Virtual Network Function Scaling

3.1 Introduction

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 Virtualised Network Functions (VNFs) in NFV infrastructures to support increasing traffic load and customer demands.

As introduced in [80] and [67], 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 [67] and [81]:

- *Horizontal scaling (scale out/in)*: Add/remove virtualized resources (e.g., VNF Components (VNFCs)).
- *Vertical scaling (scale up/down)*: Reconfigure the capacity/size of existing virtualized resources.

- *VNF Migration*: move VNF components from one hardware platform onto a better platform while still satisfying the service continuity requirement.

To address this NP-Hard elasticity problem [82], 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 an 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.

In [82], authors propose an Integer Linear Programming (ILP) model to solve the network function migration problem. In the rest of this chapter, we denote this competitor algorithm by (ILP_C). It 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.

The system model is described in Section 3.2. The proposals are introduced in Section 3.3. Section 3.4 reports the performance evaluation results and finally, we summarize the results with some future research directions in Section 3.5.

3.2 Problem Formulation

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

3.2.1 Substrate and virtual network models

The physical network (commonly known as substrate or physical infrastructure. The terms are used interchangeably), as defined by the ETSI NFV Infrastructure (NFV-I) [83], 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) [84], where PNFs are the traditional physical middleboxes offering network functions. A PNF is a dedicated hardware that implements a network function. Each physical link (i.e., $e \in E_p$) is characterized by its available bandwidth BW_e.

A client request (i.e., requested service function chain, also called VNF-FG) 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 in server or PNFs having the same type. The ingress and egress nodes can be hosted only in switches.

3.2.2 Scaling Problem

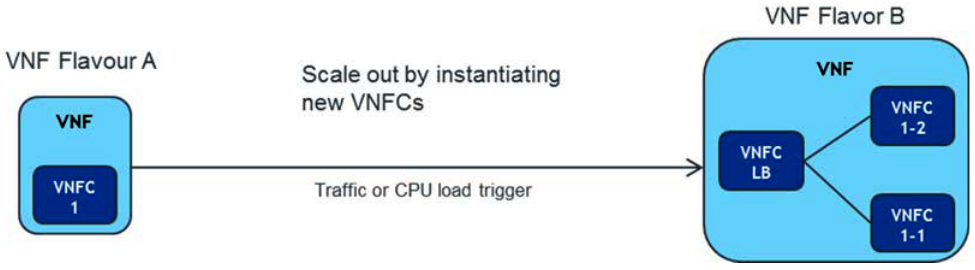


FIGURE 3.1: VNF Autoscaling by instantiating new VNFCs

Before the expiration of a tenant (i.e., a group of users who share a common access with specific privileges) or user VNF-FG lifetime, the resource 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. Figures 3.1 and 3.2 depict examples some of these scaling actions to respond to rising requirements and demand. ETSI [80] defines the types of scalings as follows:

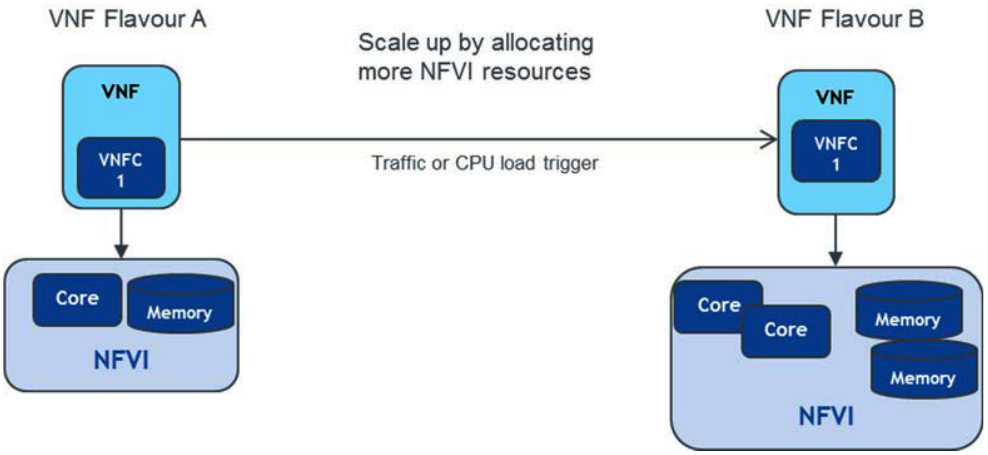


FIGURE 3.2: VNF Autoscaling by allocating additional NFVI resources

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 3.1).

