An efficient algorithm for virtual network function scaling

Omar Houidi, Oussama Soualah, Wajdi Louati, Marouen Mechtri, Djamal Zeghlache, Farouk Kamoun

An Efficient Algorithm for Virtual Network Function Scaling

Omar Houidi*†, Oussama Soualah*, Wajdi Louati‡, Marouen Mechtri§, Djamal Zeghlache* and Farouk Kamoun†
*Telecom SudParis, Samovar-UMR 5157 CNRS, University of Paris-Saclay, France
§Orange Labs, Paris, France
‡ReDCAD Lab, University of Sfax, Tunisia
†CRISTAL Lab, National School of Computer Sciences, University of Manouba, Tunisia
Email: {omar.houidi, oussama.soualah, djamal.zeghlache}@telecom-sudparis.eu,
wajdi.louati@redcad.org, marouane.mechteri@orange.com, frk.kamoun@planet.tn

***Abstract*—Network Functions Virtualization (NFV) has been revolutionizing and improving the way networking services are deployed but also requires the dynamic scaling of resources during the lifecycle management of Virtual Network Functions (VNFs) with increasing service demand. We propose an Integer Linear Programming (ILP) approach and a Greedy heuristic to address this NP-Hard scaling problem. Both horizontal (scale out/in) and vertical (scale up/down) scaling are considered and the scalability of the ILP is addressed by reducing the number of candidate hosts in the search space. Extensive simulations evaluate the performance of the algorithms in successful scalings and VNF migrations and highlight the importance of using the algorithms to tidy up the infrastructure to accept more users. The results show that the ILP can outperform the Greedy heuristic if appropriate actions are selected.**

Keywords: Network Functions Virtualization, VNF Scaling, Migration.

I. Introduction

Network Functions Virtualization (NFV) has emerged as an innovative concept to reduce cost, improve flexibility and accelerate service deployment for network providers by decoupling functions (such as NATs, firewalls, load balancers, DNS...) from dedicated hardware and moving them to virtual servers. The SNS Telecom report [1] estimates that service provider SDN and NFV investments will grow at a Compound Annual Growth Rate (CAGR) of 46% between 2016 and 2020. As service providers seek to reduce costs and virtualize their networks, these investments will eventually account for over $18 Billion in revenue by the end of 2020.

In NFV based environment, one of the most important challenges for providers is to efficiently allocate hosting resources to dynamic virtualized network services demands while increasing revenue. Elastic mechanisms and scaling algorithms are essential to improve adaptation and deployment of Virtual Network Functions (VNFs) in NFV infrastructures to support increasing traffic load and customer demands.

As introduced in [2] and [3], Virtual Network Function scaling is triggered by new client requirements and/or rising traffic load due, for example, to an increase in the user plane traffic or the need of allocating more resources to a VNF to avoid service interruption.

To enhance initial VNF placement with dynamic adaptation, three scaling mechanisms are defined in [3] and [4]: i) Horizontal scaling (scale out/in): Add/remove virtualized resources (e.g., VNF Components (VNFCs)), ii) Vertical scaling (scale up/down): Reconfigure the capacity/size of existing virtualized resources and iii) VNF migration.

To address this NP-Hard elasticity problem [5], we propose an exact algorithm and a Greedy heuristic. An Integer Linear Programming (ILP) formulation with a search space reduction is adopted to improve scalability. A Greedy algorithm searching for a scaling solution in the neighborhood of initial placement is also presented. Comparisons between the two approaches show that an adequately tuned and devised ILP can outperform Greedy solutions in terms of proportion of successful scalings.

Section II of this paper presents the related work. The system model is described in Section III. The proposals are introduced in Section IV. Section V reports the performance evaluation results.

II. Related work

We review the VNF scaling problem with emphasis on the dynamic expansion and adaptation of Virtualized Network Function-Forwarding Graphs (VNF-FGs). A VNF-FG corresponds to a set of networked VNFs providing chained network services with specified traffic steering and flow paths. The review covers consequently the related work on VNF-FG placement and related work on VNF adaptation and configuration through scaling and migration. Prior work on Virtual Network Embedding (VNE) adaptation using migration and scaling is also included in the presentation for completeness.

VNF-FG placement and chaining was formulated and solved as a Integer Linear Programming in [6], [7], [8], and [9]. These methods are efficient in finding exact solutions but are subject to combinatorial explosion and do not scale with problem size. Heuristic algorithms are typically proposed to scale better with problem size. A baseline Greedy algorithm [10], using a bipartite graph construction and matching techniques, for VNF-FG placement and chaining, solves the problem in two steps: first mapping VNFs on physical hosts and second steering inter-VNF traffic across the hosts. In all these previous works, VNF scaling and migration are not considered to address increasing requirements and load on network functions and hosting resources.

Migration has been used in the VNE context. Several dynamic algorithms using reconfiguration optimize substrate resources for virtual networks. Authors of [11] propose a path migration algorithm for reconfiguring and rerouting virtual links, unfortunately do not consider adaptation of the virtual nodes and thus find suboptimal solutions. An iterative algorithm, called Virtual Network Reconfiguration (VNRe), is presented in [12] to ensure load balancing among substrate

nodes and reduce the rejection rate caused by congested substrate links. In [13], the authors propose an adaptive optimization algorithm which selects only critical VN requests for reconfigurations. Despite this selection of most critical VN requests, computational cost and service disruptions remain important at large scale. The more important body of work on scaling and migrating services in VNE is not directly applicable to the VNF-FG placement problem. VNE and VNF graph placement are distinct and have different characteristics [9], [8], [14]. VNE requests are modeled by simple undirected graphs while VNF chains are more complex and contain both the VNFs to place and the traffic flows to steer between the VNF-FG end points.

