TNOVA

NETWORK FUNCTIONS AS-A-SERVICE
OVER VIRTUALISED INFRASTRUCTURES

GRANT AGREEMENT NO. 619520

Deliverable D3.3

Service mapping

Editor F. Liberati (CRAT)

Contributors Francesco Liberati, Federico Cimorelli, Francesco Delli Priscoli, Alessandro Giuseppi, Saverio Mascolo, Antonio Pietrabissa, Vincenzo Suraci, Letterio Zuccaro (CRAT), Alberto Ceselli, Alessandro Petrini, Marco Trubian (UniMi), Panagiotis Papadimitriou, David Dietrich, Ahmed Abujoda (LUH), Michael McGrath, Vincenzo Riccobene (INTEL), Aurora Ramos (ATOS), Maurizio Arnaboldi, Pietro Paglierani (ITALTEL), José Bonnet (PTIN), Jordi Ferrer Riera, Josep Batallé Oronich, Eduard Escalona (i2CAT), Thomas Pliakas (CLDST)

Version 3.1 - final

Date 12th January, 2016

Distribution PUBLIC (PU)

Executive Summary

This deliverable presents the outcomes of Task 3.3 “Service mapping”, of the T-NOVA project. The task is focused on the research and design of algorithms for the optimal mapping of virtual network services in virtualized network infrastructures.

This work started by a review of use cases, requirements and architecture design proposed in Work Package (WP) 2, and by the analysis of the scientific and technological state of the art in service mapping algorithms. Furthermore, the input and output interfaces for the algorithm have been specified in detail, in terms of needed input data and of required output data for a service instantiation. Based on the above points, three main service mapping algorithms together with an algorithm for service scheduling have been proposed. Two service mapping algorithms are based on integer linear programming while the third is based on a stochastic control methodology. The mathematical formulation and the properties of the algorithms are presented and discussed in detail. Simulations results in emulated environments are proposed and discussed to highlight the characteristics on the algorithms. Prior to the description of the algorithms, a common mathematical framework for modelling the service mapping problem is proposed, based on the ETSI reference documents on virtualization architecture. The modelling framework is based on graph theory and provides a way for consistently modelling:

1. The requirements of the network services to be mapped.
2. The network infrastructure’s topology and its current occupancy level so that feasible and optimal mappings are computed by the algorithm.

The developed algorithms allow to consider multiple mapping objectives, including:

1. Maximisation of network service requests’ acceptance.
2. Minimization of mapping costs (i.e. the costs for employing the different network resources).
3. Balancing of network/datacenter load distribution.

Aside of the design of the mapping algorithms, another fundamental output of this task is given by the design of a microservice-based service mapping module, aimed to host the service mapping algorithm, and of its integration inside TeNOR, the T-NOVA orchestrator. Such activity has been carried out in close cooperation with the activities dedicated to the design of the infrastructure repository (Task 3.2) and of the network service descriptor (WP 6). As a matter of fact, as said before, network services’ requirements and network status are the two key inputs to the service mapping algorithms. The service mapping module offers the possibility to integrate and avail of any service mapping algorithm conforming to the input/output specifications, which are aligned to the three mapping algorithms developed. In this way, any investigated service mapping algorithm, or even a combination of them can be potentially integrated in TeNOR, and tested.

Finally, the integration of the service mapping module including one of the developed mapping algorithms has been achieved and preliminary tested, with details reported in this document.

Table of Contents

List of Figures

List of Tables

1. INTRODUCTION

1.1. Motivation, objectives and scope

In the T-NOVA system, the *service mapping module* is the component responsible for the decision of which resources of the virtualized network infrastructure must be leveraged to best serve an incoming request of a Network Service (NS) instantiation. As a matter of fact, the availability of multiple allocation solutions and, at the same time, the heterogeneity of resources types, in terms of requirements, costs, quality, etc., makes the introduction of an intelligent mapping logic a necessity, or at least a highly desirable feature for increasing the efficiency of the instantiation process. The first aim of this deliverable is therefore the presentation of a general and comprehensive discussion of the service mapping problem in virtualized network infrastructures. A mathematical modelling framework for the problem is here provided, and it is as well discussed how the service mapping module acts as a module of TeNOR – the T-NOVA orchestrator – and how it coordinates with the other modules of the T-NOVA architecture to perform its functions. Secondly, the work in Task 3.3 has aimed at researching different *service mapping algorithms*[1], designed to tackle the requirements and the peculiarities of the mapping problems addressed, and to explore the strengths offered by the different mathematical tools here used (mainly coming from the fields of integer linear programming and stochastic control theory). A third fundamental output of Task 3.3 is the design of the service mapping module, incorporated in a micro-services architecture, and its integration within the TeNOR orchestrator. This integration, in particular, has been achieved by means of a detailed documentation of the external and internal interfaces of the mapping module. The first are the interfaces between the mapping module and the other components of TeNOR, while the second one are the interfaces inside the service mapping module with the actual mapping mathematical algorithm utilized to solve the service mapping problem. This last step in particular provides a flexible way to make possible the integration and testing of different mapping strategies, thus providing a solid basis for future work improvements, continuation of future research activities, research cooperation of the topic in future initiatives, etc.

1.2. Structure of the Document

The remainder of the document is structured as follows:

- Section 2 presents the basis knowledge needed to frame the work on service mapping. In particular, this section reviews T-NOVA use cases, requirements and architecture design, seen in the light of service mapping requirements. Then, based on the ETSI relevant documents in the field, this section proposes a unifying mathematical modelling framework for service mapping, as a common base and input to all the methodologies devised. Finally, a summary of the analysis of the state of the art in service mapping is reported.
- Section 3 presents the detailed discussion of three service mapping algorithms proposed for T-NOVA, along with a scheduling algorithm.

[1] The reader should notice the distinction between the *service mapping algorithms*, which are mathematical algorithms designed to solve the service mapping problem, as explained in Section 3, and the *service mapping module*, which is the module of the T-NOVA architecture hosting the implementation of a service mapping algorithm (as explained in Section 4).

- Section 4 deals with the design of the service mapping module and its integration in TeNOR, the T-NOVA orchestrator. This section details the integration logic and the technologies involved.
- Section 5 discusses simulation results for the different mapping logics devised. The aim of the simulations is to highlight the properties of each algorithm and to test performances and scalability in extended scenarios.
- In Section 6 the conclusions for the work of the task are given.
- The lists of references, acronyms and mathematical symbols are reported after Section 6.
- The Annexes present a brief documentation of the developed service mapping Application Programming Interface (API) and the specification of the data interfaces with the main other modules of the T-NOVA orchestrator.

2. PROBLEM DEFINITION

This chapter provides the basic information on the service mapping problem as tackled in the T-NOVA project, to properly lay the foundation for the discussion of the different algorithms conceived in the project, and to correctly frame the discussion on service mapping module design and integration into the T-NOVA orchestrator, TeNOR.

In the next section, the Use Cases (UCs) and the requirements involving service mapping are recalled from deliverable D2.1 "System Use Cases and Requirements" [1], to define the boundaries of the service mapping problem as addressed in T-NOVA.

2.1. Use Cases and Requirements

In the following we briefly summarize the T-NOVA use cases, described and discussed in the deliverable D2.1 [1], that are related to the service mapping problem. The requirements for the service mapping module are identified as well.

2.1.1. Use cases

Service mapping is a core technology in TeNOR, enabling use cases that have a central role in the addressed business framework, as detailed in the next figure.

Figure 1 T-NOVA Overall Use case diagram ([1])

Use cases analyzed in D2.1 [1] and related to the service mapping are the following:

1. *UC2.1. Map and deploy service* (see D2.1 [1], Section 5.2.2.6). The VNFs are mapped into appropriate resources and then provisioned on the Network Functions Virtualization (NFV) infrastructure. The use case may be executed in two different manners: either upon a new service request by the customer (UC2), or as a result of a service reconfiguration or rescaling (UC3).

2. *UC3. Reconfigure/Rescale NFV services* (see D2.1 [1], Section 5.2.2.7). This use case is focused on the adaptation of the resources allocated to a specific service, optimizing resource usage, and/or modification of configuration parameters. Two variants are considered, as described below.

    a) *UC3.1 scale-out/ scale-in VNF Service*
        - Scale-out of the NFV service results in additional VNF instances being added to an existing instance. The new VNF instances require the instantiation of new Virtual Machines (VMs) with compute, network, and/or storage capacity to host the new VNFs.
        - Scale-in removes VNF instances (and their host VMs) that are no longer required. This action releases compute, network and storage resources.

    b) *UC3.3 Reconfigure VNF Service*: the configuration/parameters of the service are adjusted.

3. *UC6. Terminate NFV services* (see D2.1 [1], Section 5.2.2.11). This use case defines the procedures related to the termination of a provisioned NFV service, either by the customer or the Service Provider (SP), and removal of a VNF from the catalogue of available and advertised services.

Service mapping is thus envisaged in TeNOR as a key building block in order to support optimized deploying of network services and reconfiguration of the same in case breaches of the service levels are detected by the monitoring functionalities.