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 3.2).

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 [80]) 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.

3.3 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 single VNF scaling request is triggered for each and every generated request.

3.3.1 ILP Formulation

We formulate the ILP model for virtual network function scaling. Table 3.1 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:

$$\sum_{k \in \mathcal{C}} x_{ik} = 1 \tag{3.3.1}$$

where:

$$x_{ik} = \begin{cases} 1, & \text{if the VNF } i \text{ is re-mapped to physical node } k; \\ 0, & \text{otherwise.} \end{cases} \tag{3.3.2}$$

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:

$$cpu_i \times x_{ik} \leq CPU_k, \quad \forall k \in \mathcal{C} \tag{3.3.3}$$

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. We use the $\mathcal{K}$-shortest path algorithm to calculate $\mathcal{P}$: the set of $\mathcal{K}$-shortest paths candidates

TABLE 3.1: 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 nodes candidates
$\mathcal{P}$	Set of physical paths candidates
min_{CPU}	The minimum capacity in $\mathcal{C}$
BW_e	Residual bandwidth in one physical link e
δ_{ep}	A boolean 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 binary variable indicating whether VNF i is re-mapped to physical node k
$y_{ij,P_{k,m(j)}}$	A binary 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

$P_{k,m(j)}$. We extend the Dijkstra algorithm based on residual bandwidth to find more than one candidate path $P_{k,m(j)}$ interconnecting physical nodes k and $m(j)$. Then, we supply the set $\mathcal{P}$ to the ILP.

$$\sum_{k \in \mathcal{C}} y_{ij,P_{k,m(j)}} = 1, \quad \forall (ij) \in E_N(i) \tag{3.3.4}$$

where:

$$y_{ij,P_{k,m(j)}} = \begin{cases} 1, & \text{if the virtual link } (ij) \text{ is re-mapped to physical path } P_{k,m(j)}; \\ 0, & \text{otherwise.} \end{cases} \tag{3.3.5}$$

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:

$$\sum_{(ij)\in E_N(i)} bw_{ij} \times y_{ij,P_{k,m(j)}} \times \delta_{ep} \leq BW_e, \quad \forall e \in P_{k,m(j)},\ \forall P_{k,m(j)} \in \mathcal{P} \tag{3.3.6}$$

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). Let us call $\mathcal{P}_{k,m(j)}$ the set of candidate paths between k and $m(j)$. Then, we define a binary variable for each of the candidate path $P_{k,m(j)} \in \mathcal{P}_{k,m(j)}$. This constraint can be written as:

$$x_{ik} = \sum_{P_{k,m(j)}\in\mathcal{P}_{k,m(j)}} y_{ij,P_{k,m(j)}}, \quad \forall (ij) \in E_N(i), \forall k \in \mathcal{C} \tag{3.3.7}$$

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. A way to reduce the complexity of the ILP and to scale much better with network size is to explore not all the candidate nodes space. That is cutting the exploration to a subset of the candidates, especially candidate nodes that are less loaded. This should also maximize the revenue of the providers that can maintain services and tidy up their infrastructures to

accept more users.

$$
\begin{aligned}
&maximize\ Z \\
&\text{Where:} \\
&Z = (\Delta_{tr} \times cpu_i \times Rev_u) \times \textstyle\sum_{k \in \mathcal{C}} \frac{CPU_k}{min_{CPU}} \times x_{ik} \\
&\quad - (\delta_t \times cpu_i \times Pen_u) \times \textstyle\sum_{k \in \mathcal{C} \setminus \{m(i)\}} x_{ik}
\end{aligned}
\tag{3.3.8}
$$