Few works address virtual network function scaling and migration to ensure dynamic VNF chain placement for rising demand and traffic load. Authors of [3] propose a consolidation algorithm called Simple Lazy Facility Location (SLFL) that optimizes the placement of the VNF instances in response to on-demand workload. SLFL chooses the VNF instances to be migrated on the basis of the instantaneous reconfiguration but does not assess the impact (induced benefits or penalties) of these decisions on future instants.

Bandwidth guaranteed VNF placement and scaling in Datacenter is considered in [15]. Leveraging the tree-like topology of Datacenter networks, this work proposes an on-line heuristic algorithm that achieves a near optimal allocation. Note that these two cited approaches do not take into account migration cost. In [5], authors proposed an Integer Linear Programming (ILP) model to solve the network function migration problem. In the rest of the paper, we denote this competitor algorithm by (ILP_C) that proposes a migration cost model and a heuristic algorithm to decrease the migration cost. Note that these algorithms do not consider scaling and elasticity to optimize resource utilization.

We address in this paper, through an ILP solution, the tradeoff between cost and performance when using VNF instances scaling and migration to adapt virtualized network functions graphs to increasing demand and workload.

III. PROBLEM FORMULATION

This section models the VNFs and the VNF-FG scaling problem and derives an ILP solution that ensures placement and scaling for increasing demands with minimal cost and service interruptions.

A. Substrate and virtual networks models

The physical network (commonly known as substrate and physical infrastructure and use interchangeably), as defined by the ETSI NFV Infrastructure (NFV-I) [16], is modeled as an undirected weighted graph, denoted by $G_p = (N_p, E_p)$ where E_p is the set of physical links and N_p is the set of physical nodes. Each substrate node, $k \in N_p$, is characterized by its i) available processing power (i.e., CPU) denoted by CPU_k, and ii) type T_k: switch, server or Physical Network Function (PNF) [17], where PNFs are the traditional physical middleboxes offering network functions. A PNF is a dedicated hardware that implements a network function. Each virtual link (i.e., $e \in E_p$) is characterized by its available bandwidth BW_e.

A client request (i.e., requested service function chain) is modeled as a directed graph $G_v = (N_v, E_v)$ where N_v is the set of virtual nodes and E_v is the set virtual links in the graph. Each virtual node, $i \in N_v$, is characterized by its i) required processing power cpu_i and ii) its type t_i: VNF or switch (i.e., ingress or egress). Each virtual link $(ij) \in E_v$ is described by its required bandwidth bw_{ij}. Note that the VNFs can be hosted only into server or PNFs having the same type. The ingress and egress nodes can be hosted only into switches.

B. Scaling Problem

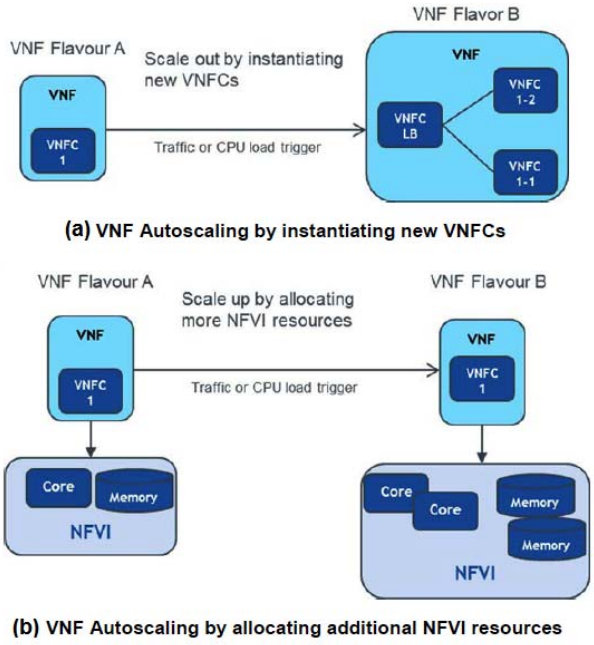


Fig. 1: VNF scaling [2]

Before the expiration of a tenant or user VNF-FG lifetime, the resources requirements of the VNFs in the forwarding graph can increase with rising demand. This will entail the spawning of new VNF instances, the allocation of additional hosting resources and possible the migration of the VNFs to other more capable physical hosts. For example, a firewall may need to install more filtering and control rules or handle more applications and users. Figure 1 depicts examples some of these scaling actions to respond to rising requirements and demand. ETSI [2] defines the type of scalings as follows:

1) When a VNF is **scaled out**, new VNF components (VNFCs) are instantiated and added to the VNF. Under such circumstances, the VNF may need a mechanism to distribute the load or traffic among the VNFCs (newly instantiated and existing VNFCs). This distribution can be accomplished through the use of load balancers (see figure 1(a)).
2) When a VNF is **scaled in**, one or more VNFCs of a VNF are terminated.
3) When a VNF is **scaled up**, it is assigned additional NFVI resources such as compute cores, memory, storage, or network resources (see figure 1(b)).
4) When a VNF is **scaled down**, NFVI resources that have been previously allocated to the VNF are de-allocated.