2.1.2. Requirements

The analysis of the service mapping problem and the design of the proposed solutions reported in this deliverable have moved from the related key functional requirements identified in deliverable D2.1 [1]. Such requirements are summarized in the following paragraph, taken by the mentioned deliverable:

> **NFV service mapping.** The T-NOVA system should be able to *map NFV service requests received from customers to the network, such that all NFV service requirements are met*. Specifically, *this requires the mapping of virtual network topology to the substrate network, while satisfying any bandwidth and/or delay requirements, as well as the assignment of NFVs to substrate nodes* that have sufficient computing and storage resources for packet processing, forwarding and/or caching. *In turn, NFV service mapping entails requirements such as the substrate network topology, processing, storage and network resource availability across the network, as well as the computational requirements of the NFVs that should be deployed*. *NFV service mapping should be optimised based on one or multiple objectives*, such as the minimisation of the mapping cost, the maximisation of the provider's revenue or the maximisation of NFV service request acceptance rate.

The text in Italic marks particularly relevant aspects to be taken into account when designing and developing a service mapping algorithm. In particular:

1. A flexible and consistent mathematical modelling of the service mapping problem, to be able to correctly capture the process of mapping the network service to the network infrastructure resources, taking into account user requirements and service/infrastructure constraints. This aspect is addressed in details in Section 2 of this report.
2. The need of feeding the mapping module with information on both the current status of the network and the detailed requirements of the service to be mapped. This aspect is addressed in Section 4, which details the interfaces of the service mapping module with the infrastructure repository and the service catalogue, designed and implemented with the purpose of retrieving the above data.
3. The fact that the developed mapping strategies have to take into account multiple objectives stemming from service users, providers, infrastructure operators, etc. (e.g. maximization of service performances, minimization of mapping costs, maximization of mapping acceptance rates, etc.). This aspect is addressed in details in Section 3, where the proposed T-NOVA mapping strategies are reported and explained, and in Section 5, where simulation results are discussed, highlighting how the multi-objectives are accounted for and achieved.

The T-NOVA atomic requirements addressed by the service mapping module are reported in the following table, for the sake of completeness. For each relevant requirement, its relation to the service mapping task is explained in the last column. The table is taken and adapted from the annex of deliverable D2.1 [1].

Table 1 T-NOVA requirements relevant to service mapping.

Req. id	Use Case	Req. Name	Requirement Description	Justification of Requirement	Relation to Service mapping
T_NOVA_04	UC1, UC2, UC3	NS Composition	The T-NOVA system SHALL be able to compose a NS from atomic VNF instances available at the NF Store and define the logical topology among the several components.	The creation of a NS from the combination of atomic/simple VNF is important in order to simplify the process provision of NS to the customers and avoid complex path calculations	The information regarding the NS composition process outcome (i.e. NS topology and requirements) is acquired by the service mapping module each time a NS mapping request is done. That is done in order to ensure the service mapping solution meets all the NS provisioning requirements.
T_NOVA_08	UC1.1, UC2	Resource Mapping	The T-NOVA system SHALL be able to map an incoming customer service selection (service + Service Level Agreement - SLA) to specific computational, storage, network infrastructure resources based on specific optimisation criteria or constraints.	Infrastructure resources used to host a specific VNF service and SLA restrictions need to be selected from a pool of infrastructure resources; this selection must comply with applicable optimization criteria or constraints	This is the key requirement addressed by service mapping.

T_NOVA_20	UC2, UC3, UC4	Resource monitoring	The T-NOVA system SHALL be able to monitor and collect information about consumption and availability of resources (computational, storage, network) on a real time basis, including the resources consumed by each specific VNF instance.	Monitoring is essential to ensure that the deployment of VNF's onto hosting infrastructure is performed adequately. Monitoring provides essential metrics required by operations such as rescaling, billing, etc.	Monitoring information is an input to service mapping module, which provides mapping solutions aligned and optimised according to resources' availability.
T_NOVA_21	UC2	VNF creation	The T-NOVA system SHALL be able to automate the instantiation of VNFs on the infrastructure based on customer requests and constraints.	Automation of VNF lifecycle is an essential characteristic of the T-NOVA system	The systems fulfilling these requirements are the actuators of the service mapping decisions.
T_NOVA_26	UC2	Topology of VNF	The T-NOVA system SHALL define the logical topology between the several VNF components.	Connectivity between VNF components must be automated	The logical topology between the several VNF components of a NS is an input to the service mapping algorithm.
T_NOVA_30	UC3, UC4, UC4.1	SLA monitoring	The T-NOVA system SHALL be able to compare service metrics with SLA requirements and indicate SLA status (conformance/breach). When the T-NOVA system determines that an SLA is in breach it SHALL initiate the applicable action, e.g. rescaling.	SLA management and monitoring is considered essential for the commercial applicability of the T-NOVA system. The T-NOVA system must determine when an SLA is in breach and trigger corrective actions.	Service mapping is one of the possible tools that TeNOR can use to react to SLA breaches.

In addition to the above requirements, also the atomic requirements on scaling/migration (T_NOVA_36-T_NOVA_45 in D2.1 [1]) are related to service mapping, in the sense that service mapping is one of the tools that TeNOR could use to deal with scaling/migration.

2.2. Reference Scenario and Architecture

This section presents the service mapping module in the context of the T-NOVA system architecture, explaining which are the relevant modules of the T-NOVA system providing inputs to the service mapping module, and which are the ones responsible for enforcing its decisions.

TeNOR service mapping module could be called in two different occasions:

- When a new instance of a network service is requested: this kind of request comes from the marketplace, and it happens when a customer wants to buy a network service.
- When the SLA enforcement module predicts a SLA breach and notifies the NS management module, which requests a new mapping for the resources that the NS instance is consuming.

For the service mapping module, these two scenarios involve the same kind of workflow, which is the focus of this deliverable.

2.2.1. Architecture and Flows

The place of the service mapping module within the TeNOR architecture is shown in Figure 2 below.

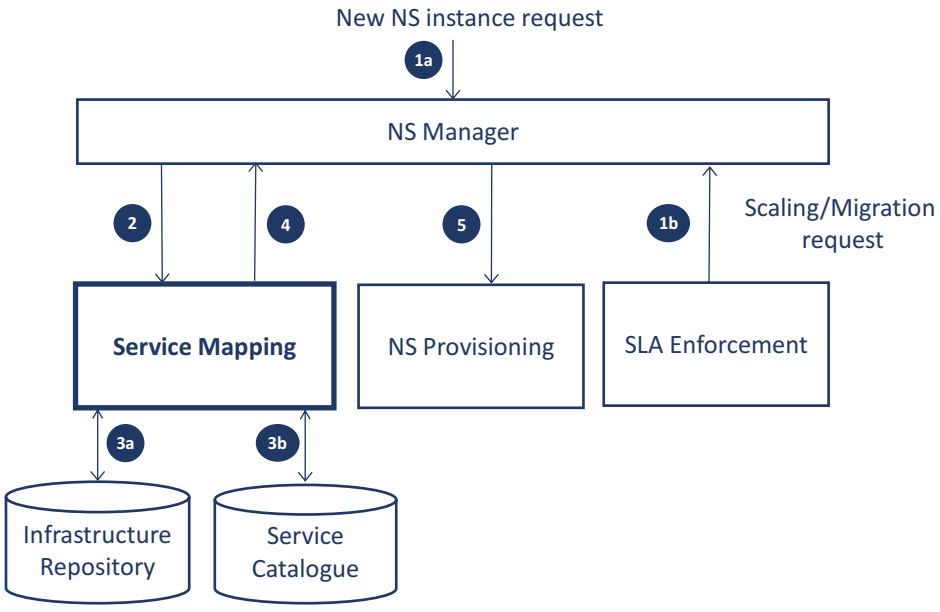

Figure 2 The service mapping module in the TeNOR architecture

The figure shows the events and flows involved in the calls to the service mapping module:

1. As described above, there are two kinds of reasons a call to the service mapping can be made: because a new NS instance (1a) or because a scaling/migration of an existing instance in order to keep the agreed SLA (1b) is needed.
2. The request for a new mapping is passed to the service mapping module.

3. The service mapping module grabs from the infrastructure repository the most up-to-date data about the infrastructure (step 3a). In addition, the service mapping module grabs from the service catalogue (T-NOVA Network Service Descriptor (NSD)) the relevant information about the service to be mapped (step 3b), such as thresholds for the technical SLA metrics, and/or node and link requirements, as explained in the next Section 2.3.
4. After the mapping optimization, the service mapping module returns the list of Points of Presence (PoPs) where the resources can be located.
5. Finally, the NS Manager passes these locations to the NS Provisioning, in order to implement the decisions taken by the service mapping module.

The objective of this deliverable is to specify in detail the workflow that has been outlined above.

The two key modules of the T-NOVA system supporting the service mapping module are the **infrastructure repository** and the **service catalogue**; they are briefly introduced in the following paragraphs, while full details are given in Section 4.2.

The infrastructure repository subsystem provides the service mapping module with the infrastructure related information, gathered from the Virtualized Infrastructure Manager (VIM) and Network Function Virtualized Infrastructure (NFVI) components, as shown in Figure 3. Also, a resource discovery mechanism allows the subsystem to augment the information provided by cloud and SDN environments.

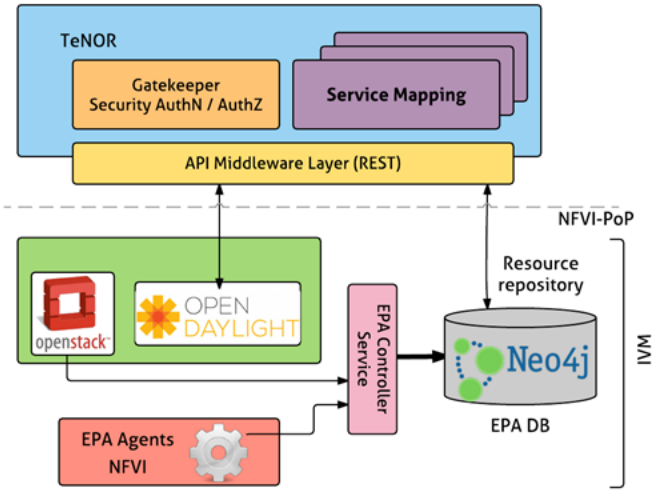

Figure 3 Resource repository high level architecture.