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

maximize Z

subject to:

$$\textstyle\sum_{k \in \mathcal{C}} x_{ik} = 1$$

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

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

$$\textstyle\sum_{(ij) \in E_N(i)} bw_{ij} \times y_{ij,P_{k,m(j)}} \times \delta_{ep} \leq BW_e,$$

$$\forall e \in P_{k,m(j)}, \forall P_{k,m(j)} \in \mathcal{P}$$

$$x_{ik} = \textstyle\sum_{P_{k,m(j)} \in \mathcal{P}_{k,m(j)}} y_{ij,P_{k,m(j)}}, \ \ \forall (ij) \in E_N(i), \forall k \in \mathcal{C}$$

PROBLEM 1: VNF scaling optimization summary

3.3.2 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). These candidate nodes are also sorted by available resources;

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 ← Pop(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 (Algorithm 1) $|\mathcal{C}|$ iterations in the worst case and a complexity $|N_p|^2$ (in the worst case) in Step 12 (Algorithm 1) 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 Fibonacci heaps can improve the complexity of the Dijkstra algorithm to be $O(|E_p| + |N_p| \times \log |N_p|)$. 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 in the worst case.

3.4 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.

3.4.1 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 [85] is used for the first assessment (with 50 nodes). This topology is defined by the German National Research and Education Network (DFN). The capacity of physical nodes and links are generated randomly in the $[50, 100]$ interval. The size of the VNF-FG requests is set to 5 nodes. The GT-ITM [86] 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 resource capacities (i.e., CPU and bandwidth) are also drawn randomly in the $[50, 100]$ interval. The VNF-FG request sizes vary 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 the needs and provider priority as well as to measure its effect on performance. The migration penalty (Pen_u in the second term of the objective function in Equation 3.3.8) 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 single VNF 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 [58] to generate the initial VNF-FG mapping and initialize the assessment runs.

3.4.2 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.

- **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.
- **Scaling ratio** is the ratio of successful scalings defined by the number 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.

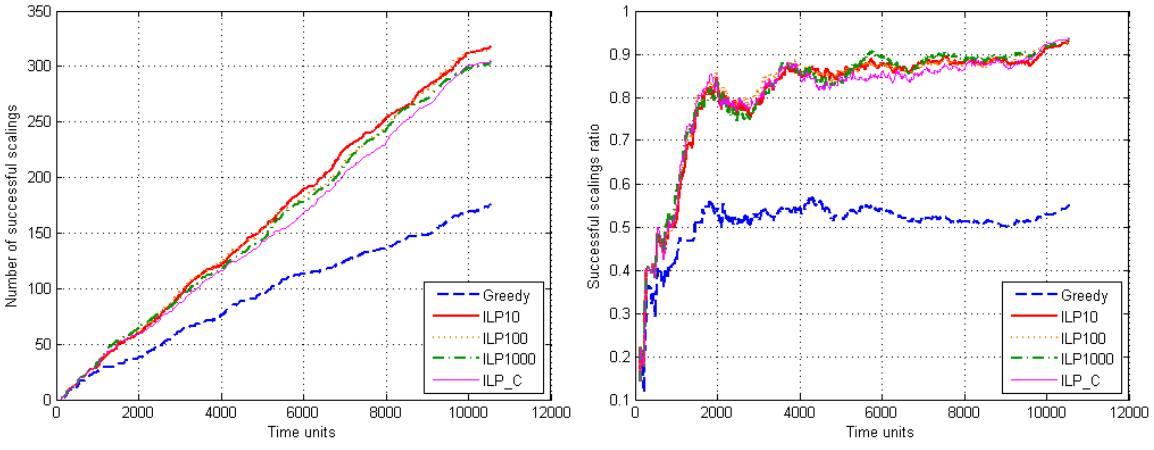
- **Number of migrated VNFs** is the number of the scaled VNFs through migration. Migration involves a temporary service interruption until the concerned VNF is activated in the new physical host.