From the point of view of physical resource consumption, scaling out and scaling up can be treated the same way (i.e., increasing VNF scale is accomplished by scaling out or scaling up [2]) despite technical deployment differences. Both actions require additional resources and result in increased physical

resources consumption. Scaling out and up are taken into account in our proposed model and optimization algorithms.

IV. PROPOSALS

We propose an Integer Linear Programming solution and a Greedy algorithm to address the VNF scaling problem. We assume that the scaling concerns the i^{th} VNF in the VNF-FG. Hence, i is fixed (i.e., not variable) for each scaling request.

A. ILP Solution

We now formulate the ILP model for virtual network function scaling. Table I summarizes the parameters and the used variables. The model includes integrity constraints associated to the objective function used to achieve optimal scaling.

Node re-mapping constraint: Each scaled VNF i is re-mapped to exactly one physical candidate node k. A VNF (i.e., its components (VNFCs)) can not be split into (distributed across) many physical nodes. This is expressed by:

$$\sum_{k \in \mathcal{C}} x_{ik} = 1 \tag{IV.1}$$

where:

$$x_{ik} = \begin{cases} 1, & \text{if the VNF } i \text{ is re-mapped to physical node } k; \\ 0, & \text{otherwise.} \end{cases} \tag{IV.2}$$

Capacity constraint: This constraint ensures that the residual capacity in a physical node k satisfies the required capacity by VNF i. This leads to the following inequality:

$$cpu_i \times x_{ik} \leq CPU_k, \ \ \forall k \in \mathcal{C} \tag{IV.3}$$

Link re-mapping constraint: Each virtual link (ij) will be re-mapped to exactly one physical path $P_{k,m(j)}$ where k is a physical candidate for the i^{th} VNF and $m(j)$ is the initial mapping location of the j^{th} VNF. Note that i and j are neighbors in the VNF-FG request.

$$\sum_{k \in \mathcal{C}} y_{ij,P_{k,m(j)}} = 1, \ \ \forall (ij) \in E_N(i) \tag{IV.4}$$

where:

$$y_{ij,P_{k,m(j)}} = \begin{cases} 1, & \text{if the virtual link } (ij) \text{ is re-mapped} \\ & \text{to physical path } P_{k,m(j)}; \\ 0, & \text{otherwise.} \end{cases} \tag{IV.5}$$

Bandwidth constraint: Obviously the residual bandwidth in a physical path $P_{k,m(j)}$ has to satisfy the bandwidth requirement of virtual link (ij). We should not violate the remaining bandwidth of each physical link $e \in E_p$ on the physical path $P_{k,m(j)}$. This condition can be formally expressed as:

$$\sum_{(ij) \in E_N(i)} bw_{ij} \times y_{ij,P_{k,m(j)}} \times \delta_{ep} \leq BW_e, \quad \forall e \in P_{k,m(j)}, \ \forall P_{k,m(j)} \in \mathcal{P} \tag{IV.6}$$

Node and link re-mapping constraint: When a VNF i is re-mapped to a physical candidate node k, each virtual link (ij), starting from VNF i, has to be re-mapped to a physical path $P_{k,m(j)}$ having k as one of its endpoint or extremity. The other endpoint is $m(j)$ (i.e., where j is a neighbor of i in VNF-FG).

$$x_{ik} = y_{ij,P_{k,m(j)}}, \ \ \forall (ij) \in E_N(i), \forall k \in \mathcal{C} \tag{IV.7}$$

TABLE I: Main notations

Notation	Description
CPU_k	Residual capacity in a physical node k
$m(j)$	Initial mapping location of VNF j
$P_{k,m(j)}$	Physical path interconnecting physical nodes k and $m(j)$
$BW_{P_{k,m(j)}}$	Residual bandwidth in a physical path $P_{k,m(j)}$
$\mathcal{C}$	Set of physical node candidate
$\mathcal{P}$	Set of physical path candidate
min_{CPU}	The minimum capacity in $\mathcal{C}$
BW_e	Residual bandwidth in one physical link e
δ_{ep}	A binary coefficient determining whether physical link $e \in E_p$ belongs to path $P_{k,m(j)} \in \mathcal{P}$: $\delta_{ep} = 1 \Leftrightarrow e \in P_{k,m(j)}$
$E_N(i)$	Set of virtual links (ij) where j is a virtual neighbor of i
cpu_i	Required capacity by virtual node i
bw_{ij}	Required bandwidth between virtual nodes i and j
x_{ik}	A boolean variable indicating whether VNF i is re-mapped to physical node k
$y_{ij,P_{k,m(j)}}$	A boolean variable indicating whether virtual link (ij) is re-mapped to physical path $P_{k,m(j)}$
Δ_{t_r}	Remaining time of request before departure
δ_t	Required time to migrate one capacity unit
Rev_u	Revenue unit
Pen_u	Penalty unit

Objective function: We are interested in minimizing the service interruption time caused by scaling and favor for this reason new physical hosts with the most available resources. This should also maximize the revenue of the providers that can maintain services and tidy up their infrastructures to accept more users.