The infrastructure repository component that interfaces the service mapping module is the API Middleware Layer. It is a layer that provides a common set of REST API calls that can be used by the TeNOR modules to request and retrieve information from the infrastructure repository. It is through the API Middleware layer that the service mapping module can retrieve the infrastructure repository information and build a consistent representation of current status of the infrastructure, either in term of its topology or of the current availability of resources. That is done each time a NS mapping request is made, in such a way that the solution computed by the mapping algorithm is aligned with the current status of the infrastructure, in terms of resources' availability.

The service catalogue, on the other hand, provides the service mapping module with a consistent representation of the network services requirements and with the main

characteristics. In fact, a feasible solution of the service mapping problem must respect network service node and link requirements, along with the expected service performance defined in the SLA, as agreed between the service provider and the customer in the marketplace and included in the network service descriptor. ESTI's NSD format (see e.g. [2]) has been extended by the T-NOVA project to include additional fields to define the thresholds for service performance metrics, based on the expected performance of the different deployment flavours of the VNFs that are part of the service. For each VNF deployment flavour the VNF developer indicates the Virtualization Deployment Unit (VDU) requirements which are included in the VNFD. Therefore, by means of the NSD and of the VNFD the service mapping algorithm gets the resource requirements needed to deploy the service in order to meet the agreed SLA. Additional details on the descriptor parameters relevant to the service mapping problem are discussed in Section 4.2.2.

The sequence diagram below summarises the interaction of the service mapping module with the service manager, the infrastructure repository and the service catalogue.

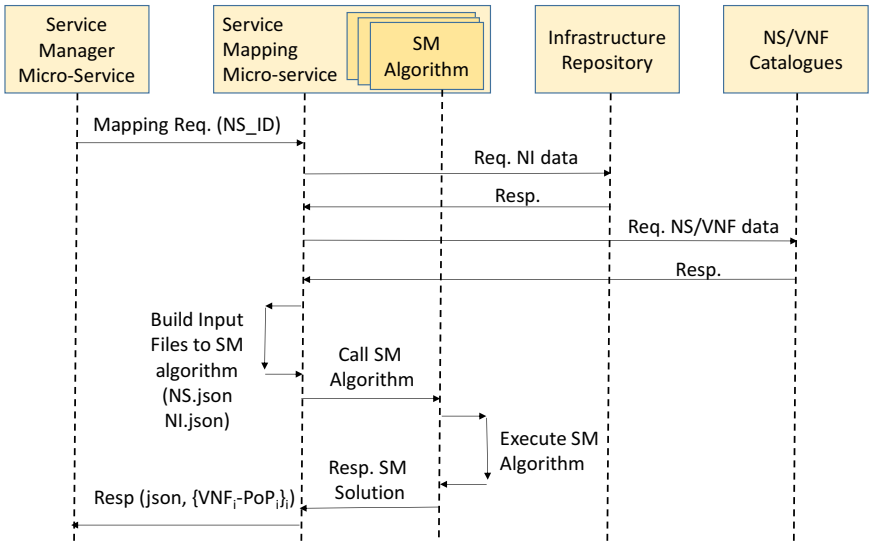

Figure 4 Interaction of the service mapping module with the infrastructure repository and the network service catalogue

From that picture it can be already seen that the mapping module is divided into two main sub-components:

1. A module responsible for providing interfacing and data adapting functionalities; it retrieves and organises (in the two JSON files specified in the figure) the data from the mapping call, the infrastructure repository and the service catalogue.
2. The actual module responsible for the mapping problem solving, which receives, from the two JSON files, all the inputs needed to build the mapping problem itself.

More details on the integration of the mapping module into the T-NOVA architecture can be found in Section 4.

The next section presents a modelling framework for the service mapping problem tackled in T-NOVA. The sections are based on the preliminary outputs of the task as reported in the interim deliverable D3.01 [3].

2.3. Problem Modelling

The Service mapping (SM) problem addressed in T-NOVA focuses on the optimal assignment of Network Service (NS) chains to servers hosted in interconnected datacenters (DCs) operated by the same network service provider.

The optimality concept can be defined toward different objectives: economical profit, Quality of Service (QoS), energy-efficiency and others.

The SM is an online problem. That is, the requests for NSs are not known in advance. Instead, requests arrive to the system dynamically and, when accomplished, they can stay in the network for an arbitrary amount of time. Algorithms for the SM problem have to handle service requests as they arrive.

According to ETSI's NFV Architectural Framework [4], a NS is represented by a forwarding graph in which each vertex is a Virtual Network Function (VNF). Hence, in T-NOVA, a NS is defined as a directed graph $G(NS) = (V, A)$ in which each vertex, say h, in the set V represents a VNF, and each arc, say (h, k), in A represents a link connecting two VNFs, required for the correct implementation of the service (e.g. a chain in a web server tier composed by firewall, NAT and load balancer).

The Network Infrastructure (NI) on which we want to run the NS can be described as a directed graph $G(NI) = (V^I, A^I)$ in which each vertex, say *p*, in the V^I set represents a DC, and each arc, say (p, q), in A^I represents the network connection established by the network provider among the DCs.

Hence, the first problem arises when a new NS instance request arrives to the orchestrator and the SM is asked to assign each VNF in the required service to a DC within the available network infrastructure (note that it is possible that all the involved VNFs are eventually assigned to the same DC). More formally, this "first level problem" can be stated as follows.

First level problem: Given a NS and a NI, solving the first level SM problem requires to assign each VNF of the service, to a DC in the network (i.e. each vertex in *V* to a vertex in V^I) and each arc (*h*, *k*) in *A*, to an oriented path in G(NI) from the DC to which the vertex *h* has been assigned, to the DC to which the vertex *k* has been assigned.

Figure 5(a) reports a NS composed by two VNFs, a NI composed by four interconnected DCs and their corresponding graphs.

Figure 5(b) reports a possible solution of the first level problem involving the graphs of Figure 5(a). VNF_1 has been assigned to DC_1, VNF_2 has been assigned to DC_4 and the arc connecting VNF_1 and VNF_2 has been assigned to the blue path from DC_1 to DC_4, through DC_3.

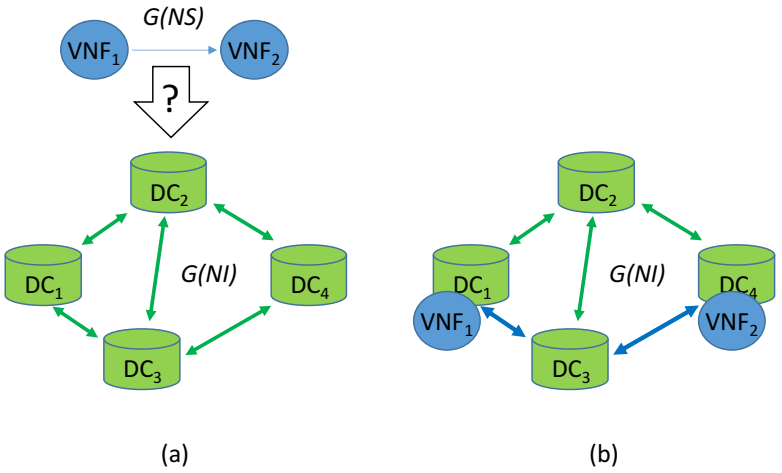

Figure 5 Example of a first level SM problem (a) and its solution (b)

Moreover, each VNF can have a complex structure, i.e., it can be made of elementary interconnected components, each one executed on a VM. At the same time, each DC is composed by hundreds (or thousands) of interconnected servers.

Hence, once a VNF has been assigned to a DC, a second problem (referred to as a "second level problem") arises with the request to instantiate each VM composing the VNF on a server hosted in the DC.

More formally, each VNF can be described as a directed graph $G(VNF) = (V^F, A^F)$ in which each vertex, say *i*, in the V^F set represents a Virtual Network Function Component (VNFc) [5], and each arc, say (*i*, *j*), in A^F represents a link between the VNF components.

In turn, each DC can be described as a directed graph $G(DC) = (V^D, A^D)$ in which each vertex in the V^D set represents a hardware device, either a server or a network switch, and each arc in A^D represents the network connection established by the DC owner between the hardware devices.

Figure 6 displays, on the left side, a VNF composed by four interconnected components, and, on the right side, the internal structure of a DC model with its interconnected devices.

