A Graph Approach to Placement of Service Functions Chains

Nicolas Tastevin*, Mathis Obadia*†, Mathieu Bouet*
† TELECOM ParisTech, Paris, France
*Thales Communications & Security, Gennevilliers, France
{name.surname}@thalesgroup.com

***Abstract*—Network Functions Virtualization (NFV) is a new network architecture concept which leverages virtualization technologies to make the management of network functions like firewalls, load balancers, WAN optimizers more flexible and cost effective. In this approach, traditional middlebox appliances are replaced by virtual machines embedding Virtual Network Functions (VNFs). One of the main challenges of NFV orchestration is to appropriately deploy and instantiate sequences of VNFs that form Service Functions Chains (SFC). In this paper, we propose a general cost-driven Integer Linear Programming (ILP) formulation of this problem. We then propose a graph-based heuristic that combines graph centrality and multi-stage graphs. We evaluate our heuristic by comparing to optimal solutions provided by the ILP, showing that it is 1000 times faster and very close in terms of costs (less than 1.15% of difference). Then, we evaluate our heuristic on larger instances and compare it to one of the best suited state of the art approach. The evaluation results show that our heuristic scales well and outperforms the related approach, especially when VNF deployment costs are higher than link bandwidth usage costs.**

I. Introduction

Network Functions Virtualization (NFV) has emerged as a new concept to design, deploy and manage networks [1]. Launched by the biggest internet service providers in 2012, it now gathers more than 250 companies in the pre-standardization group of ETSI and has been relayed at the IETF and IRTF in several working groups such as SFC (Service Function Chaining). NFV aims at using IT technologies such as virtualization to softwarize an extensive set of network functions (e.g. firewall, load balancer, intrusion detection systems, ciphering etc.) that may be connected or chained together to create network services. A Virtual Network Function (VNF) consists in one or several virtual machines running onto industry standard high volume servers, switches and storage, by opposition to custom middlebox hardware for each network function, which could be located in datacenters and Points of Presence (PoPs), network nodes and in the end user premises. The NFV concept is envisioned as a way to reduce Internet Service Providers' costs and bring flexibility. NFV orchestration consists in deploying, instantiating and managing a composition of VNFs that form a Service Function Chain (SFC). However, it is challenging to allocate multiple SFC requests on NFV Infrastructure, especially in a cost-driven objective. VNFs have to be chained in a specific order. Moreover, depending on their type and isolation considerations, VNFs can be potentially shared among several SFCs. Finally, VNFs must not be placed far from the shortest path to avoid increasing SFC delay and network usage.

In this paper, we address the problem of allocating Service Functions Chains on an NFV Infrastructure composed of PoPs with VNF hosting capabilities. We formulate this general problem as an Integer Linear Program (ILP). Then, we propose a graph-based heuristic that leverages graph centrality and a multi-stage graph derived from Viterbi algorithm to select the minimum set of PoPs and then allocate the SFC requests on them. Our general approach aims at being easily extensible to address variants of the SFC allocation problem. We implemented the ILP and the heuristic. We evaluate our heuristic by comparing it to i) the optimal solutions provided by the ILP, ii) a randomized approach, and iii) one of the best state of the art heuristic for this problem [2]. We show that despite its generality our heuristic is very close to optimal solutions and outperforms the state of the art solution when PoP and VNF costs are larger than interconnection costs.

The rest of the paper is organized as follows. First, Section II presents related work. Then, Section III describes our Service Functions Chaining Problem and proposes an Integer Linear Programming (ILP) formulation of the model. Section IV presents the heuristic we propose. In Section V, we provide an evaluation of the heuristic by means of comparison with the ILP and an heuristic from Bari et al. [2]. Finally, Section VI concludes this paper.

II. Related Work

Network virtualization was initially driven by the convergence of computation, storage and network in cloud computing. A lot of works in the literature address the placement of virtual machines in datacenters. Several techniques to optimize their placement with respect to server load balancing or energy consumption have been proposed [3], [4]. The placement problem has been previously studied as an offline facility location problem by Laoutaris et al. [5] and has been extended to include reconfigurations costs [6] for instance. Prior to NFV technologies, several works have addressed the Virtual Network Embedding (VNE) problem where multiple virtual networks have to be mapped onto a physical substrate graph [7]. However, the problems of optimizing the placement of VMs and of embedding virtual networks differ from the

978-3-901882-89-0 @2017 IFIP

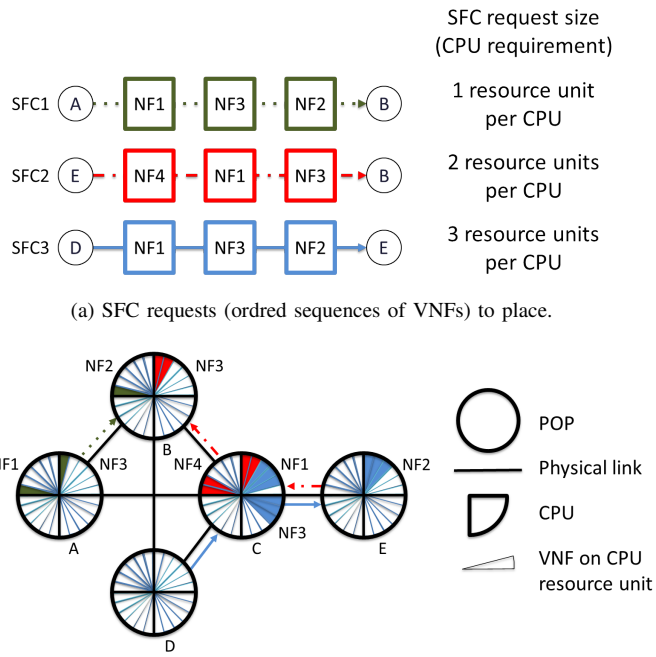
problem of optimizing the placement and the chaining of VNFs. Indeed, the first problem is node-centric, VMs being many and small endpoints, while the second problem is edge-centric, nodes being points to interconnect. The Service Functions Chaining problem is node and edge centric since demands have to pass through a sequence of nodes (VNFs) which have to be placed on PoPs.