$$maximize \ Z$$

Where:

$$Z = (\Delta_{t_r} \times cpu_i \times Rev_u) \times \textstyle\sum_{k \in \mathcal{C}} \frac{CPU_k}{min_{CPU}} \times x_{ik} - (\delta_t \times cpu_i \times Pen_u) \times \textstyle\sum_{k \in \mathcal{C} \setminus \{m(i)\}} x_{ik} \tag{IV.8}$$

The addressed VNF scaling problem is summarized by lumping the objective function and the constraints:

$$\begin{aligned} \text{maximize} \quad & Z \\ \text{subject to:} \quad & \textstyle\sum_{k \in \mathcal{C}} x_{ik} = 1 \\ & cpu_i \times x_{ik} \leq CPU_k, \ \ \forall k \in \mathcal{C} \\ & \textstyle\sum_{k \in \mathcal{C}} y_{ij,P_{k,m(j)}} = 1, \ \ \forall (ij) \in E_N(i) \\ & \textstyle\sum_{(ij) \in E_N(i)} bw_{ij} \times y_{ij,P_{k,m(j)}} \times \delta_{ep} \leq BW_e, \\ & \forall e \in P_{k,m(j)}, \forall P_{k,m(j)} \in \mathcal{P} \\ & x_{ik} = y_{ij,P_{k,m(j)}}, \ \ \forall (ij) \in E_N(i), \forall k \in \mathcal{C} \end{aligned}$$

Problem 1: VNF scaling optimization summary

B. Heuristic Algorithm

We also propose a Greedy algorithm for VNF scaling and re-mapping to compare with our ILP and gain some insight on achievable performance, benefits and strengths of the exact modeling approach. The proposed Greedy algorithm operates using the following 5 steps:

1) check initial mapping $m(i)$ capacity: if the remaining resource in node $m(i)$ can satisfy the new required

capacity by VNF i, scaling occurs in the same physical node $m(i)$ (i.e., without migration). Else, use migration to scale;
2) find a set of neighboring physical candidate nodes $\mathcal{C}$ related to m(i);
3) check candidate node re-mapping: if node k capacity and type constraints are met, move to links capacities checking;
4) compute the shortest path between candidate k and all physical nodes $m(j)$ (where j is a virtual neighbor of i) using Dijkstra's algorithm based on available bandwidth. In fact, this corresponds to computing the best paths that maximize the minimum bandwidth along their route between k and $m(j)$;
5) check link re-mapping: if links capacities are respected, confirm candidate k as re-mapping solution.

Algorithm 1: Scaling Greedy pseudo-code

```
1  Inputs: G_p, VNF i, cpu_i
2  Output: VNF i scaled
3  m(i) ← GetInitialMapping(VNF i)
4  if CPU_{m(i)} ≥ cpu_i then
5      return Mapping(VNF i, cpu_i, m(i))
6  else
7      C ← ComputeNeighborCandidate(m(i), cpu_i)
8      stack ← Sort(C)
9      while stack ≠ ∅ and solution = False do
10         k ← Top(stack)
11         if CPU_k ≥ cpu_i and
12         ComputeLinkMapping(k, m(j)) = True then
13             solution ← True
14 return Mapping(VNF i, cpu_i, k)
```

Our Greedy algorithm has a complexity of $O(|\mathcal{C}| \times |N_p|^2 \times |E_N(i)|)$. The algorithm requires in step 9 $|\mathcal{C}|$ iterations in the worst case and a complexity $|N_p|^2$ in step 12 since the Dijkstra's algorithm is used to compute the shortest path needed to map a virtual link (ij) crossing the scaled VNF i. The Dijkstra algorithm is called $|E_N(i)|$ times and this leads to the $O(|\mathcal{C}| \times |N_p|^2 \times |E_N(i)|)$ overall complexity.

V. Performance Evaluation

The performance of our proposed ILP based is assessed through extensive simulations. Simulator settings and performance evaluation metrics used for evaluation and comparison purposes are presented. The ILP is compared to the Greedy heuristic. The evaluation focuses on the gains reconfiguration (just to tidy up the network) and scaling (during rising demands) provide to the stakeholders by analyzing the proportion of rejected (failed) adaptation attempts.

A. Simulation Environment

Our simulations are based on a realistic topology as well as extensive simulations for which it is possible to evaluate in depth the scalability of the algorithms using larger infrastructures and request sizes. A 2.50 GHz Quad Core server with 6 GBytes of available RAM is used for the performance evaluation and comparison of the proposed algorithms.

For the realistic topology and the extensive simulations, the VNF-FG requests are generated using a Poisson process with an average arrival rate of 5 requests per 100 time units. The lifetime of each request follows an exponential distribution with a mean of 1000 time units.

The Germany50 network topology [18] is used for the first assessment since it corresponds to the largest publicly available network topology (with 50 nodes). This topology is defined by the German National Research and Education Network (DFN). The capacity of physical nodes and links capacities are generated randomly in the $[50, 100]$ intreval. The size of the VNF-FG requests is set to 5 nodes. The GT-ITM [19] tool is used to generate the requested VNF-FG topologies. The initial VNF-FG computing and bandwidth requirements are set to 10. Scaling in CPU is fixed to 20 with one scaling request at a time per VNF-FG.