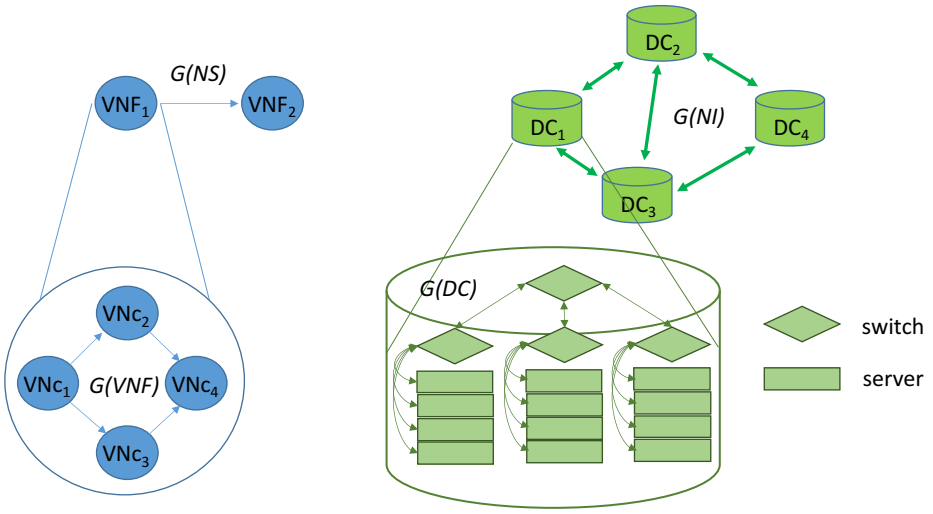

Figure 6 Example of a VNF composed by four VNFcs (on the left) and the internal structure of a DC model (on the right)

Second level problem: Given a VNF and a DC, solving the second level SM problem requires to assign each VNFc in the VNF to a server in the DC (i.e. each vertex in V^F to a vertex representing a server in V^D) and each arc (i,j), in A^F, to an oriented path in G(DC) from the hardware device hosting VNFc *i* to the hardware device hosting VNFc *j*.

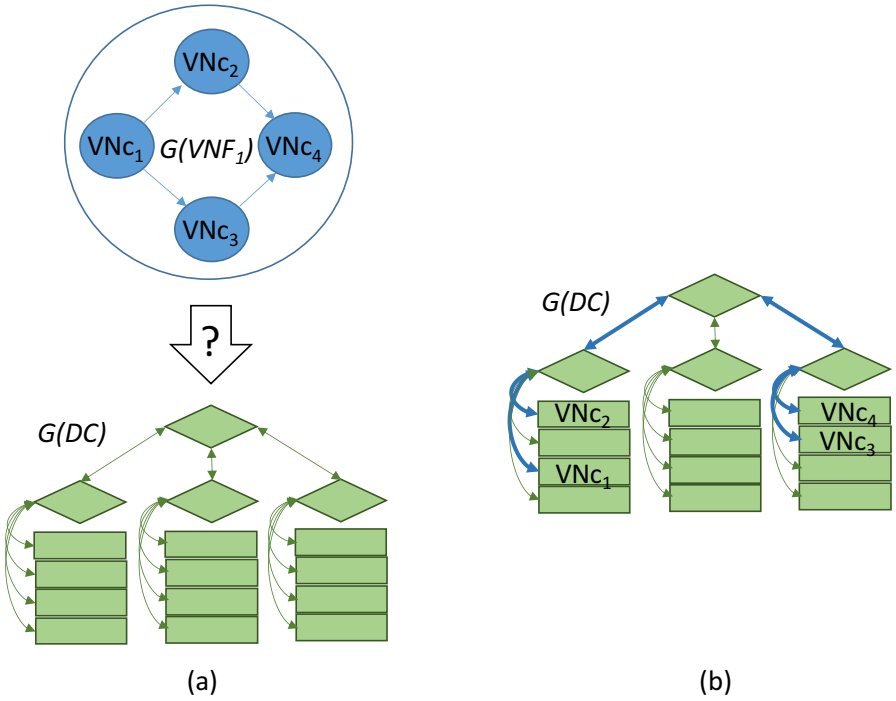

Figure 7 Example of second level SM problem (a) and its solution (b)

Figure 7(a) shows an instance of the second level problem, in which the VNF_1 components must be assigned to the DC_1 servers. Figure 7(b) shows a solution of the second level problem, where each component has been assigned to a (suitable) server and the links connecting the components have been mapped to the blue paths involving switches and servers.

The second level problem is not part of the service mapping module and it is solved by calling the appropriate OpenStack functions (see discussion in next Section 2.4.2).

The candidate hardware for a mapping, i.e. servers and links within each DC and links between couple of DCs, have to be able to support the performance requirements of the virtual components. This means that in each *G*(*NS*) directed graph and in each *G*(*NI*) directed graph, resource pools are associated to each vertex and to each arc. These resources must be available on servers that will host the involved virtual machines and in the links used to guarantee the connectivity required by the network service (i.e. network links with specific capacities and QoS).

In particular, a feasible solution of the service mapping problem must respect the following three requirements.

Node Requirements. A set of node resource types, say NT, is associated to the nodes of the $G(NS)$ and $G(NI)$ graphs. Each member of the NT set represents a particular resource (e.g. CPU power need, number of cores, number of hardware and software accelerators, number of GPUs, etc.), which can be required by a VNF, since it could be required by some of its *VNF*c. In turn, each member of the NT set can be present in a DC, since it could be present in some $R_h{}^t$, is associated to each VNF node h, with $t \in NT$. It represents the amount of aggregate resource of type t *required* by the VNF h. A numeric value, say RA_u^t, is associated to each *NI* NT. It represents the amount of aggregate resource of type t available in the DC u. The

aggregate values in each node are computed summing up the values of the corresponding resource quantities, required or available, in the single components, VNFc or servers, in that node.

For each DC node u and resource type t, the sum of the aggregated resource needs of all VNFs mapped to it cannot exceed the aggregate available resource RA_u^t.

Link Requirements. A set of link resource types, say LT, is associated to the links of the $G(NS)$ and $G(NI)$ graphs. Each member of the set LT represents a particular resource (e.g. bandwidth) which can be required by an arc (h,k) in $G(NS)$. In turn, each member of the LT set can be present in a link of the NI. A numerical value, say RR_{hk}^t, is associated to each arc LT. It represents the amount of resource of type t required by the arc (h,k). A numeric value, $A_{pq}{}^t$, is associated to each arc (p,q) in $G(NI)$, with $t \in LT$. It represents the amount of resource LT, the sum of the RR_{ij}^t values of NS arcs mapped to paths including (p,q) cannot exceed RA_{pq}^t.

Δ_π, is associated to each path π in P. An actual delay δ_{pq} is associated to each arc (p,q) in δ_{pq} of all the arcs $(p,q) \in A'$ belonging the paths used for connecting all the links belonging to Δ_π.

Since we are facing an online problem, the amount of physical resources available at any instance in each time is equal to the amount of the infrastructure hardware devices in the DCs minus the one allocated to the VMs currently running on their servers in response to satisfied NS requests. Only when a service is completed the resources (computational and bandwidth demands) allocated to it become newly available, and can be assigned, on the involved devices, to other incoming service requests.

2.4. Scientific and Technological State of the Art

After the previous general introduction to the service mapping problem, the proposed T-NOVA architecture and the modelling framework, this section presents a review of the scientific and technological state of the art in service mapping. Founding knowledge for framing the service mapping problem in the context of network function virtualization can be found in the relevant ETSI documents specifying terminology and concepts, use cases, reference architecture, etc. (see e.g. [2], [4], [5], [6]).

2.4.1. Scientific State of the Art

NFV has received attention by the research community since a few years. With regard to service mapping, the focus is on DC networks. Many works present cloud platform implementations that allow NFs to be arbitrarily integrated into virtual machines without considering the functionalities of a service chain. More in details, Oktopus [7], CloudMirror [8] and SecondNet [9] assign virtual clusters to DCs taking into account performance guarantees. Other works, such as STRATOS [10], discusses service composition considering different NFs. However, those works are mainly based on heuristic algorithms that seek to minimize inter-rack traffic within DC networks. In a similar vein, [11] discusses a heuristic second level mapping algorithm for assigning VMs to servers in a data centre. The paper addresses the case of NSs composed by multiple VNFs, pointing out that the typical case addressed in literature regards instead the mapping of single network functions. Interestingly the paper discusses how automated second level mapping procedures could be built on top of the existing cloud management systems, with particular regard to the OpenStack case (this aspect is briefly addressed in the next section, which details the OpenStack mechanisms for assigning VMs to the compute nodes).

Other early works, such as [8], are devoted to develop NS modelling techniques for solving the inefficiencies of previously adopted models, such as hose, VOC (Virtual Oversubscribed Cluster) and pipe models [8]. The technique, called TAG (Tenant Application Graph), allows to accurately capture bandwidth requirements for the VMs to be deployed, avoiding the overprovisioning inefficiencies that the other mentioned models do allow. The TAG is essentially a graph based model in which VMs, or tiers of VMs, are represented by nodes, and ingress and egress bandwidth requirements among VMs are modelled by directed edges, or self-loops, for modelling intra-tier bandwidth requirements. The NS models reported in this deliverable are an extension of the TAG model. In the same work, also a second level mapping strategy based on min-cut and knapsack algorithm is outlined, with supporting simulations showing the effectiveness of the algorithm in avoiding overprovisioning of resources. Basically, the logic of the algorithm is to maximise co-location of VMs linked by "heavy" edges, in terms of link bandwidth requirements.

The deployment of NFs over multiple DCs, i.e., the mapping of NF service chains over inter-DC networks finally enables the wide-area deployment of network services. Such service chain mapping is often compared to the Virtual Network (VN) embedding problem, which has been studied extensively. Fischer et al. [12] provide a survey on VN embedding accordingly. However, the rich variety of proposed VN embedding algorithms (w.r.t., resource mapping) cannot be directly applied to service chains due to the different NF types, policies incurred by the middlebox operators, and the changing traffic rates caused by some NFs. In any case, those works can inspire further approaches related to service chain embedding.