Regarding the VNF placement and routing problem or service chaining, Luizelli et al. [8] have recently proposed an heuristic for efficiently guiding an ILP solver towards near-optimal solutions. They perform a binary search to find the lowest number of possible network functions instances that meets the current demand. Then, they run the ILP on this more constrained problem which allows reducing the ILP computational time. Addis et al. [9] proposed an ILP model that considers the traffic compression induced by multimedia gateways or firewalls for the cost-efficient placement of VNF chains. They apply linearization techniques to increase the execution speed of an ILP solver (i.e. CPLEX). Moens et al. [10] proposed an ILP formulation that includes physical and virtual nodes. Bari et al. [2] proposed a heuristic based on the Viterbi algorithm that takes into account path length, potential cost of opening a new PoP and penalty cost due to a SLO violation. Rankothge et al. [11] proposed a genetic algorithm for service function chaining, aiming at being faster than ILP approaches. Bouet et al. [12] tackled the virtual Deep Packet Inspection (vDPI) placement problem. They propose a centrality based greedy algorithm to find an interesting trade-off between the number of vDPIs to deploy and the number of requests redirection. Their problem can be seen as a simplification of a SFC chaining problem. Indeed, it can appear as a SFC problem with requests of 1 VNF (vDPI) only in the chain. Finally, the SFC placement problem has also been studied in specific contexts: Gateway placement in mobile networks [13] or Network Function placement in optical networks [14]. But these works have been implemented with specific metrics related to their context.

Our heuristic, as in [8], makes the assumption that minimizing the number of deployed VNFs gives near-optimal solutions. It is also inspired by Viterbi algorithm. However, it differs from [2] in several ways. It only uses the number of hops in the shortest path as transition cost. PoPs where VNFs can be instantiated are selected before running the algorithm to avoid the dependency between the chosen PoPs where VNFs are deployed and the order on which SFC requests are allocated. Our heuristic aims at being general so that it can be easily extented to integrate variant assumptions or constraints.

III. Service Function Chains Placement Problem Definition and ILP Formulation

In this section, we describe our Service Function Chains allocation problem and outline the assumptions we make. Then, we provide an ILP formulation of the problem.



(a) SFC requests (ordred sequences of VNFs) to place.

(b) Physical topology and resources allocation to SFC requests.

Fig. 1. Example of a deployment of VNF instances to serve 3 SFC requests.

A. Overview of the SFC Placement Problem

Service Function Chaining consists in constructing a graph of virtual network functions that composes a requested network service and choosing the NFV Points of Presence (PoPs) that will hosted the VNF instances. A flow in an SFC has to pass through a determined sequence of network functions, which can be ordered in certain cases. Service requests can have some requirements such as a end-to-end delay to respect, affinity or anti-affinity constraints etc. The objective of our SFC problem is two-sided. First, we aim at finding a placement of VNF instances and a chaining which can serve all traffic demand and respect requests' requirements. Second, we want to find a placement and a chaining which minimizes the cost induced by the deployment of the VNF instances on PoPs (e.g. du to license fees, energy consumption etc.) and by the usage of links' bandwidth. SFC problems can have other objectives such as penalty costs and constraints such as flow compression/decompression rate per VNF, isolation etc. We purposely keep our approach simple to have a general problem that can be easily extended or customized. The hypotheses we do are explained in the next sub-section.

B. Problem Description and Hypotheses

The goal of our SFC problem is to serve all traffic demands and to minimize OPEX (OPeration EXpenditure). We consider that OPEX is composed by two components: i) the VNF deployment cost which is directly linked to the number of PoPs hosting VNF instances and ii) the cost of forwarding traffic which is linked to the number of hops in a traffic request path

and its bandwidth usage. The more the SFC path is long, the more it will cost.

In our problem SFC requests do not have end-to-end delay requirements. The only constraint concerning SFC requests is that they are ordered as a logical sequence of VNFs. We suppose that link resources are infinite. This means that an infinite amount of requests can pass through a link. This hypothesis is viable when link capacity is largely superior to the traffic demand. Nevertheless, we set a cost for link bandwidth usage which will encourage reducing SFC path elongation with respect to the shortest paths and thus will lower the impact on the end-to-end delay.

We mainly constrain our problem on PoP resources. Each PoP has a limited CPU capacity. Thus, a PoP cannot host an infinite number of VNFs. Multiple PoPs should be needed to serve all traffic demands. For example, on Figure 1 each PoP has 4 CPUs, each CPU being able to process 6 resource units per VNF instance. When a CPU is allocated to a VNF instance, the rest of its CPU resource units can serve only requests for the same type of the VNF instance as it is done in [8]. On Figure 1, PoP C hosts 3 different VNF instances of 3 different types of VNF: NF1, NF2 and NF3. These instances serve SFC2 and SFC3 whose traffic demand is 2 and 3 CPU resource units respectively. Finally, if a PoP hosts at least one VNF instance, then it generates what we name an opening cost. It reflects operational costs such VNF license fees, energy consumption etc.