To assess the scalability of our proposed algorithms, we generate using the GT-ITM tool a network topology with 100 nodes and a connectivity of 0.2 (or 20%). The physical resources capacities (i.e., CPU and bandwidth) are also drawn randomly in the $[50, 100]$ interval. The VNF-FG requests size varies between 3 and 15 nodes. The required CPU for each VNF in the VNF-FG is set to 10 units. The required bandwidth between two VNFs, to ensure communication between them, is set also to 10 units. The connectivity between nodes in the VNF-FG is set to 0.3.

The ILP algorithm is evaluated for three values of the migration penalty Pen_u from 10 to 1000 units in order to tune the penalty according to needs and provider priority as well as measure its effect on performance. The migration penalty (Pen_u in the second term of the objective function in equation VI.8) is used to control the re-mapping process and to especially trade-off "in node scaling" with "migration to other hosts". The migration penalties in the performance evaluation correspond to the reported ILP10, ILP100 and ILP1000 results. The performance of the ILP for these penalty values (10, 100 and 1000) are also compared to the Greedy algorithm, to assess the effectiveness and usability of this migration. For the realistic topology and the simulation, 1000 VNF-FG requests are generated and a scaling request is triggered for each and every generated request. Since, we are focussing on scaling and adaptation of already embedded VNF-FGs, we used the heuristic approach of [10] to generate the initial VNF-FG mapping and initialize the assessment runs.

B. Performance Metrics

The metrics used for the performance evaluation are described in this section that reports also the results obtained using both the realistic topology and the extensive simulations for the extended performance assessment.

1) ***Successful scalings****:* represents the number of VNF scaling requests that have been accomplished. Indeed, due to physical resources limitation the algorithm may reject some scaling requests. From the provider point of view, this number should be maximized in order to improve the global revenue.

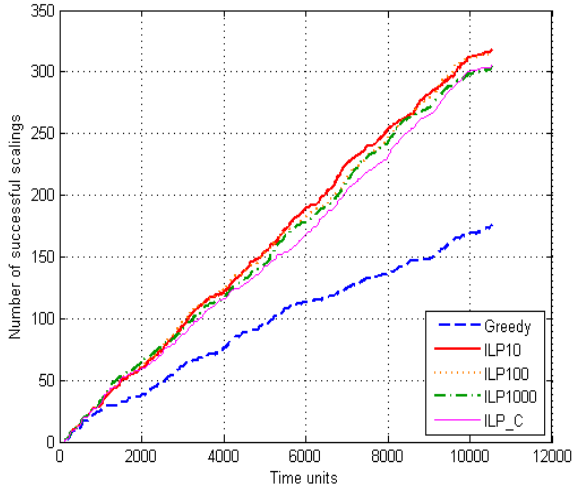
2) ***Scaling ratio****:* is the ratio of successful scalings defined by the ratio of successful scalings to the accepted VNF-FG requests. The number of accepted VNF-FG requests depends on the initial provisioning and on how the scaling algorithm deals with scaling requests. Since we use as initial mapping the same basis for all algorithms, the scaling ratio reflects their relative efficiency and enables their comparison.

*3) **Number of migrated VNFs**:* is the number of the scaled VNFs through migration. Migration involves a temporary service interruption until the concerned VNF is activated in the new physical host.

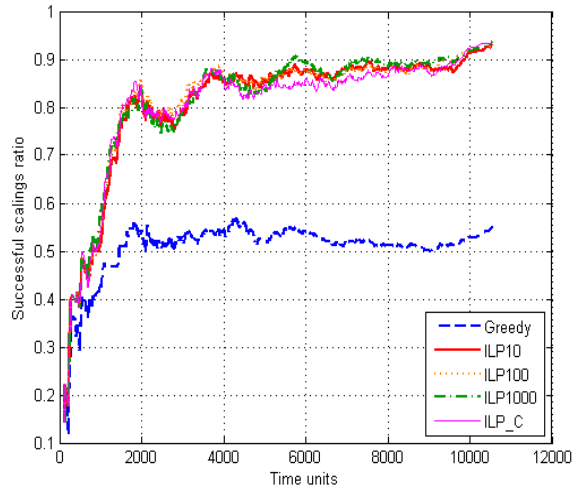
*4) **The ratio of migrated VNFs**:* is the ratio of the number of migrated VNFs to the number of successful scalings. This metric measures the proportion of adaptations accomplished through migration that providers prefer to minimize to avoid or limit the service interruptions due to migrations. Minimizing this measure reduces disruptions to applications.

*5) **Execution time**:* is a decisive measure in order to assess the scalability of the algorithms. Service providers prefer efficient and rapid algorithms in order to quickly serve clients.