Other approaches exist also for the service chain mapping over multiple DCs and providers. Huang et al. [13] propose a distributed algorithm for network service placement assuming the

ability to deploy NFs in the data path. MIDAS [14] employs a heuristic algorithm for order-preserving NF assignment to middleboxes deployed along the data path.

The recent contribution by Riggio et al. [15] introduces an algorithm for mapping Service Function Chain (SFC) requests to a virtualised network infrastructure, focusing on the case of WLANs. The paper proposes a general NS (i.e. SFC) and virtualised network modelling framework, still at an early stage, relying on a graph formalism compliant with the ETSI NFV model. Both the sub-problems of VNF mapping to the underlying network nodes and of VNF virtual link mapping to network paths are here addressed. The proposed mapping strategy is a greedy one, in the sense that it sequentially visits the VNFs to be mapped, starting from the one with maximum connectivity degree. Each visited VNF is mapped to the network nodes which optimise a cost metric. The cost metric accounts for the residual capacity level of the network node, and for the capacity level of the paths connecting the network node with the network nodes hosting the previously mapped VNFs of the service. The virtual link to network path mapping is decided based on shortest path computation. Results are presented relying on performance metrics widely used in the relevant literature, such as SFC acceptance rates and nodes/links utilization levels.

Another interesting and advanced contribution can be found in [16], where authors propose a service mapping algorithm based on integer linear programming. The services and the infrastructure are modelled as undirected graphs, with specification of topology and node/link availabilities and requirements. One of the interesting features of that paper is the investigation of the so called "lookahead" property (previously introduced by the references mentioned in [16]), according to which more network services are embedded at the same time. Authors show that the solution efficiency increases as the number of services simultaneously mapped increases (i.e. network resources are better exploited and more service requests can be accommodated, even if it is shown that increasing the number of simultaneously mapped services increases the mapping time).

Authors of [17], [18][2] analyse instead the case in which a network service chain consisting of several network functions can be realized in multiple ways (the process of choosing the actual implementation "shape" for the requested service chain being referred to as "service decomposition"). The authors then propose a mapping algorithm which integrates a service decomposition phase, with the aim of optimally selecting, at runtime, the most suitable service configuration to implement. Both an ILP and a heuristic model for solving the problem are proposed.

Reference [19] introduces a context-free language to build a model for formalizing the network function chaining requests. Also, a mixed integer quadratic programming node and link mapping strategy is presented, with a Pareto evaluation of three different objective functions: (i) maximization of remaining data rate on underlying network links, (ii) minimization of the number of used nodes and (iii) minimization of latencies along the paths.

Also soft computing optimization techniques have been proposed for solving the service mapping problem. An interesting example in this sense is given in the early work [20], where a mapping approach relying on binary PSO (Particle Swarm Optimization) is proposed. Five different target functions for governing the mapping behaviour are here proposed (minimization of network nodes used, minimization of network link used, minimization of the cost of used links, and others not relevant to this discussion). Virtual link mapping is performed via shortest path computation, with the cost given by the free link resources. Other soft computing, approximate, evolutionary optimization techniques (such as simulated annealing,

[2] Works supported by the UNIFY project, 7th Framework Programme for Research and Technological Development, Grant Agreement No. 619609, www.fp7-unify.eu.

genetic algorithms, etc.) have been applied as well. The interested reader is referred to references in [20].

Among the recent works related to service mapping, reference [21][3], proposes a modular NFV architecture that permits policy-based management of VNFs, introducing as well an information model describing and abstracting network resources and network functions. This paper is interesting because it investigates the synergies between Network Functions Virtualization (NFV) architectures and Software-Defined Networks (SDN). To prove the concept, a simple virtual link mapping algorithm based on shortest path computation is proposed and the policy based framework for orchestrating VNFs is tested in a small-scale testbed. It is shown how the proposed rule-based framework allows to dynamically orchestrate the VNFs and the underlying infrastructure, in such a way as to ensure the SLAs defined for the different client tiers are satisfied.

Concluding, research on service mapping algorithm is continuously evolving, with many novel contributions being proposed. The interested reader may find additional discussion of consolidated state of the art in service mapping in [12], [22], [23].

Finally, the output of the current efforts of T-NOVA consortium in the field of service mapping research can be found in the following papers: [14], [24] (integer linear programming service mapping for multi domain service mapping with limited information disclosure); [25] (scheduling problem); [26] (early results on service mapping via reinforcement learning); [27] (description of the T-NOVA service mapping module and its integration with the T-NOVA orchestrator).

2.4.2. Technological State of the Art

This section presents details on the OpenStack filtering and weighting procedure, which is the technology aimed to decide on which hosts the VM should be instantiated. This is a technological solution to what has been called in Section 2.3 the second level service mapping problem, which is namely the problem of deciding which machines in the DC should host the mapped services. In Section 3.3.4, an algorithm proposed by T-NOVA for second level mapping is presented.

2.4.2.1. OpenStack Filtering and Weighting

When scheduling a VM in an OpenStack environment the “compute” service uses the nova-scheduler to determine where to instantiate the VM. When resourcing VM instances, the nova filter scheduler filters and weights each host in the list of acceptable hosts. The first step is the application of filters to determine which hosts are eligible for consideration when dispatching a resource, as shown in Figure 8. Filters are binary: either a host is accepted by the filter, or it is rejected.

The current available filters in OpenStack are as follows:

- AggregateCoreFilter
- AggregateDiskFilter
- AggregateImagePropertiesIsolation
- AggregateInstanceExtraSpecsFilter
- AggregateIoOpsFilter
- AggregateMultiTenancyIsolation
- GroupAffinityFilter
- GroupAntiAffinityFilter
- ImagePropertiesFilter
- IsolatedHostsFilter
- IoOpsFilter
- JsonFilter

[3] Work supported by the GN3plus project, 7th Framework Programme for Research and Technological Development, Grant Agreement No. 605243.

- AggregateNumInstancesFilter
- AggregateRamFilter
- AggregateTypeAffinityFilter
- AllHostsFilter
- AvailabilityZoneFilter
- ComputeCapabilitiesFilter
- ComputeFilter
- CoreFilter
- NUMATopologyFilter
- DifferentHostFilter
- DiskFilter
- MetricsFilter
- NumInstancesFilter
- PciPassthroughFilter
- RamFilter
- RetryFilter
- SameHostFilter
- ServerGroupAffinityFilter
- ServerGroupAntiAffinityFilter
- SimpleCIDRAffinityFilter
- TrustedFilter
- TypeAffinityFilter

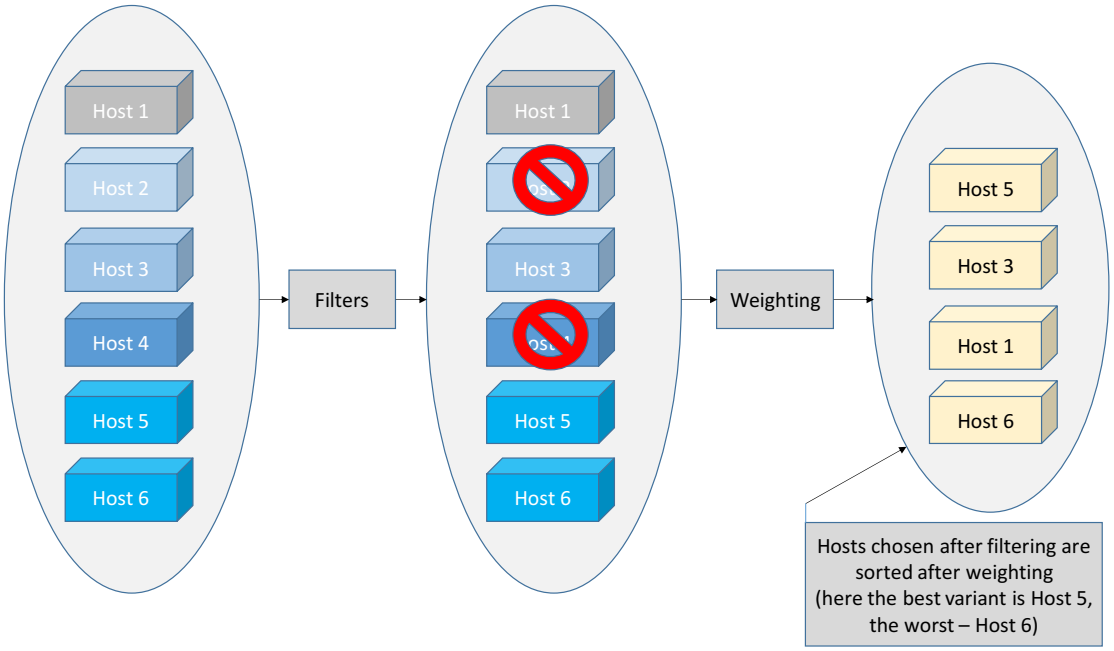

Figure 8 OpenStack Filter scheduler approach (picture elaborated from [28])

Hosts that are accepted by the filter are then processed by a weighing step to decide which hosts to use for that request, as shown in Figure 9. All weights are normalized before being summed up; the host with the largest weight is given the highest priority.