C. ILP Formulation

We mathematically formalize our problem through the following Interger Linear Program. The notations for the variables and the inputs are gathered in Table I.

The physical topology of our NFV infrastructure is composed by PoPs and links. Interconnections between PoPs are represented by a directed graph $G = (N, E)$ where N is the set of nodes and E the set of edges.

We use the same notations to represent logical SFC requests. The set of different VNF types is depicted by V. If m is the number of different VNF types, $V = \{1, 2, ..., m-1, m\}$. In the example illustrated on Figure 1, $m = 4$. A logical SFC request is represented by an Ingress node, an Egress node and a logical sequence of VNFs. We call u_r the number of VNFs requested by a request $r \in R$, where R is the set of SFC requests to serve. $V_r = (v_r^1, v_r^2, ..., v_r^{u_r})$ is the set of requested VNFs, where $v_r^i \in V$ is the type of the i^{th} VNF requested by r. The input I_r gives the ingress node of the request r. Similarly, E_r gives the egress node of the request r.

Each SFC request has a size named s_r which represents two things. First, the number of resource units each VNF of the chain requires at a PoP. Second, the bandwidth, in terms of resource units, the request requires when passing through a physical link. We assume that the cost for forwarding one resource unit through a link is equal to L_c dollars.

As explained before, in our model PoPs have a limited amount of resources. Each PoP has a limited number of CPUs able to host VNF instances. Each CPU can host only one type of VNF instance. The PoP capacity in terms of number of CPUs is called C_c. For example, on Figure 1 PoP B has 4 CPUs and only 2 CPUs host a VNF instance. Upper-left CPU hosts VNF instance of type NF2 while upper-right CPU hosts VNF instance of type NF3. Moreover, each CPU has a limited amount of resource units call V_c. On Figure 1, CPUs have 6 resource units. As we assume that all nodes have the same architecture, inputs C_c and V_c are constant and independent from the considered PoP. If a PoP hosts at least one VNF, it produces a VNF deployment cost of N_c dollars.

TABLE I
VARIABLES & INPUTS NOTATION.

Variables	
$o_n \in \{0,1\}$	PoP n hosts at least one VNF
$y_{r,i,n_1,n_2} \in \mathbb{R}$	Amount of resource units through physical link (n_1, n_2) corresponding to traffic between v_r^{i-1} and v_r^i
$x_{r,i,n} \in \{0,1\}$	PoP n hosts the i^{th} VNF of request r
$x_{r,i,n,c} \in \{0,1\}$	PoP n hosts thethe i^{th} VNF of request r on its c^{th} CPU
$z_{v,n,c} \in \{0,1\}$	PoP n hosts on its c^{th} CPU a VNF of type v
Inputs	
N	Set of nodes
E	Set of edges
G	Directed graph, $G = (N, E)$
R	Set of SFC requests to serve
$m \in \mathbb{N}$	Number of different VNF types
V	Set of different VNF types, $\{1, 2, ..., m-1, m\}$
$u_r \in \mathbb{N}$	Number of VNFs requested by r
V_r	Sequence of VNFs for the request r
$v_r^i \in V_g$	type of the i^{th} VNF of the request r
$I_r \in N$	Ingress node of the request r
$E_r \in N$	Egress node of the request r
$s_r \in \{1, 2, 3\}$	Size of the request r
N_c	Cost for using one PoP
L_c	Cost for forwarding one resource unit through a link
C_c	Number of CPUs per PoP
V_c	Number of resource units per CPU

We then formulate our SFC allocation problem as follows:

Objective:

$$\min\Big(\sum_{n\in N} o_n \cdot N_c + \sum_{r\in R, l\in L, (n_1,n_2)/n_1,n_2\in ens E\in N} y_{r,l,n_1,n_2} \cdot L_c\Big)$$

Subject to:

$$x^c_{r,i,n,c} \leq x_{r,i,n} \quad \forall n \in N, i \in [1 : u_r], c \in C, r \in R \tag{1}$$

$$\sum_{r\in R, i\in[1:u_r]} x^c_{r,i,n,c} \cdot s_r \leq V_c \quad \forall n \in N, c \in C \tag{2}$$

$$\sum_{v\in V, c\in C} z_{v,n,c} \leq C_c \qquad \forall n \in N \tag{3}$$

$$\sum_{v\in V} z_{v,n,c} \leq 1 \qquad \forall n \in N, c \in C \tag{4}$$

$$x^c_{r,i,n,c} \leq z_{v^i_r,n,c}$$
$$\forall r \in R, n \in N, c \in C, i \leq u_r \quad (5)$$

$$z_{v,n,c} \leq o_n \quad \forall v \in V, n \in N, c \in C \quad (6)$$

$$\sum_{n \in N, c \in C} x^c_{r,i,n,c} = 1 \forall r \in R, i \leq u_r \quad (7)$$

$$\sum_{n_2 \in N} y_{r,i,n_1,n_2} - \sum_{n_2 \in N} y_{r,l,n_2,n_1} =$$
$$\begin{cases} (x_{r,i-1,n_1} - x_{r,i,n_1}) \cdot s_r, \forall r \in R, (n_1, n_2) \in E, 2 \leq i \leq u_r \\ (1 - x_{r,i,n_1}) \cdot s_r, \text{ for } n_1 = I_r, i = 1, \forall r \in R \\ x_{r,i,n_1} \cdot s_r, \text{ for } i = 1, \forall r \in R, (n_1, n_2) \in E, n_1 \neq I_r \\ (x_{r,i-1,n_1} - 1) \cdot s_r, \text{ for } n_1 = E_r, i = u_r + 1, \forall r \in R \\ x_{r,i-1,n_1} \cdot s_r, \text{ for } i = u_r + 1, \forall r \in R, (n_1, n_2) \in E, n_1 \neq E_r \end{cases} \quad (8)$$

$$y_{r,l,n_1,n_2} \geq 0$$
$$\forall r \in R, (n_1, n_2)/(n_1, n_2) \in E, i \leq u_r + 1 \quad (9)$$