C. Simulation Results



(a) Successful scalings

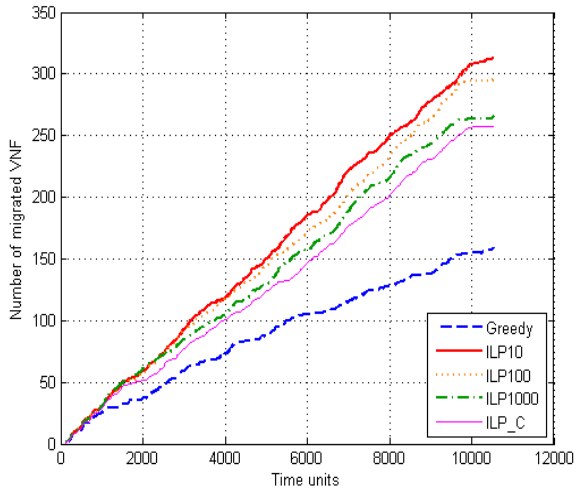


(b) Scalings ratio

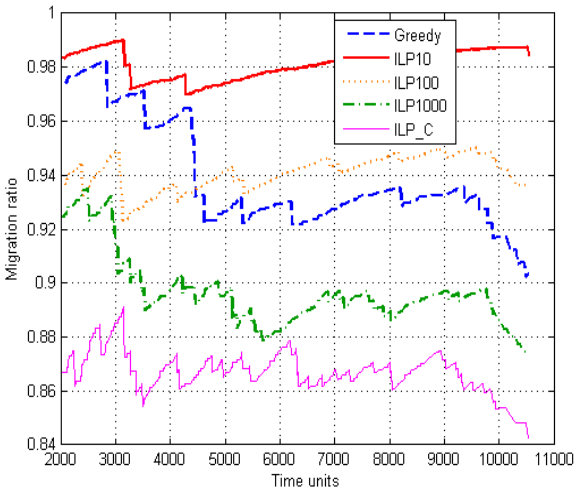
Fig. 2: Germany50 topology results: Scalings

1) Scenario 1: We use the Germany50 topology as a NFV-I (i.e., physical network) to conduct the first performance assessment. Figure 2(a) shows that the ILP algorithm, using the three migration penalty values, outperforms the Greedy solution. This is accomplished despite the fact that the ILP does not consider all possible candidates but just a subset for scalability reasons. The ILP satisfies in addition more scaling demands as confirmed by figure 2(b) where the scaling ratio exceeds 90% for the ILP variants and ILP_C. This proportion is only about 55% for the Greedy strategy.

Figure 3(a) depicts the number of the migrated VNFs. The ILP with a migration penalty of 10 performs more migrations than ILP1000 for which a higher migration penalty forces the scaling to occur within the nodes in priority. At the first glance, one could conclude that the Greedy algorithm and ILP_C use less migrations than the ILP1000 but figure 3(b) shows that the migration ratio of the Greedy strategy (about 90%) is higher than ILP1000 (roughly 87%) and ILP_C ($84,3\%$). If we analyze only the three ILP configurations, we observe as expected that ILP10 migrates almost all VNFs (more than 98%) to find more hosting resources while ILP100 migrates about 94% of the requests.



(a) VNF migration

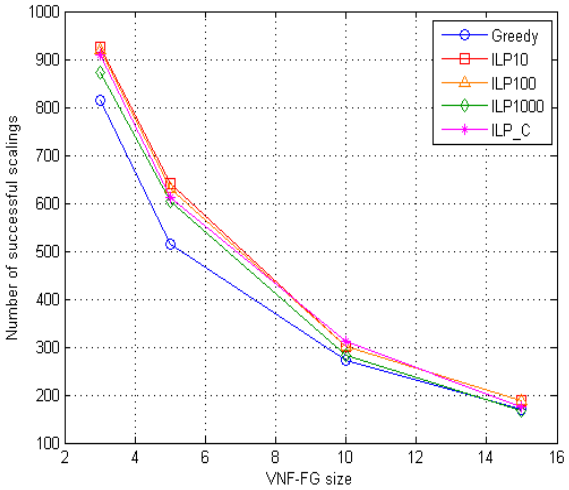


(b) Migration ratio

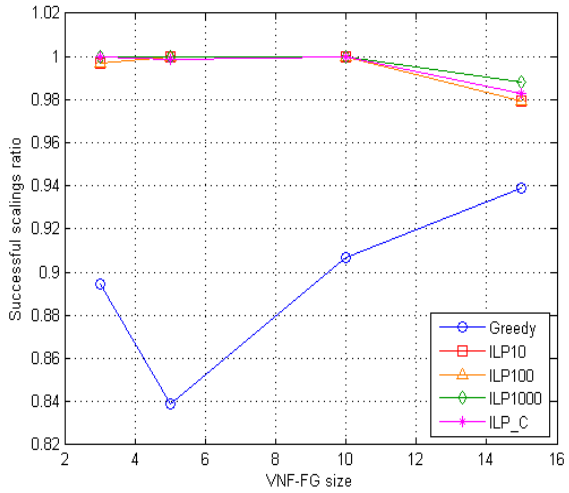
Fig. 3: Germany50 topology results: Migration

2) Scenario 2: To analyze the behavior of the algorithms and their scalability with problem size, we use larger NFV-Is (with 100 nodes) and initial VNF-FG request sizes with a number of VNFs in the $[3,15]$ interval. Figure 4(a) reports the number of successful scalings. The ILP algorithm outperforms

the Greedy heuristic by satisfying more scaling demands. Figure 4 (b) confirms this with scaling ratios exceeding 97% for the ILP variants and ILP_C. The Greedy strategy achieves ratios only between 84% and 94%.