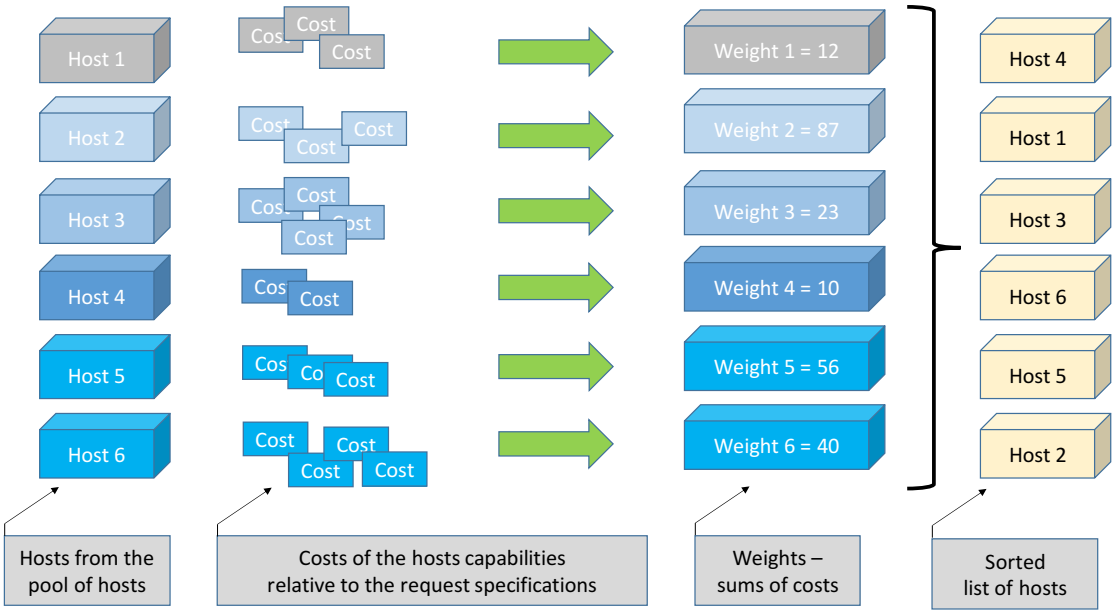

Figure 9 Weighting hosts (picture elaborated from [28])

3. T-NOVA SERVICE MAPPING ALGORITHMS

This section presents the mathematical details of the four service mapping approaches proposed in T-NOVA.

The first proposed approach is based on Integer Linear Programming (ILP) and aims to optimize NS to PoP mappings, having as objective the minimization of mapping costs and service delay in respect of infrastructure and services' constraints. This ILP approach takes decisions based on the current state of the network (as provided by the network monitoring) and on the NS requirements and specifications (as gathered from the NS catalogue).

The second proposed approach is a stochastic one (while the previous one is deterministic) and is based on the reinforcement learning theory. The approach aims at exploring the possibility of improving mapping decisions based on iterative learning of the environment dynamics (VNF types, arrival and termination rates, resources' capacity, etc.).

The third approach is also based on ILP, and investigates both a first level mapping strategy and a second level one. At first level, the approach investigates a different target function with respect to the one investigated by the first proposed ILP approach, aimed at balancing the load across the network. In addition, a second level approach based on Mixed Integer Programming (MIP) is proposed, where the objective is to maximise the NS co-location, while minimizing the traffic within the DC.

Finally, the fourth theoretic algorithm here proposed addresses the problem of NS scheduling, an issue complementary the of service mapping (details are in Section 3.4).

To ease the reading, a table reporting the list and a brief explanation of the mathematical symbols can be found in Section 9. Also, each section has a separate numbering of equations.

3.1. An Integer Linear Programming based Approach

This mapping strategy has been proposed by the University of Milan (Unimi). It is based on an ILP model for the first level problem. The aim of the first level problem is to identify a mapping of VNFs to DCs and links to paths which minimize the overall cost, while satisfying constraints on resource usage and delay (eventually coming from SLAs and defined in the NSD of the NS to be instantiated). The objective function of the optimization model is a weighted sum of three components: (i) the cost of assigning VNFs to DCs, (ii) the overall delay and (iii) the overall link resource usage, as derived by assigning the links among VNFs to path among DCs. Since we are facing an online problem, this objective implicitly models the true overall target function, i.e. the maximization of the number of accepted NS requests.

Given a service request and a network infrastructure, for each VNF, say h, composing the service, and for each DC, say p, composing the network infrastructure, we need to define a cost, say c_p^h, which can model the *cost* of assigning h to p in term of the maximization of accepted requests. These quantities can be estimated gathering data from the quality of the solution obtained when solving the second level problem through the OpenStack API calls. have been tuned according to the results of an experimental campaign (see Section 5.1).

In this ILP formulation we use the binary variables y_p^h to express the assignment of VNF h to the DC p, whereas the binary variables x_{pq}^{hk} indicate whether the link (h,k) in graph $G(NS) = (V,A)$ has been mapped onto a path among DCs in graph $G(NI) = (V^I, A^I)$ which uses the link (p,q).

The ILP approach for first level mapping can be therefore formalized as follows.

Problem (*ILP based Service mapping*)

Minimize

$$\alpha \sum_{h\in V} \sum_{p\in V^I} c_p^h y_p^h + \beta \sum_{\pi\in P} \sum_{(h,k)\in\pi} \sum_{(p,q)\in A^I} \delta_{pq} x_{pq}^{hk} + \gamma \sum_{t\in LT} \sum_{(h,k)\in A} RR_{hk}^t x_{pq}^{hk} \quad (1)$$

Subject to

$$\sum_{p\in V^I} y_p^h = 1 \qquad \forall h \in V \quad (2)$$

$$\sum_{p\in V^I} x_{pq}^{hk} - \sum_{p\in V^I} x_{qp}^{hk} = y_p^h - y_p^k \qquad \forall (h,k) \in A, \forall p \in V^I \quad (3)$$

$$\sum_{(h,k)\in\pi} \sum_{(p,q)\in A^I} \delta_{pq} x_{pq}^{hk} \leq \Delta_\pi \qquad \forall \pi \in P \quad (4)$$

$$\sum_{(h,k)\in A} RR_{hk}^t x_{pq}^{hk} \leq RA_{pq}^t \qquad \forall (p,q) \in A^I, \forall t \in LT \quad (5)$$

$$\sum_{h\in V} RR_h^t y_p^h \leq RA_p^t \qquad \forall p \in N^I, \forall t \in NT \quad (6)$$

$$y_p^h \in \{0,1\} \qquad \forall h \in V, \forall p \in V^I \quad (7)$$

$$x_{pq}^{hk} \in \{0,1\} \qquad \forall (h,k) \in A, \forall (p,q) \in A^I \quad (8)$$

The objective function (1) is a weighted sum of three components: the cost of assigning VNFs to DCs, the overall delay and the overall link resources usage.

Constraints (2) ensure that each VNF h is mapped exactly to one DC. Conditions (3) ensure that for a given pair of VNFs h and k assigned to DCs p and q, respectively, there is a path in the network infrastructure graph $G(NI)$ connecting p to q to which the edge (h,k) has been mapped. Constraints (4) impose the satisfiability of a SLA based on the delay thresholds for in the P set. Constraints (5) impose the link resource limit of the inter DC connections for each link resource type t in the resource type set LT. Constraints (6) impose the node resource limit of the DC for each node resource type t in the resource type set NT. The conditions (7) and (8) express the binary domain constraints for the variables used.

The above ILP model is solved by invoking the open source GLPK ILP solver. This choice has been tested against the choice of the commercial solver CPLEX (see Section 5.1).

3.2. Reinforcement Learning Based Approach

This section presents a service mapping strategy based on Reinforcement Learning (RL) [29]. We first focus on a strategy to map single VNFs and then provide a possible extension of the method to the general case of NS mapping.

This mapping strategy has been proposed by the Consortium for Research in Automation and Telecommunication (CRAT). Early results of the work can be found in [26].

3.2.1. VNF Mapping via Reinforcement Learning

This section describes a reinforcement learning strategy aimed at mapping single VNFs. Early work on the topic can be found in the paper [26], to which the interested reader is redirected.

3.2.1.1. Problem Modelling based on Markov Decision Process

Reinforcement learning methodology applies to control problems which can be modelled as a Markov Decision Process (MDP) [30].

A MDP is a discrete-time stochastic control process defined by the quadruple $\{S, Act, T, r\}$ where:

- S is a discrete and finite state space set.

- Act is a discrete and finite action space.

- T is the transition probability matrix, which describes the system dynamics.

- $r: S \times S \times Act \rightarrow R_{\geq 0}$ is the reward function, that describes the reward obtained in the transition from s to s' when action $a \in Act$ is taken.

The main MDP definitions rely on the Markovian (or memory-less) property and on the stationary distribution of the stochastic process. Under these assumptions, the transition probabilities are stationary and depend on the current state-action pair: the generic element of the matrix T, denoted with $t(s, a, s')$, describes the probability that the system trajectory transits from state s to state s' when action $a \in Act$ is taken. A policy Π is a mapping of each state s to an action a. The state value function $V_{\Pi}(s)$ is defined as the expected reward when the system is in state s and the system evolves under policy Π. Similarly, the state-action value function $Q_{\Pi}(s, a)$ is defined as the expected reward when the system is in state s, action a is chosen and the system evolves under policy Π.

In the following, the actual SM problem is modelled, in order to derive information on the associated MDP $\{S, Act, T, r\}$. Let us denote with T the time horizon over which the problem is defined. Let $|\mathrm{A^D}|$ denote the number of PoPs in the network infrastructure and $PoP = \{1,2,3,\ldots,|\mathrm{A^D}|\}$the PoPs' IDs.