The objective function aims at minimizing both the total cost composed of the VNF deployment cost and the forwarding traffic cost. Constraint 1 ensures that a VNF that is on a PoP CPU is also on this PoP. Constraint 2 ensures that there are no more resource units on a CPU than the resource unit capacity per CPU. Constraint 3 ensures that the number of CPUs instantiated on a node does not exceed the PoP capacity in terms of number of CPUs. Constraint 4 ensures that a CPU hosts at most one VNF type. Constraint 5 ensures that if a VNF of a SFC request is hosted by a CPU of a PoP then the PoP hosts on this CPU a VNF instance of the same type. Constraint 6 ensures that if at least one CPU of a PoP hosts a VNF instance, this PoP is considered as opened or "deployed". Constraint 7 ensures that a VNF of a SFC request is only allocated one time. Constraints 8 ensure the network flows conservation. Finally, constraint 9 ensures that the amount of resource units allocated on a physical link is non-negative.

IV. Our Graph-based Heuristic

In this section, we present the heuristic we propose. It can be decomposed into 3 steps. The first step consists in finding the minimum number N_{min} of PoPs that have to be used to serve all SFC requests. Step 2 concerns the election of the N_{min} PoPs to use. This set of PoPs can be different than the one found through the first step. Finally, step 3 deals with the chaining of SFC requests by assigning VNFs on the elected PoPs and building SFC requests path. Thus step 1 minimizes the VNF deployment cost, while steps 2 and 3 deals with the minimization of the traffic forwarding cost. Our heuristic is formalized in Algorithm 1. In the rest of this section, we detail its three main steps.

Algorithm 1 Our heuristic algorithm.

```
allocationVariables ← initialisation(N, L)
orderedRequests ← orderRequestsBySize(R)
P ← ∅                                          ▷ Beginning of Step 1
P.addEmptyPoP()
for r ∈ orderedRequests do
    if P.isAllocationPossible(r) then
        P.allocate(r)
    else
        P.addEmptyPoP()
        P.allocate(r)
    end if
end for
N_min ← Card(P)
for n ∈ N do                                   ▷ Beginning of Step 2
    n.computeCentrality(G)
end for
orderedNodes ← orderNodesByCentrality(N)
PoPsToDeploy ← bestCentrality(orderedNodes, N_min)
for r ∈ orderedRequests do                     ▷ Beginning of Step 3
    msGraph ← stagesGraph(G, r,PoPsToDeploy,
allocationVariables)
    viterbiResults ← viterbi(msGraph)
    allocationVariables.update(viterbiResults)
end for
return allocationVariables
```

A. Calculation of the minimum number of PoPs to use

One of our assumption is that PoPs and VNFs have a larger impact on OPEX that PoP interconnections. By assuming that $N_c \gg L_c$, minimizing the VNF deployment cost becomes highly necessary to minimize the total cost. Our strategy thus consists in starting by minimizing the VNF deployment cost, that is directly linked to the number of PoPs used to instantiate them. For example, on Figure 1, 4 PoPs are deployed but only 1 PoP is enough to serve all requests. Indeed, each of the 4 CPUs can host a VNF type different from the other one and each VNF instance can serve the total demand for it. The placement proposed on Figure 1 is thus not optimal. Obviously, once the VNF deployment cost is minimized, the traffic forwarding cost cannot anymore achieve its lowest value corresponding to SFC requests following their shortest path. However, from the minimum set of PoPs to use we can find a chaining which keeps the traffic forwarding cost near to its lowest value.

To find this minimum number of PoPs to use, we rank requests from decreasing order in terms of resource units required. Then we initialize a set of PoPs called P with one empty PoP. For each request, we try to allocate all VNF instances on the PoPs $\in P$. If it is not possible, we add an empty PoP to P and allocate the VNFs on the updated set P. Once all the requests are allocated, we obtain the minimum number of PoPs to deploy by looking at the cardinality of the set P. We call N_{min} this minimum number. On the example

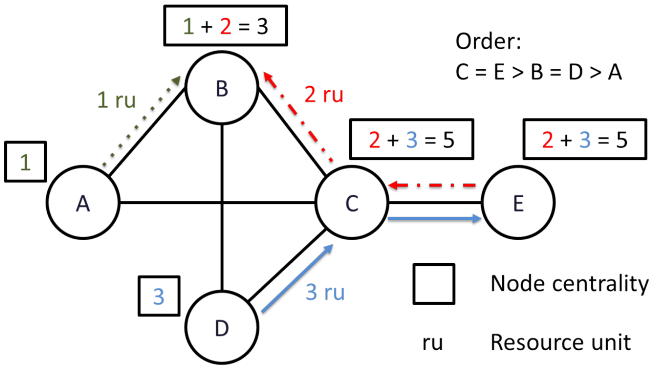


Fig. 2. Example of the modified betweenness centrality.

on Figure 1, N_{min} would be equal to 1.

The complexity of this part of the heuristic is $\Theta(r)$ where r is the number of requests.