- **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.

- **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.

3.4.3 Simulation Results

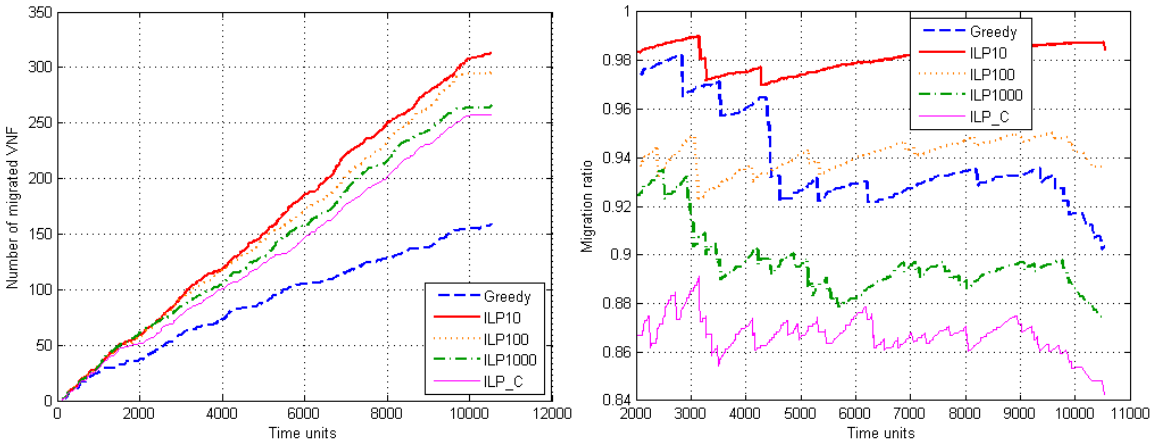
3.4.3.1 Evaluation on a realistic topology



(a) Successful scalings (b) Scalings ratio

FIGURE 3.3: Germany50 topology results: Scalings

We use the Germany50 topology as an NFV-I (i.e., physical network) to conduct the first performance assessment. Figure 3.3(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



(a) VNF migration (b) Migration ratio

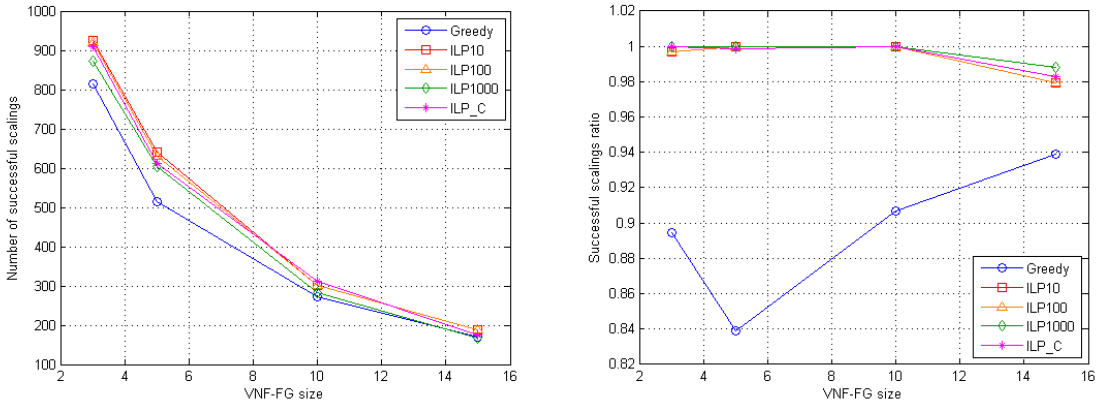
FIGURE 3.4: Germany50 topology results: Migration

reasons. The ILP satisfies in addition more scaling demands as confirmed by figure 3.3(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 3.4(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 3.4(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.

