcal{DC}|-1}{k})))$ time. □

V. Evaluations

A. Experiment Settings

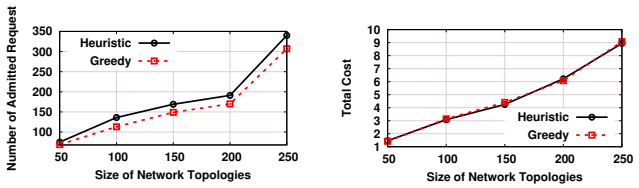
We consider synthetic networks generated by GT-ITM [14]. The network size ranges from 50 to 250 nodes with a node connectivity of 0.2 (i.e., the probability of having an edge between two nodes is 0.2) [15]. In these networks, the server to DC ratio is set to 0.1, and each DC has a CPU capacity in the range 4,000 to 8,000 Mhz. The transmission delay of a network link varies between 2 milliseconds (ms) and 5 ms [16]. The costs of transmitting and processing 1 GB (approximately 16,384 packets with each having size of 64 KB) of data are set within [$0.05, $0.12] and [$0.15, $0.22], respectively, following typical charges in Amazon EC2 with small variations [17]. We consider five categories of NFs: Firewall, Proxy, NAT, DPI, and Load Balancer, their computing demands (e.g., CPU) are adopted from [18]. Further, the consumed computing resources of a service chain is the sum of the computing demands of its contained NFs (the number of contained NFs is randomly selected between 1 and 50). The processing delay of a packet for each NF is randomly drawn from 0.045 ms to 0.3 ms [18], and the processing delay of a service chain is the total processing delay of its NFs. Each request r_j is generated by randomly selecting its source s_j and destination t_j from G with packet rate ρ_j randomly selected between 400 and $4,000$ packets/second [19]. Each request has a delay requirement ranging from 10 ms to 100 ms [20], [21]. The running time is obtained based on a machine with a 3.40GHz Intel i7 Quad-core CPU and 16 GB RAM.

There has not been any existing work considering the fault-tolerant stateful VNF placement. One possible solution is to derive each decision variable in a separate step (similar to an existing approach for stateless VNF placement problem [5]). In this sense, we compare our algorithms against a greedy algorithm (described in III-A) that separately selects the placement of active/stand-by SC instance, request routings and state transfer paths. The greedy aims to maximize the throughput by admitting requests with small packet rates first. For simplicity, we refer to this greedy algorithm as algorithm `Greedy`, and the greedy without bandwidth constraint as $Greedy_{noBW}$. The proposed heuristic and approximation algorithms (Algorithms 1 and 2) are referred to as `Heuristic` and `Appro`, respectively.

B. Performance evaluation

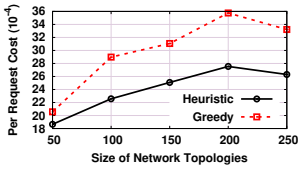
We first compare the performance of algorithm `Heuristic` against that of algorithm `Greedy` for networks with various sizes. Fig. 3 shows the result in terms of the number of admitted requests, the average cost of admitting a request, and the running time. We see from Fig. 3 (a) that the proposed algorithm `Heuristic` consistently achieves a number of admitted requests higher than `Greedy` by 10%. This is due to the fact that algorithm `Heuristic` jointly selects the active and stand-by instances. As such, both network resources and DCs' computing resources are efficiently utilized, which avoids the request rejections that happened with `Greedy` due to separate selection process. The surge of admitted requests observed with both algorithms for networks with size equal to 250 can be explained by the fact that in this case, the bandwidth resources between any two network nodes are on average increased (i.e., more network links exist between two nodes when network size becomes larger), which results in relaxed constraints in terms of bandwidth. From Fig. 3 (b), we observe that the two algorithms achieve almost the same total cost. However, since the overall admitted request number obtained with `Heuristic` is higher than that of `Greedy`, we see from Fig. 3 (c) that the `Heuristic` achieves a lower per request cost than `Greedy`. Furthermore, `Heuristic` achieves a lower cost in terms of the average cost per admitted request than that of `Greedy`. Meanwhile, we see from Fig. 3 (d) that `Heuristic` slightly results in a longer running time than that of algorithm `Greedy`.

We then compare the performance of algorithm `Appro` with that of algorithm $Greedy_{noBW}$ in terms of the maximum resource utilization of DCs, the average cost of implementing a request, and the running time under the same network settings. It can be seen from Fig. 4 (a) that the proposed algorithm `Appro` consistently delivers solutions with lower maximum DC resource utilization than that obtained with algorithm $Greedy_{noBW}$. For example, when the network size is 100, the minimum maximum resource utilization of DCs of

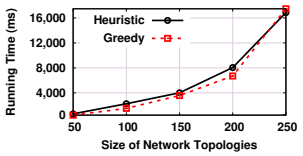


(a) Admitted number of requests

(b) Total cost



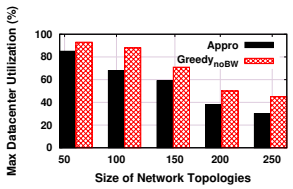
(c) Average cost per request



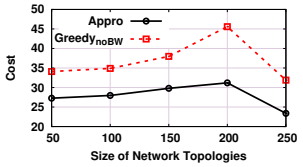
(d) Running time

Fig. 3. Performance of algorithms `Heuristic` and `Greedy`.