B. Selection of the set of PoPs to use

The second part of the heuristic consists in electing the N_{min} PoPs to deploy. The idea is to choose PoPs in order to minimize request redirections and thus path elongation. To do it we borrow a centrality graph concept. We employ the centrality proposed by [12], derived from the well-known betweenness centrality. The betweenness centrality of a node is equal to the number of shortest paths from all vertices to all others that pass through this node. In this paper as in [12], we propose to focus only on the shortest paths from the ingress node to the egress node of each SFC request. Moreover, a request requiring x resource units is counted x times. Thus the centrality of a node becomes the sum of the requests' shortest path that pass through this node, where a request requiring x resource units is counted x times. We illustrate this modified betweenness centrality on Figure 2 with a simple example. We see that PoP E has the highest centrality. However with a classic betweenness centrality PoP E would have the lowest centrality.

Then, we rank the PoPs by decreasing order in terms of centrality value and elects the N_{min} first nodes. This modified centrality is a key step in our algorithm. It allows having a placement which takes into account the global behavior of all requests on the NFV infrastructure.

This step needs to perform a shortest path algorithm as Dijkstra's algorithm on all requests. If we denote by r the number of SFC requests and n the number of PoPs in the NFV infrastructure, then for an Erdos-Renyi graph of degree d the complexity of this step is $\Theta(r \cdot \frac{d+2}{2} \cdot n \cdot log(n))$.

C. Placement and allocation of SFC requests on the elected PoPs

The last part of the heuristic consists in assigning the VNFs of the SFC requests on the elected N_{min} PoPs and ensuring that every request passes through the proper sequence of VNFs. The goal of this step is to minimize the cost of forwarding traffic between the PoPs.

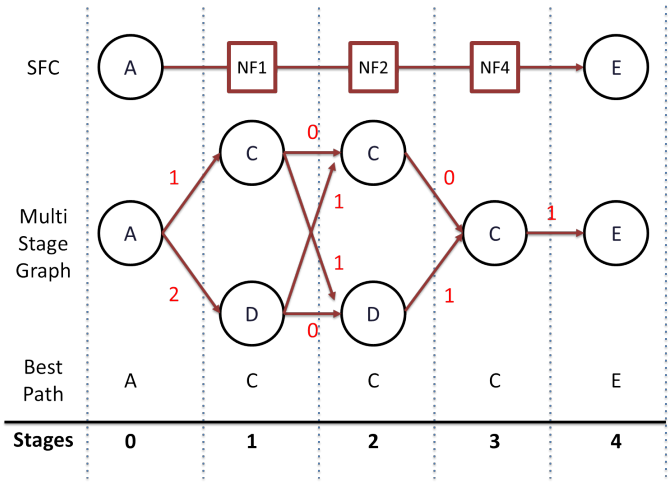


Fig. 3. Example of a multi-stage graph derived from Viterbi algorithm for a SFC request. NF1 and NF2 are available on nodes C and D, while NF4 is only available on node D.

The Viterbi algorithm is a famous method for finding the most probable sequence of states from a set of observed states. It starts by building a multi-stage graph where the nodes represent possible states. Each stage of the graph lists the possible states at this sequence stage. Every node of a stage is linked to all nodes of the next stage by a weighted edge. The edge between a node a at stage $x-1$ and a node b at stage x is weighted by the probability to arrive in node b knowing we are in node a .

In our problem we have to allocate the VNFs of SFC requests on the elected PoPs. We also have to minimize the path length of the requests. To do this, we borrow the idea of how Viterbi algorithm works. For each SFC request we build a multi-stage graph. If we call u_r the number of VNFs requested by the request r, the graph counts $u_r + 2$ stages. Stage 0 is only composed by the ingress node of the request. Similarly stage u_r+1 is only composed by the egress node of the request. At stage $1 \leq x \leq u_r$, nodes represent the possible PoPs to host the x^{th} VNF of the request. To be considered as a candidate PoP, a PoP needs to belong to the set of elected PoPs and to have enough resource units available to host the VNF at this stage. A node at stage $0 \leq x \leq u_r$ is linked to all nodes of stage $x+1$ by a weighted edge. The weight corresponds to the shortest path between the two nodes. By performing the Viterbi algorithm on this multi-stage graph, we obtain the VNF placement and chaining which minimize the request path length and ensure to follow the logical sequence of VNFs. Obviously, once the allocation of a request is done, PoPs resources have to be updated in consequence. When implementing this part of Algorithm 1, we have to be very careful as the choice of a PoP at stage x influences the set of candidate PoPs at stage $x+1$. For example, if we allocate the last resources of PoP k at stage x, this PoP will not be able to host another VNF. In other terms, our multi-stage graph is dynamic contrary to usual Viterbi multi-stage graph.

Figure 3 illustrates a multi-stage graph for a request com-

posed of 3 VNFs. In particular, it shows that PoPs C and D can host NF1 and NF2 whereas PoP C can only host NF4. However, this graph can change, as explained before; if PoP C has not enough resources to host both NF1 and NF2 for example. The weight on an edge corresponds to the length of the shortest path between the two vertices of the edge. By applying the Viterbi algorithm on this multi-stage graph, we obtain the best path in terms of path length and thus aim at keeping traffic forwarding cost at a low value. At the end of this third step, all the SFC requests are allocated on the N_{min} elected PoPs.