(a) Successful scalings

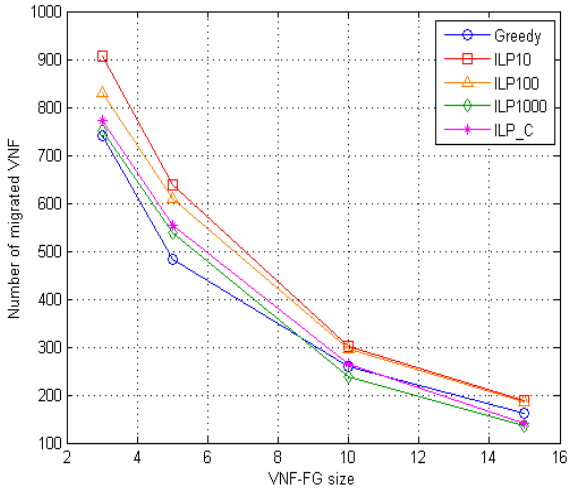


(b) Scalings ratio

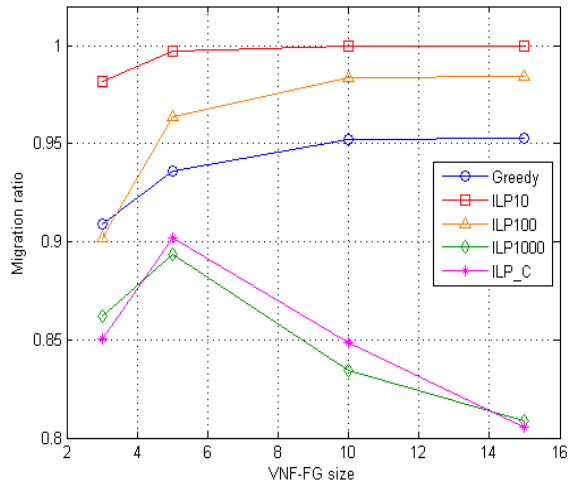
Fig. 4: Large scale scenarios: Scalings

Figure 5(a) illustrates the number of the migrated VNFs. ILP10 performs more migration than the ILP1000. ILP10 and ILP100 perform more migrations than the Greedy and ILP_C. ILP10 roughly migrates the totality (exactly 100% if VNF-FG sizes exceed 10) of the scaled VNFs compared to ILP100 that migrates $98,4\%$. Figure 5(b) shows that the migration ratio of the Greedy strategy (varies between 90% and 95%) is higher than ILP1000 and ILP_C (between 80% and 90%). Clearly the ILP can be tuned to outperform other algorithms as well as tradeoff migrations with in node scaling to fulfill the provider preferences and business interests and policies.

3) Importance of reconfiguration: In this subsection we focus on the gain that the reconfiguration process may provide. The provider can perform re-mapping of some VNFs re-mapping to tidy-up their physical networks and prevent bottlenecks and degradations in the NFV-I and in addition can host more client requests. Figure 6 depicts the rejection



(a) VNF migration



(b) Migration ratio

Fig. 5: Large scale scenarios: Migration

ratio, based on the initial embedding, of the ILP algorithm, the Greedy heuristic and the baseline solution (i.e., dealing with only the initial mapping). In this scenario there is no extra CPU demand but only the concerned VNF will be reconfigured. The baseline algorithm (i.e., without reconfiguration) rejects more requests (about 14%) compared to the Greedy heuristic and ILP_C (9%) and the ILP solution (7%). This result can be explained by the dynamics in the requests arrivals and departures that are exploited more efficiently and especially when departures occur and resources are released. The reconfiguration opportunistically tidies up the network and make room for new requests that would be otherwise rejected because of suboptimal use of the infrastructure.

TABLE II: Execution time (ms)

	$\|\mathcal{C}\|$=5	$\|\mathcal{C}\|$=10	$\|\mathcal{C}\|$=15	$\|\mathcal{C}\|$=20
Greedy	11,65	12,07	14,64	15,43
ILP	23,04	38,21	43,74	44,62
ILP_C	26,81	41,87	45,62	46,12

4) Average time resolution (Execution time): Table II reports the execution time performance of the algorithms to gain

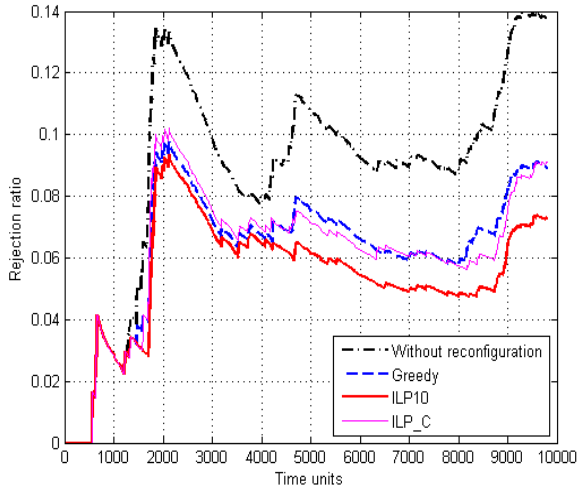


Fig. 6: Rejection Ratio

insight on their scalability and complexity with problem size. In order, to evaluate this metric, the substrate graph size is fixed at 100 nodes. We generate 1000 VNF-FG requests and vary the set of potential candidates size from 5 to 20. These results indicate that the Greedy algorithm has significantly better execution time when compared to our ILP algorithm and ILP_C execution times. However, the faster execution times of the Greedy algorithms are accomplished at the expense of the number of accepted scaling requests. Indeed, the ILP achieves 918 scalings and ILP_C 909 while the Greedy accepts only 815. The extra time taken by the ILP is due to the ability of the ILP to find more solutions and hence accept more scaling requests.