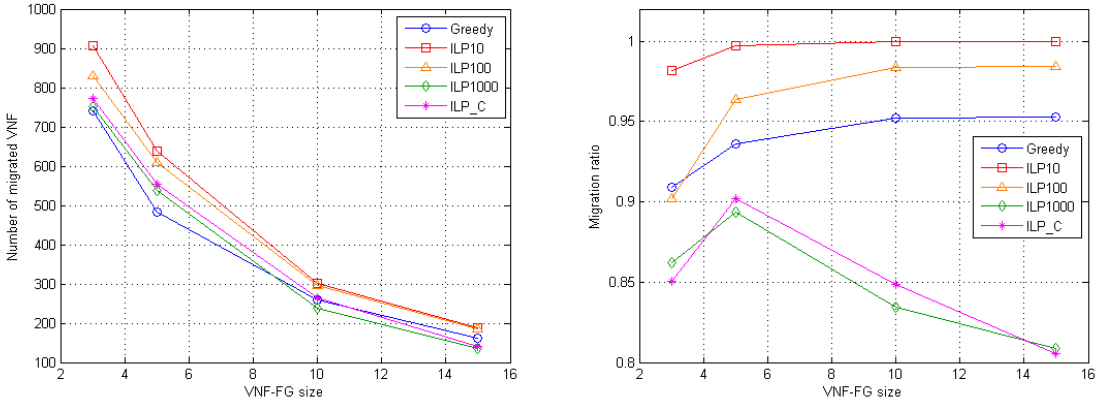
3.4.3.2 Large-scale evaluation

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 3.5(a) reports the number of successful scalings. The ILP algorithm outperforms the Greedy heuristic by satisfying more scaling demands. Figure 3.5 (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

FIGURE 3.5: Large scale scenarios: Scalings



(a) VNF migration (b) Migration ratio

FIGURE 3.6: Large scale scenarios: Migration

Figure 3.6(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 3.6(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.

3.4.3.3 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 3.7 depicts the rejection 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 model (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.

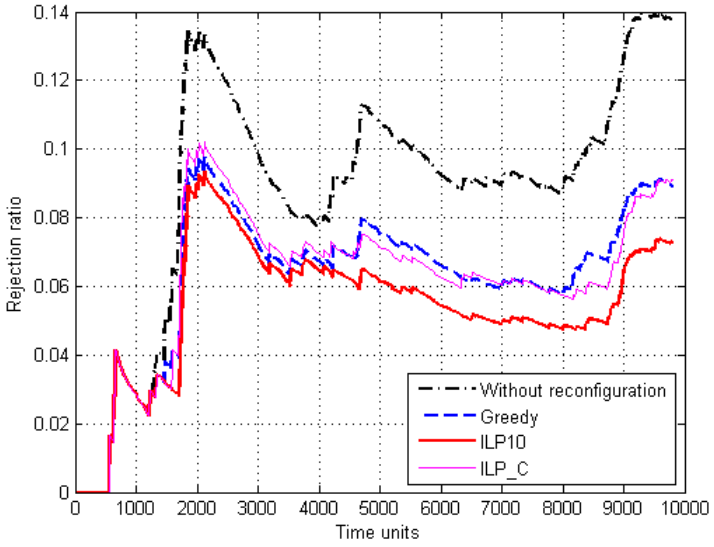


FIGURE 3.7: Rejection Ratio

3.4.3.4 Average time resolution (Execution time)

Table 3.2 reports the execution time performance of the algorithms to gain insight on their scalability and complexity with problem size. In order, to evaluate this metric, the substrate

TABLE 3.2: 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	$26,81$	$41,87$	$45,62$	$46,12$

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.

3.5 Conclusion

This chapter 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 model 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.

Chapter 4

Dynamic VNF Forwarding Graph Extension Algorithms

4.1 Introduction

Network Function Virtualization (NFV) coupled with Software-Defined Networking (SDN) is expected to impact the business to business to consumer relationships, current business models and value chains by enabling requests from multiple tenants for virtual infrastructures to deploy and provide their services to other providers and consumers.