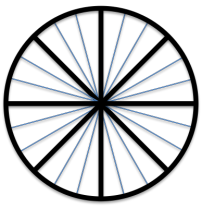
To build such a multi-stage graph, we need to compute all shortest paths between 2 stages of the graph. We number stages of a multi stage-graph from 0 to $u_r + 1$. For stages $\in [1; u_r]$, corresponding to VNFs, there are on average $\frac{r}{4}$ PoPs. Thus between these two stages we have to compute $\frac{r^2}{16}$ shortest paths. There are $u_r - 1$ such transitions to perform. Consequently, the number of shortest paths to compute to perform all the transitions between the VNFs stages is $\frac{r^2}{16}(u_r - 1)$. For a transition between the ingress node and the first VNF of a request, or between the last VNF and the egress node, there are only $\frac{r}{4}$ shortest paths to compute. We thus neglect these steps in the total complexity. The complexity to find the best path for a request becomes $\Theta(\frac{r^2}{16}(u_r - 1) \cdot \frac{d+2}{2} \cdot n \cdot log(n))$. As there are r requests to allocate, the total complexity for this third part of the algorithm is $\Theta(\frac{r^3}{16}(u_r - 1) \cdot \frac{d+2}{2} \cdot n \cdot log(n))$. This is also the total complexity of Algorithm 1 as the complexity of this third step is dominant front of the complexity of the two previous steps.

V. Evaluation

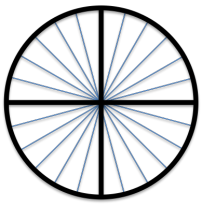
In this section, we evaluate our heuristic. First, we compare it to the ILP which gives optimal solutions for small instances and we formulate a theoretical approximation of the optimal solutions for large instances. Then, we outline the importance of the centrality-based step in our heuristic approach. Finally, with large instances, we compare our heuristic to one of the best state of the art approach proposed by Bari et al. [2] that we call $Viterbi$.

A. Setup

We implemented our ILP using Cplex Optimization Studio v12.6 [15] and our heuristic with Python 3. To simulate physical NFV infrastructures, we build at random connected Erdos-Renyi graphs of degree 3 [16]. We average our experiments by generating 30 random graphs for each problem instance, that is for each set of parameters. We set VNF deployment costs larger than link costs as explained in Section IV. All our SFC requests count 3 VNFs randomly chosen among a set of 4 VNF types. There are 3 possible SFC request sizes: 1 resource unit, 2 resource units, and 3 resource units. The type of each VNF and the size of the requests are chosen at random with a uniform law. For a problem instance, all PoPs are equal (same number of CPUs and same number of resource units per CPU) as in [8]. We perform simulation on two types of PoP architecture as presented on Figure 4.



(a) PoP of type A: 8 CPUs with 3 processing resource units per CPU.



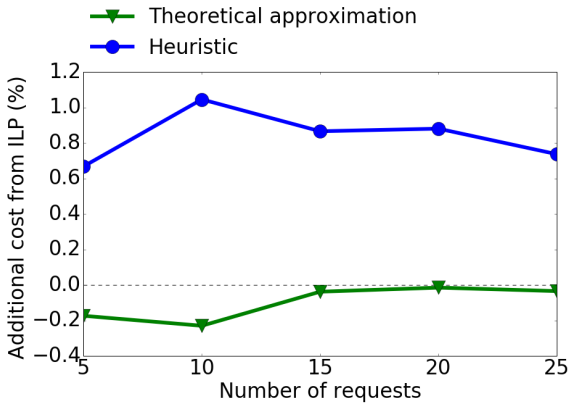
(b) PoP of type B: 4 CPUs with 6 processing resource units per CPU.

Fig. 4. The two PoP architectures, each having a total of 24 resource units.

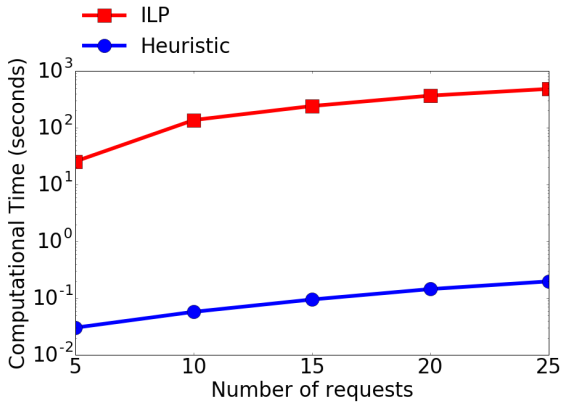
B. Results and analysis

1) Comparison with optimal solutions from the ILP: First, we focus on small instances and compare our heuristic to our ILP. Moreover, we formulate a theoretical approximation to anticipate the fact that the ILP will not scale on large instances. Our theoretical approximation is a lower bound built as follows. The number of PoPs to deploy is equal to N_{min} and requests' path is assumed to be the shortest path. In addition to the comparison of our heuristic with the optimal solution, we test the relevance of our theoretical approximation. We set the number of PoPs to 10 and we fluctuate the number of requests from 5 to 25 by step of 5. All PoPs have the same architecture (type A). Figure 5a shows the additional cost of the heuristic and the theoretical approximation with respect to the optimal solutions provided by the ILP. For example, if the heuristic produces a total cost of $\$11,000$ and the optimal solution indicates a total cost of $\$10,000$, then the additional cost of the heuristic is $\frac{11{,}000-10{,}000}{10{,}000} \times 100 = 10\%$. Obviously, the additional costs of the heuristic are positive values as the ILP gives the optimal solution. For the theoretical approximation, they are negative as we build the approximation as a lower bound of the ILP solutions. We notice that both the heuristic and the approximation are very close to the ILP on these small instances. We thus assume that for large instances the theoretical approximation will also be relevant. Figure 5b compares the computational time, which is the time taken to find a solution, between the ILP and the heuristic. We see that our heuristic is around 1000 times faster than the ILP. For small instances, the ILP computational time is already high which explains the impossibility to use it on large instances.