VI. Conclusion

Our work addresses the problem of virtual network function reconfiguration with the main objective of reducing the rejection of VNF scaling requests when faced with rising demand and traffic load. We proposed an ILP solution and a new Greedy reconfiguration algorithm for the purpose and show that the solutions can efficiently scale virtualized network functions and forwarding graphs with good execution time and scaling requests acceptation performance over distributed and dynamically varying cloud environments.

References

[1] "SNS Telecom, The SDN, NFV & Network Virtualization Ecosystem: 2016 - 2030 - Opportunities, Challenges, Strategies & Forecasts," Tech. Rep., 2016. [Online]. Available: http://www.snstelecom.com/sdn-nfv

[2] ETSI GS NFV-TST 001: "Network Functions Virtualisation (NFV); Pre-deployment Testing; Report on Validation of NFV Environments and Services".

[3] M. Ghaznavi, A. Khan, N. Shahriar, K. Alsubhi, R. Ahmed, and R. Boutaba, "Elastic virtual network function placement," in *IEEE CloudNet*, 2015, pp. 255–260. [Online]. Available: http://dx.doi.org/10.1109/CloudNet.2015.7335318

[4] G. Galante and L. C. E. D. Bona, "A survey on cloud computing elasticity," in *IEEE Fifth International Conference on Utility and Cloud Computing, UCC 2012, Chicago, IL, USA, November 5-8, 2012*, 2012, pp. 263–270. [Online]. Available: http://dx.doi.org/10.1109/UCC.2012.30

[5] J. Xia, Z. Cai, and M. Xu, "Optimized virtual network functions migration for NFV," in *22nd IEEE International Conference on Parallel and Distributed Systems, ICPADS 2016, Wuhan, China, December 13-16, 2016*, 2016, pp. 340–346. [Online]. Available: http://dx.doi.org/10.1109/ICPADS.2016.0053

[6] M. F. Bari, S. R. Chowdhury, R. Ahmed, and R. Boutaba, "On orchestrating virtual network functions in NFV," *CoRR*, vol. abs/1503.06377, 2015. [Online]. Available: http://arxiv.org/abs/1503.06377

[7] A. Gember, A. Krishnamurthy, S. S. John, R. Grandl, X. Gao, A. Anand, T. Benson, A. Akella, and V. Sekar, "Stratos: A network-aware orchestration layer for middleboxes in the cloud," *CoRR*, vol. abs/1305.0209, 2013. [Online]. Available: http://arxiv.org/abs/1305.0209

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

[9] S. Mehraghdam, M. Keller, and H. Karl, "Specifying and placing chains of virtual network functions," in *Cloud Networking (CloudNet), 2014 IEEE 3rd International Conference on*, Oct 2014, pp. 7–13.

[10] M. Mechtri, C. Ghribi, and D. Zeghlache, "Vnf placement and chaining in distributed cloud," in *the 9th IEEE International Conference on Cloud Computing, June 27 - July 2, 2016, San Francisco, USA*.

[11] M. Yu, Y. Yi, J. Rexford, and M. Chiang, "Rethinking virtual network embedding: substrate support for path splitting and migration," *SIGCOMM Comput. Commun. Rev.*, vol. 38, pp. 17–29, 2008.

[12] I. Fajjari, N. Aitsaadi, G. Pujolle, and H. Zimmermann, "VNR Algorithm: A Greedy Approach For Virtual Networks Reconfigurations," in *GLOBECOM*. IEEE, 2011, pp. 1–6.

[13] Y. Zhu and M. Ammar, "Algorithms for assigning substrate network resources to virtual network components," *IEEE INFOCOM*, pp. 1–12, 2006.

[14] R. Guerzoni, R. Trivisonno, I. Vaishnavi, Z. Despotovic, A. Hecker, S. Beker, and D. Soldani, "A novel approach to virtual networks embedding for SDN management and orchestration," in *2014 IEEE Network Operations and Management Symposium, NOMS 2014, Krakow, Poland, May 5-9, 2014*, 2014, pp. 1–7. [Online]. Available: http://dx.doi.org/10.1109/NOMS.2014.6838244

[15] F. Wang, R. Ling, J. Zhu, and D. Li, "Bandwidth guaranteed virtual network function placement and scaling in datacenter networks," in *34th IEEE International Performance Computing and Communications Conference, IPCCC 2015, Nanjing, China, December 14-16, 2015*, 2015, pp. 1–8. [Online]. Available: http://dx.doi.org/10.1109/PCCC.2015.7410276

[16] ETSI GS NFV 001: "Network Functions Virtualisation (NFV); Use Cases".

[17] ETSI GS NFV 003: "Network Functions Virtualisation (NFV); Terminology for Main Concepts in NFV".

[18] S. Orlowski, M. Pióro, A. Tomaszewski, and R. Wessäly, "SNDlib 1.0–Survivable Network Design Library," in *Proceedings of the 3rd International Network Optimization Conference (INOC 2007), Spa, Belgium*, April 2007, http://sndlib.zib.de.

[19] E. Zegura, K. Calvert, and S. Bhattacharjee, "How to model an internetwork," *Proceedings of IEEE INFOCOM*, pp. 594–602, 1996.