Providers that deploy this variety of services and network functions in both physical and virtual infrastructures on behalf of tenants will have to treat via multiple network functions traffic originating, transiting and terminating in their infrastructures. The service provider has dynamic SFC establishment, update, and extension requirements and needs that the infrastructure providers have to meet. The latter will consequently need not only to remove, replace, insert, and add VNF instances, but also to extend already instantiated service chains without disturbing the previously deployed chains.

In this foreseen context, network providers (or physical infrastructure providers and more generally NFV architecture designers and actors) are actually faced with dynamic and variable demands of virtualized network infrastructures from tenants. These actors are indeed

likely to deploy their services, hosted in virtualized infrastructures, gradually as their business grows. These players not only need to control, classify, and steer user and application traffic flows in their dedicated slices but also want to extend their already acquired and operational slices with additional service graphs (adding new forwarding paths, new service chains, new services and virtualized network functions). These extensions of already hosted network function graphs have to be achieved without disrupting initially deployed and active service instances. The extensions must be seamless to applications and services that do not tolerate interruptions, migration or disruptions in quality of service (QoS) and experience.

In order to meet these requirements, this chapter proposes algorithms to extend already deployed network services and function graphs to respond to new demands while taking into account the constraint of minimizing the effect on the original service graphs.

According to [67] and [80], Virtual Network Function scaling and extension are triggered by new client requirements and the rising of network load over time. This is especially true for services with periodic resource demands (e.g., cloud applications such as an online shopping store during holiday seasons, etc.), and managing VNF workload changes [87] (e.g., during peak hours for cellular telecom operators).

In this chapter, we address the more generic and general problem of extending already deployed service function chains (or VNF-FGs) and aim at providing algorithms that can achieve various types of extensions. The latter may be a request to insert a new VNF in an existing graph, a new demand to extend a Network Forwarding Path (NFP) by adding VNFs to an existing chain, and more generally extend an already deployed VNF-FG with a complex service graph.

We refer to the graph extension as a VNF-FG extension to emphasize that we also address networking level extension of already deployed forwarding graphs in addition to the fact that we inherently answer the needs for service function chaining extensions. The extensions, as mentioned, can be triggered by new requests or by the VNF-FG life cycle management that may decide to extend the initial graph to adapt to network and traffic load changes such as adding load balancers, spawning new VNFs to absorb increasing load, etc.

We first propose an Integer Linear Programming (ILP) model as an exact formulation of the VNF-FG extension problem. The ILP can find optimal solutions for reasonable problem sizes and thus can serve as a reference for the performance evaluation and as a way to assess the quality of the solutions for heuristic algorithms at least for small graph sizes. The heuristics should be efficient, find good solutions, and scale with problem size much better than the ILP. To reduce complexity while finding good solutions, we resort to a Steiner Tree based algorithm and an Eigendecomposition algorithm using network adjacency matrices of the requested and hosting graphs as a basis for optimal embedding. The ILP and the heuristic algorithms ensure that the specified connecting points between the previously described graphs and the extensions are respected as specified in the client requests and that there are no disruptions (or minimal disruption) to the initially deployed VNF-FG. We compare the performance of these algorithms in terms of rejection rate of new requests, quality of the solutions and service disruptions as a function of infrastructure size, network connectivity, and system load.

Section 4.2 of this chapter presents the related work. The problem formulation is described in Section 4.3. The proposals are introduced in Section 4.4. Section 4.5 reports the performance evaluation results. Section 4.6 summarizes the main findings.

4.2 Related Work

Since we are addressing the extension of service function chains already deployed in hosting infrastructures, the related work review focusses on the virtual network topology change problem with emphasis on the dynamic expansion and adaptation of Virtualized Network Function-Forwarding Graphs (VNF-FGs) whenever relevant. Indeed, previous work addressed partially or simply did not address extension of already deployed services but rather initial placement, adaptation to changes of an existing graph and/or the hosting infrastructure and reaction to faults. We consequently limit the description to work as close as possible to ours while being aware that extensions of SFCs or VNF-FGs have not been directly and explicitly and fully addressed in the past.