2) Evaluation on large instances: Then, we evaluate our heuristic on large instances to see how it behaves. We start by evaluating the impact of the centrality computation step of the heuristic. To do it we choose nodes to deploy in the

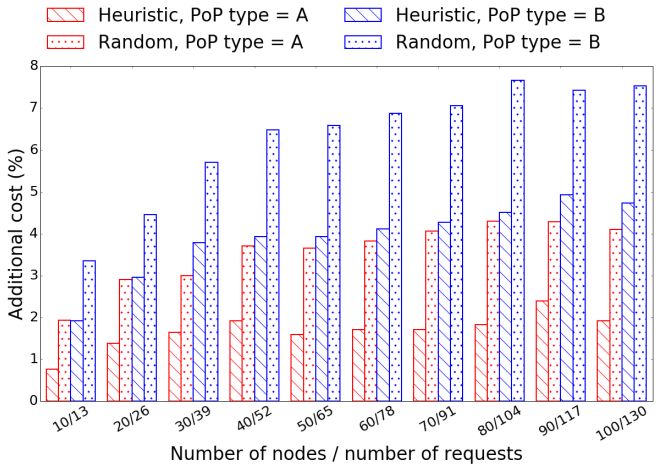


(a) Deviation to ILP.

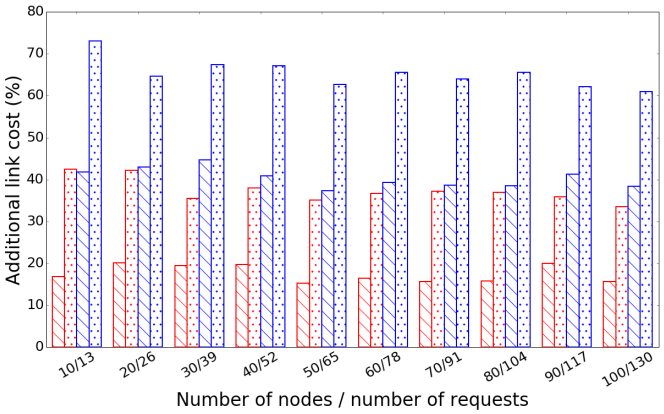


(b) Computational time (s).

Fig. 5. Comparison between the heuristic and the ILP on small instance (10 nodes) - VNF cost = \$2500, link cost = \$10, PoPs of type A.



(a) Total cost deviation.



(b) Link cost deviation.

Fig. 6. Deviation to the theoretical approximation on large instances - VNF cost = \$2500, link cost = \$10.

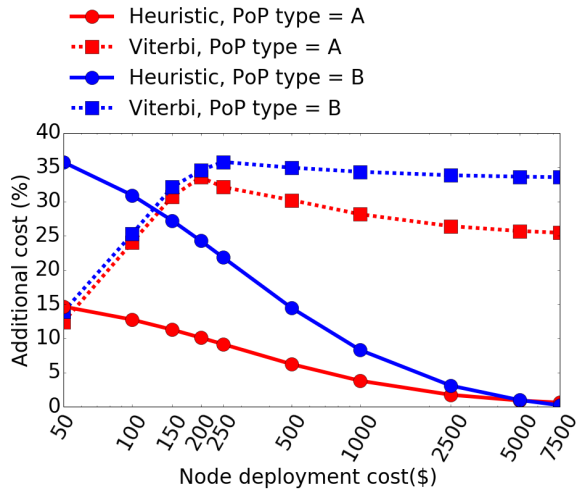
second step at random instead of using our centrality criterion. On Figure 6, we call $random$ the heuristic where we modify the second step as explained and $heuristic$ our proposed heuristic. We vary the size of the instances by keeping the ratio $\frac{NumberOfNodes}{NumberOfRequests}$ constant. We study both the total additional cost and the additional link cost. The latter refers to the additional cost induces by link cost only. It allows seeing if request redirections are important. As we set a node deployment cost higher than link cost, focusing on this cost is a good mean to infer on average requests end-to-end delays. We compute additional costs by measuring the deviation to the theoretical approximation since the ILP cannot be run on such large instances. We see on Figure 6a that the second step of the heuristic reduces by a factor 2 the total additional cost compared to its randomized version ($random$). The second step of the algorithm that uses the centrality to select PoPs is thus important. Furthermore, we can notice that the total additional cost for the heuristic is low and does not exceed 4%.

We also compare the 2 PoP architectures. We can see that PoPs of type B give worst results than for type A. We can explain it as the first case induces less flexibility. On Figure 6b we see that our heuristic keeps link additional cost reasonable which does not exceed 40%. Note that it would be less compared to optimal solutions since the theoretical approximation is a lower bound of the ILP.

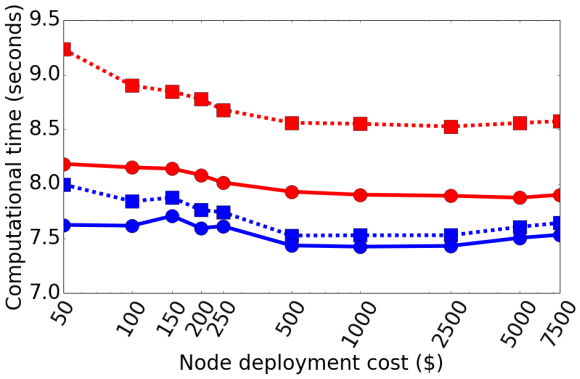
3) Comparison with state of the art heuristic: Finally, we compare our heuristic to the one proposed by Bari et al. [2] called $Viterbi$ on Figure 7. We fix the size of the instances to 50 nodes and 65 requests. We vary the VNF deployment cost between \$100 and \$7500. We look at the total additional cost and the computational time. Figure 7a shows that our heuristic gives lower additional costs than [2], especially for high VNF deployment costs. Computational durations are of the same order of magnitude for our heuristic and $Viterbi$, as shown on Figure 7b.