Recall that RA denotes the vector of resources available at the different PoPs. The generic component of RA, called RA_p, denotes the amount of the different resources available at the PoP p. In turn, the component t of RA_p, named RA_p^t, denotes the amount of resources of type t (e.g. memory, computation, storage, etc.) available in the PoP p (we consider a univocally ordered set of resource types).

As a matter of fact, resources are differentiated based on their nature: a cloud provider can offer, for example, both computational and storage resource, and, regarding storage, either on ssd or on hdd disks; some machines may mount a specific network card, some other machines may have a higher uptime, and so on.

On the other hand, each VNF is characterised by the requirement vector RR_h^t, stating the amount of resources of type t required by the VNF h to be mapped.

State Space Definition

In the early work [26] the state space has been first defined as

$$S = \{s(t) = s_{v,p}(t), t \in \mathrm{T}, v \in VNF, p \in PoP\} \quad (1)$$

where $s_{v,p}(t)$denotes the number of VNF of type v deployed at PoP p. The state space thus gives a view of the occupancy level of the PoP at the different times. A formulation considering such definition of the state space presents a scalability problem due to the state space explosion observed when the number of VNF types and PoPs increases. Thus, the following aggregated formulation for the state space is considered in the following formula

$$S = \{s \in \{0,1\}_{[0,|V^I|]}\} \quad (2)$$

in which the length of the generic state vector is equal to the number of PoPs in the network infrastructure ($|V^I|$), and the generic i-th component of the state vector is equal to one if the aggregated occupancy level of the i-th PoP is greater than a predefined threshold. By providing an aggregated view of the PoPs occupancy level, this state space formulation improves scalability with respect to the previous one. The current state of the systems is evaluated by the service mapping module through the API made available by the infrastructure repository, as detailed in Section 4.2.1.

Action Set

The action set regards the decision about where VNF is deployed, and is defined as follows

$$A = \{1,2, \dots, |A^D|\} \quad (3)$$

where $a = i$ means that the VNF in question is deployed on the i-th PoP.

Transition Matrix

We assume that the requests and the terminations of services are characterized as follows: for each service of type k, the new NS requests are distributed according to a Poisson distribution of mean arrival frequency $\lambda_k$and terminations according to a Poisson distribution of mean arrival frequency μ_k.

In case the state space definition (1) is adopted, a state transition due to a new service instantiation can be modelled as $s(t+1) = s(t) + \delta_{i,k}$ where $\delta_{i,j}$ is a vector all zeroes but the i*j-th component, which is equal to one, meaning that a new i-th VNF has been allocated in the j-th PoP. VNF termination can be modeled similarly.

The transition probabilities among states can be modelled through a transition matrix $T(s, a, s')$, specifying the probability that performing action a in state s will lead to the new state s'. Transition probabilities depend on the arrival and departure rates. In case the state space definition in (2) is chosen, it is not possible to easily derive a state transition matrix, since an aggregated state description is adopted, so that there is no longer a granular view on the PoP occupancy level. However, the reinforcement learning solution approach proposed in the following does not require a complete knowledge of the MDP, and of the transition matrix in particular (we adopt a model-free approach).

Reward Function

The reward matrix is chosen to reflect the service mapping goals. A possible choice in particular is the following

$$r(s,a,s') = \begin{cases} r, & r > 0 \; in \; case \; of \; sussessfull \; mapping \\ 0 & otherwise \end{cases} \quad (4)$$

where different choices for the reward factor *r* are possible:

- A constant reward, leading to maximization of requests' acceptance.
- A reward function depending on the VNF to be mapped and the PoP chosen. This choice can be adopted if minimisation of mapping costs is sought (when the costs for mapping to the different PoPs are known), or as well maximization of mapping revenue (when different VNFs yields to different revenues).

The following section briefly describes the solution strategy devised to derive an optimal mapping strategy, that is, a strategy which maximises the revenues in the long term.

3.2.1.2. MDP Solution Strategy

The objective of this section is to derive a feasible solution strategy to work out the optimal policy for the MDP defined in the section above. An optimal policy is the one optimizing the expected reward in the long run.

Several strategies are available in literature to find the optimal policy (a review of the main ones can be found in [29]). It is known that, in case full information on the MDP are available, the optimal policy can be computed by solving the Bellman equation of dynamic programming [29]. In our case, however, it is not realistic to assume perfect model on the MDP, and in particular on the transition matrix (since there is not perfect knowledge of the NS request and termination patterns and the state space has an aggregated structure). For this reason, RL approaches for the optimal policy calculation are investigated next. RL are *model free* methods, which do not assume complete knowledge of the environment (of transition matrix T in particular). Different RL methods exist, of which Q-learning will be investigated next. Both methods aim at estimating the previously introduced state-action value function $Q_{\Pi}(s,a)$, defined as the expected value of the cumulative reward obtained when taking action a from state s and then following policy Π.

$$Q_{\Pi}(s,a) = E_{\pi}\{\textstyle\sum_{\tau} r_{\tau} \,|s_{\tau} = s, a_{\tau} = a\} \quad (5)$$

where $r_{\tau}, s_{\tau} \; and \; a_{\tau}$ are the reward, state and action at time τ.

Q-Learning

Q-Learning [29] iteratively estimates the best state-action value function through the following update rule, executed each time an action is taken and the effect (i.e. reward) of the action is observed.

$$Q(s_{\tau}, a_{\tau}) \leftarrow Q(s_{\tau}, a_{\tau}) + \alpha_{\tau}[r_{\tau+1} + \lambda \, max_a Q(s_{\tau+1}, a) - Q(s_{\tau}, a_{\tau})] \quad (6)$$

Here τ is the time, α_{τ} is the so called learning rate and $\lambda \in [0,1]$ is a discount factor weighting current rewards versus future ones.

Based on the current estimate of $\mathrm{Q}(s_\tau, a_\tau)$, the mapping action to be taken is decided as the one which maximizes the current state-action value function.

$$\pi_\tau(s) = argmax_a \mathrm{Q}(s_\tau, a_\tau) \tag{7}$$

The above defined update rule for Q defines a feedback rule in which $\alpha_\tau$is a gain factor and $[r_{\tau+1} + \lambda\, max_a Q(s_{\tau+1}, a) - \mathrm{Q}(s_\tau, a_\tau)]$ is the error between a mixed observed and estimated state action value $(r_{\tau+1} + \lambda\, max_a Q(s_{\tau+1}, a))$ and the previous estimate for $\mathrm{Q}(s_\tau, a_\tau)$. A similar policy that guarantees faster state exploration and convergence of the algorithm is the $\epsilon - greedy$ Q-Learning policy, in which, at each mapping step, action $argmax_a \mathrm{Q}(s_\tau, a_\tau)$ is taken with probability $1 - \epsilon$, while a random action is taken with probability ϵ. The effect observed is that of letting the algorithm escape from local minima. ϵ is commonly referred to as exploration parameter, since the greater it is, the further we are from the pure Q-Learning policy. In practice, ϵ can be chosen large at the beginning of the operation, to explore the policy space, and then decreased when convergence to the optimal policy is assessed.

Based on the above, the RL service mapping problem can be therefore stated as follows.

Problem (*Reinforcement Learning based Service Mapping*)

Given the current state of the system s_τ and the observed reward r_τ, act according to the Q-learning $\epsilon - greedy$ policy

$$\pi_\tau(\mathrm{s}) = \begin{cases} \mathrm{argmax_a Q(s_\tau, a_\tau)} & \text{with prob. } 1 - \epsilon \\ \mathrm{rand(a)} & \text{with prob. } \epsilon \end{cases} \tag{8}$$

and update the state action value function according to (6).

The flow of the proposed strategy is presented in the figure below.

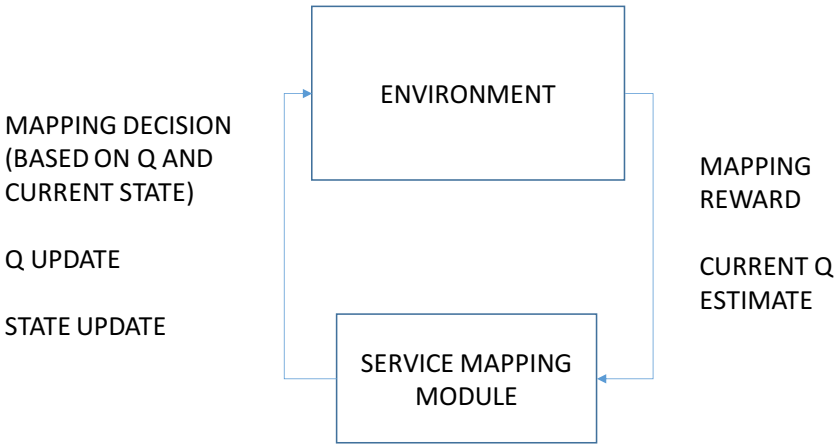

Figure 10 Reinforcement Learning workflow.

The presented algorithm for mapping VNFs is outlined in the following table, in the form of pseudo code.