In [88], authors present JASPER, a fully automated approach to jointly optimize scaling, placement, and routing decisions for complex network services. Two algorithms are developed for adaptation of existing services to changes in the demand, a Mixed Integer Linear Program (MILP) and a custom constructive heuristic.

Ayoubi et al. [89] introduced RELIEF, an availability-aware embedding and reconfiguration framework for elastic services in the cloud. The framework comprises two main modules. The JENA sub-system that performs virtual network (VN) embedding to provide just-enough availability guarantees based on the availability of the physical servers hosting the virtual one. The second module, ARES, is a reliable reconfiguration bloc that adapts the embedding of hosted services as they scale (by migrating the VN or adding backup nodes). This work does not address either, explicitly, graph extension requests from already served clients or increasing workloads which is our instead our central concern or objective.

Liu et al. [90] consider the readjustment of VNF chaining in dynamic situations and conditions by trying to optimize jointly the deployment of new SFCs and the readjustment of existing service chains in order to satisfy variable user requirements. This is closer to some extent to our work and stated goal. However, unlike our approach, they use pre-calculated paths in the network. We are focussing on the extension of already deployed chains and service graphs while keeping the initially provided and operating tenant graph unmodified, except for edges involved in the extension. The authors of [90] use first an ILP formulation to solve the problem exactly then reduce time complexity using a Column Generation (CG) heuristic algorithm that approximates the performance of the ILP. The main idea behind the CG based algorithm is to decompose the original problem into a master problem and a sub-problem to solve them iteratively in order to obtain a near-optimal solution.

In [91], authors present a comprehensive analysis of a resource allocation algorithm for VNFs based on genetic algorithms (GA) for the initial placement of VNFs and the scaling of VNFs to support traffic changes. For the scaling of existing policies proposed in [91], the algorithm starts with the current state and searches for the re-assignment of resources for the set of VNFs that need scaling. Therefore, some VNFs change their initial locations and the algorithm tries to minimize the number of server and link configuration changes (to minimize disruptions to existing traffic).

Li et al. [92] implemented a real-time resource provisioning system for NFV called NFV-RT, which integrates timing analysis with several techniques, such as service chain consolidation, and Integer Linear Programming with rounding. Also, the scaling is achieved by duplicating full instances of VNF chains, and by reshuffling chains and migrating traffic.

The proposals defined in this chapter differs from previous work and that of [91] and [92]. They actually address the problem of **Resource and Function Scaling** defined in [93] as the adaptation of the assigned network resources to functions or flows, or adaptation of the number of deployed instances of a specific VNF (by scale out, scale in, or migration). We are instead concerned by another type of flexibility, specifically **Topology Adaptation**, defined also in [93], by extending the graph structure of already active virtual networks that require an extension with additional nodes and links. Our addressed graph extension can contain new network and service functions and is far more complex than simply duplicating the service chain by a graph replication technique as proposed in [92]).

Note that all our algorithms, in this chapter, maintain the initial mapping location of already instantiated service chains (original VNF-FG) to avoid disturbing previously deployed and running services (i.e., without migrations or interruptions), to preserve previous deployments and ensure network stability.

In summary, our work differs significantly from the existing VNFs scaling and adaptation proposals since none really addresses the problem of extending an already deployed tenant VNF-FGs following a new request for seamless (non disrupting) evolutions of these dedicated service chains (i.e., maintaining acceptable service performance).

4.3 Problem Formulation

The problem of extending an already deployed and operating tenant SFC or VNF-FG corresponds to the placement of new requested VNFs and flow paths in the hosting infrastructure. This has to be accomplished while respecting all previous deployments and the specified connectivity between the initial graph and its extension. This is a placement problem with a set of very specific constraints. The placement of virtual functions and path extensions