VI. Conclusion

Network Functions Virtualization is changing network architectures making the management of network functions more flexible. It becomes possible for ISPs to efficiently place and chain VNFs to reduce costs and serve all SFC requests. However, it is challenging to allocate multiple SFC



(a) Total cost deviation.



(b) Computational time (s).

Fig. 7. Comparison of our heuristic with the one, based on Viterbi algorithm, from Bari et al. [2] - 50 nodes, 65 requests, link cost = $10. Note that the x scale is purposely not linear.

requests on NFV Infrastructure, especially in a cost-driven objective. In this paper, we proposed a general formulation of the SFC allocation problem through an ILP. We then proposed a heuristic based on graph centrality and multi-stage graphs derived from the Viterbi algorithm. We implemented and compared them. The evaluation results show that the heuristic is around 1000 times faster than the ILP and provides SFC allocations whose total costs are very close to optimal solutions on small instances (less than 1.15% of difference). We also show that our heuristic scales well on larger instances and remains efficient. Finally, we compared it to one of the most appropriate state of the art heuristic and show that our approach outperforms it, especially when PoP deployment cost is higher than link bandwidth usage cost.

We plan to extend our approach to address SFC reliability. We intend to assign a reliability value on each PoP and set a reliability requirement to each SFC request. The challenge would then consist in adjusting the placement given by our heuristic to ensure reliability and propose a robust resource allocation, or in revisiting the heuristic to take into account specific reliability constraints.

Acknowledgment

This work was partially supported by the French ANR REFLEXION project (ANR-14-CE28-0019) and the Celtic-Plus SENDATE-TANDEM project.

References

[1] "Network functions virtualisation (NFV): Architectural framework (v1.1.1)," ETSI NFV ISG, Tech. Rep., 2013.

[2] M. F. Bari, S. R. Chowdhury, R. Ahmed, and R. Boutaba, "On orchestrating virtual network functions," in *Proceedings of the International Conference on Network and Service Management (CNSM)*, 2015, pp. 50–56.

[3] O. Biran, A. Corradi, M. Fanelli, L. Foschini, A. Nus, D. Raz, and E. Silvera, "A stable network-aware VM placement for cloud systems," in *In Proceedings of IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, 2012.

[4] J. Jiang, T. Lan, S. Ha, M. Chen, and M. Chiang, "Joint VM placement and routing for data center traffic engineering," in *Proceedings of IEEE INFOCOM*, 2012.

[5] N. Laoutaris, G. Smaragdakis, K. Oikonomou, I. Stavrakakis, and A. Bestavros, "Distributed placement of service facilities in large-scale networks," in *Proceedings of IEEE INFOCOM*, May 2007, pp. 2144–2152.

[6] Q. Zhang, Q. Zhu, M. Zhani, and R. Boutaba, "Dynamic service placement in geographically distributed clouds," in *Proceedings of IEEE Distributed Computing Systems (ICDCS)*, June 2012, pp. 526–535.

[7] A. Fischer, J. Botero, M. Till Beck, H. de Meer, and X. Hesselbach, "Virtual network embedding: A survey," *IEEE Communications Surveys Tutorials*, Fourth 2013.

[8] M. C. Luizelli, L. R. Bays, L. S. Buriol, M. P. Barcellos, and L. P. Gaspary, "Piecing together the NFV provisioning puzzle: Efficient placement and chaining of virtual network functions," *Proceedings of IFIP/IEEE International Symposium on Integrated Network Management (IM)*, 2015.

[9] B. Addis, D. Belabed, M. Bouet, and S. Secci, "Virtual network functions placement and routing optimization," in *Proceedings of IEEE International Conference on Cloud Networking (CloudNet)*, Oct 2015, pp. 171–177.

[10] H. Moens and F. De Turck, "VNF-P: A model for efficient placement of virtualized network functions," in *Proceedings of the International Conference on Network and Service Management (CNSM)*, 2014, pp. 418–423.

[11] W. Rankothge, J. Ma, F. Le, A. Russo, and J. Lobo, "Towards making network function virtualization a cloud computing service," in *IFIP/IEEE International Symposium on Integrated Network Management (IM)*, May 2015, pp. 89–97.

[12] M. Bouet, J. Leguay, T. Combe, and V. Conan, "Cost-based placement of vDPI functions in NFV infrastructures," *International Journal of Network Management*, Nov 2015.

[13] A. Basta, W. Kellerer, M. Hoffmann *et al.*, "Applying NFV and SDN to LTE mobile core gateways, the functions placement problem," in *Proceedings of CellNet*. ACM, 2014, pp. 33–38.

[14] M. Xia, M. Shirazipour *et al.*, "Network function placement for NFV chaining in packet/optical data centers," in *Proceedings of IEEE ECOC*.

[15] ILOG, Inc, "ILOG CPLEX: High-performance software for mathematical programming and optimization."

[16] P. Erdos and A. Renyi, "On random graphs i." *Publ. Math. Debrecen*, vol. 6, pp. 290–297, 1959.