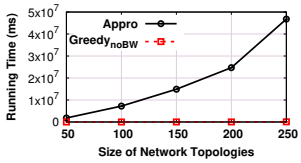
`Appro` is 10% lower than that of algorithm $Greedy_{noBW}$. The rationale behind is that algorithm `Appro` explores a fine-grained trade-off between the resource utilizations and the cost of implementing requests. Fig. 4 (a) also shows that the maximum resource utilization of DCs is decreasing with the network size. This is because larger networks mean on average more computing resources in DCs, which incurs lower resource utilization. In addition, as shown in Fig. 4 (b) and (c), algorithm `Appro` also delivers a lower implementation cost. Regarding the running time, it should be noted that our algorithms are intended to be executed offline and to compute solutions that will be implemented in DC networks at the network configuration stage. The running time of `Appro` is therefore considered as tolerable. Finally, we observe that both the maximum DC utilization in Fig. 4 (a) and the total cost in Fig. 4 (b) are not increasing as the network size grows, which further justifies the performance guarantee of the proposed `Appro` algorithm.



(a) Maximum DC utilization



(b) Total cost



(c) Running time

Fig. 4. Performance of algorithms `Approximation` and $Greedy_{noBW}$.

VI. Conclusion

In this paper, we proposed a novel efficient heuristic approach for the fault-tolerant VNF placement problem that jointly computes the placement active and stand-by instances of stateful VNFs, the routing paths and update paths of user requests. For a special case of the problem without network bandwidth constraint, we proposed a bicriteria approximation algorithm with performance guarantees. We evaluated the performance of the proposed algorithms based on simulations under realistic settings. Our evaluation results show that the performance obtained with each algorithm outperforms existing solutions that separately determine placements, routings and state update paths.

Acknowledgment

This work was partially funded by the CHIST-ERA CONCERT/EPSRC, project number I1402 and the EU H2020 UMOBILE, project number 645124.

References

[1] ETSI, *NFV white paper 1*, https://portal.etsi.org/NFV/.
[2] Z. Xu *et al.*, "Throughput maximization and resource optimization in nfv-enabled networks," in *IEEE ICC*, 2017.
[3] S. Clayman *et al.*, "The dynamic placement of virtual network functions," in *IEEE NOMS*, 2014.
[4] R. Potharaju and N. Jain, "Demystifying the dark side of the middle: a field study of middlebox failures in datacenters," in *ACM IMC*, 2013.
[5] F. Carpio, S. Dhahri, and A. Jukan, "Vnf placement with replication for loac balancing in nfv networks," in *IEEE ICC*, 2017.
[6] B. Kothandaraman, M. Du, and P. Sköldström, "Centrally controlled distributed vnf state management," in *ACM SIGCOMM Workshop on Hot Topics in Middleboxes and Network Function Virtualization*, 2015.
[7] J. Khalid *et al.*, "Paving the way for nfv: Simplifying middlebox modifications using statealyzr." in *UNSENIX Proc. of NSDI*, 2016.
[8] S. Rajagopalan, D. Williams, and H. Jamjoom, "Pico replication: A high availability framework for middleboxes," in *ACM Proc. of SoCC*, 2013.
[9] M. Huang *et al.*, "Throughput maximization in software-defined networks with consolidated middleboxes," in *IEEE LCN*, 2016.
[10] Y. Kanizo *et al.*, "Optimizing virtual backup allocation for middleboxes," *IEEE ICNP*, 2016.
[11] H. D. Chantre and N. L. da Fonseca, "Redundant placement of virtualized network functions for lte evolved multimedia broadcast multicast services," in *IEEE ICC*, 2017.
[12] H. Huang *et al.*, "Service chaining for hybrid network function," *IEEE Trans. on Cloud Computing, DOI:10.1109/TCC.2017.2721401*, 2017.
[13] S. G. Kolliopoulos and C. Stein, "Approximation algorithms for single-source unsplittable flow," *SIAM J. on Computing*, vol. 31, no. 3, pp. 919–946, 2001.
[14] GT-ITM, http://www.cc.gatech.edu/projects/gtitm/.
[15] B. Yang *et al.*, "Cost-efficient nfv-enabled mobile edge-cloud for low latency mobile applications," *IEEE Trans. on Network and Service Management, DOI:10.1109/TNSM.2018.2790081*, 2018.
[16] S. Knight *et al.*, "The internet topology zoo," *IEEE J. on Selected Areas in Communications*, vol. 29, no. 9, pp. 1765–1775, 2011.
[17] I. Amazon Web Services, *Amazon ec2 instance configuration*, https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ebs-ec2-config.html.
[18] J. Martins *et al.*, "Clickos and the art of network function virtualization," in *USENIX Proc. of NSDI*, 2014.
[19] Y. Li, L. T. X. Phan, and B. T. Loo, "Network functions virtualization with soft real-time guarantees," in *IEEE INFOCOM*, 2016.
[20] Microsoft, *Plan network requirements for Skype for business*, https://technet.microsoft.com/en-us/library/gg425841.aspx.
[21] B. Yang *et al.*, "Seamless support of low latency mobile applications with nfv-enabled mobile edge-cloud," in *IEEE Cloudnet*, 2016.