Table 2 Pseudo code for the VNF mapping based on reinforcement learning

Algorithm 1 Virtual Network Function Mapping Based on Reinforcement Learning
1 **Start** 2 initialize state action value function Q

```
3 listen for incoming VNF activation/termination requests
4 receive incoming VNFs activation/termination request
5 retrieve VNF requirements
6 if VNF activation event
7       measure current state according to (2)
8       map the VNF based on Q-Learning policy (8)
9       check PoP feasibility
10      if feasible
11              positive reward
12      else
13              failure reward (e.g. zero or negative reward)
14      endif
15      update Q function based on (6)
16 else (i.e. VNF termination event)
17       compute state after VNF resources are released
18       reward=0
19       update Q function based on (6)
20 endif
```

3.2.2. NS Mapping Algorithm

The above sections were concerned with a reinforcement learning strategy for mapping VNFs. This section proposes an extension of the strategy to the case in which the mapping of an entire NS is sought (i.e., a NS composed by more VNFs).

The procedure for mapping the VNFs is kept unchanged, but the following two additional steps are added:

1. *VNF link – PoP path mapping*. In the general case, a NS is composed by more VNFs related by topological constraints (expressed by the forwarding graph) and link requirements. In addition to the VNF-PoP mapping action, it is therefore necessary to also decide how links among the VNFs should be mapped on the paths connecting the PoPs hosting the mapped VNFs.
2. *Feasibility check extended to link requirements.* When the VNF link mapping problem is considered as well, it is necessary to check that the actions decided by the reinforcement learning mapping engine do not lead to the violation of the link resources requirements (such as available link bandwidth, maximum admissible delay between mapped VNFs, etc.). Therefore, in addition to the PoP capacity feasibility check, the feasibility check on link requirements is added.

The resulting algorithm for NSs mapping based on reinforcement learning is outlined below, in pseudo code formalism.

Table 3 Pseudo code for the NS mapping based on reinforcement learning

Algorithm 2 Network Service Mapping Based on Reinforcement Learning
1 **Start**

```
2 initialize state action value function Q
3 listen for incoming events (NS activation or termination events)
4 receive incoming NS activation or termination event
5 retrieve NS data (number of VNFs, node/link requirements, etc.)
6 if NS activation event
7       measure current state according to (2)
8       for i=1 to number of VNFs composing the NS
9             map the VNF based on Q-Learning policy (8)
10            check PoP feasibility
11            map ingress and egress VNF links
12            check link feasibility
13            if feasible
14                  positive reward
15            else
16                  failure reward (e.g. zero or negative reward)
17            endif
18            update Q function based on (6)
19       endfor
20 else (i.e. NS termination event)
21       compute state after NS resources are released
22       reward=0
23       update Q function based on (6)
24 endif
```

The above strategy has a degree of freedom in the choice of the VNF link mapping strategy (line 11). Driven by the mapping goals of selecting feasible PoP paths leading to minimal mapping costs and balanced link resources' consumption patterns, a shortest path link mapping strategy has been selected, in which the cost matrix of the shortest path problem is given by a linear combination of the matrix carrying the information on the link mapping costs, and the matrix carrying information on the link congestion metrics. While the cost matrix can be assumed fixed, the "link congestion matrix" has to be updated regularly, to reflect the current congestion level of the infrastructure. This link mapping strategy is suited to the problem because it reflects the mapping goals and it is fast (the shortest path problem can be resolved with fast algorithms such as Dijkstra). The balance between the goal of minimizing link mapping costs and link balancing can be adjusted by tuning the coefficients governing the linear combination of the cost and the balancing matrices, similarly to what is done in the selection of the related terms in the objective functions of the proposed ILP mapping strategies.

3.3. Multi-stage Network Service Embedding

This approach has been proposed by the Gottfried Wilhelm Leibniz Universität Hannover (LUH). Previous work can be found in [31], [24].

3.3.1. Main assumptions

This strategy is based on the assumption that the NF providers (DC operators) advertise PoP-level graph with link costs, and the NF cost, i.e., CPU cost at the DC, that are expressed as weights in the algorithm description. The strategy is based on an ILP model applicable to any DC topology such as the two-level hierarchical fat-tree.

3.3.2. Iterative Algorithm

The proposed mapping algorithm is based on the following steps.

1. Identify location-dependent VNFs (e.g., proxies; resources should be in proximity of the client's network).
2. Identify candidate DCs for each VNF in the service chain.
3. If there is no DC satisfying all VNF requirements and constraints, partition the service chain among DCs:
    - ILP model as shown in Section 3.3.3 below.
    - Different objectives can be considered, depending on the service and NF providers' preference, like for example:
        - Minimizing the client's expenditure.
        - Maximizing load balancing across the DCs by considering (i.e., minimisation of) weight values that express NF Service Providers' preferences.
4. Upon partitioning, assign the VNFs to servers within the selected DCs (second level mapping problem):
    - Formulation as (Integer) Linear Program similar to existing Multi-commodity-flow problem formulations.
        - Objectives: Minimize inter-rack traffic and the number of used servers.
    - Alternative solution: Heuristic algorithm that aims at assigning the VNFs to the smallest number of racks and servers, while CPU load and bandwidth are balanced across the racks and servers.
5. Stitch together the VNF service chain segments (mapped to different DCs) with the assignment of virtual links connecting the DCs:
    - Objectives: To find the shortest path between a pair of DCs that offers the required amount of bandwidth.
    - Multi-commodity flow problem formulation.

3.3.3. ILP Model for Service Chain Partitioning

Next, we present our ILP model for service chain partitioning. We consider each link associated with a weight value that expresses the utilization of links and DCs. The objective function of the ILP aims at minimizing the sum of used weights w_{pq} which essentially leads to network load balancing, and therewith, to increased resources efficiency.

In the ILP formulations, we use the binary variable y_p^h to express the assignment of VNF h to DC p. Similarly, the binary variable x_{pq}^{hk} indicates whether the VNF graph edge $(h,k) \in A$ has been mapped onto the PoP-level graph edge $(p,q) \in A^I$ (same notation is used as for the first ILP approach in Section 3.1).

The service chain request partitioning ILP is therefore defined as follows.

Problem (*Service mapping based on Service Chain Request Partitioning*)

Minimize:

$$\sum_{(p,q)\in A^I} w_{pq} \sum_{(h,k)\in A} RR_{hk}^t \, x_{pq}^{hk} \qquad (1)$$

subject to:

$$\sum_{p\in V^I} y_p^h = 1 \qquad \forall h \in V \qquad (2)$$

$$\sum_{q\in V^I} \left(x_{pq}^{hk} - x_{qp}^{hk}\right) = y_p^h - y_p^k \qquad h \neq k, \qquad \forall (h,k) \in A, \forall p \in V^I \qquad (3)$$

$$y_p^h \in \{0,1\} \qquad \forall h \in V, \forall p \in V^I \qquad (4)$$

$$x_{pq}^{hk} \in \{0,1\} \qquad \forall (h,k) \in A, \forall (p,q) \in A^I \qquad (5)$$

Hereby, we briefly discuss the ILP constraints.

Constraint (2) ensures that each VNF h is mapped exactly to one DC p. Condition (3) preserves the binding between the VNF and the link assignments. More precisely, this condition ensures that for a given pair of assigned nodes i, j (i.e., VNFs or end-points), there is a path in the network graph where the edge (i,j) has been mapped. Finally, the conditions (4) and (5) express the binary domain constraints for the variables y_p^h and x_{pq}^{hk} . In addition, we fix the assignment of each end-point h in the request to its respective location p by setting $y_p^h \leftarrow 1$.

3.3.4. MIP Model for NF-subgraph Mapping

The MIP for VNF-subgraph mapping aims at maximizing NF co-location while minimizing the traffic within the DC. In this respect, the binary variable y_p^h denotes the assignment of VNFc h to the server p, while the binary variable z_p indicates whether the server p has been assigned to any VNFcs (i.e., $z_p = 0$ when there is no VNFc assigned to server p; otherwise $z_p = 1$).

Based on the multi-commodity flow problem formulation, we use the term commodity, defined as $m^{ij} = \{i, j, RR_{hk}^t\}$, to express the bandwidth demands RR_{hk}^t between a pair of VNFcs h, k. In this context, the flow variable f_{pq}^{hk} denotes the amount of flow (i.e., bandwidth units) over the DC link (p,q) for the VNF-graph edge (h,k) .

The VNF-subgraph mapping MIP is therefore formulated as follows.

Problem (VNF-subgraph Service mapping)

Minimize:

$$\sum_{p \in V^I} z_p + \frac{1}{\sum_{h,k \in A^F} RR_{hk}^t} \sum_{\substack{(p,q) \in A^I \\ p \neq q}} \sum_{(h,k) \in A^F} f_{pq}^{hk} \quad (6)$$

subject to:

$$\sum_{p \in V^I} y_p^h = 1 \qquad \forall h \in V^F \quad (7)$$

$$\sum_{\substack{q \in V^I \\ p \neq q}} \left(f_{pq}^{hk} - f_{qp}^{hk} \right) = RR_{hk}^t \left(y_p^h - y_p^k \right) \qquad h \neq k, \qquad \forall (h,k) \in A^F, \forall p \in V^I \quad (8)$$

$$\sum_{h \in V^F} RR_h^t \, y_p^h \leq RA_p^t \, z_p \quad \forall p \in V^I \quad (9)$$

$$\sum_{(h,k) \in A^F} f_{pq}^{hk} \leq RA_{pq}^t \quad \forall p, q \in V^I \quad (10)$$

$$y_p^h, z_p \in \{0,1\} \qquad \forall h \in V^F, \forall p \in V^I \quad (11)$$

$$f_{pq}^{hk} \geq 0 \qquad \forall (h,k) \in A^F, \forall (p,q) \in A^I \quad (12)$$

The objective function (6) consists of two terms, i.e., the number of assigned servers and the accumulated flow divided by the total bandwidth demand. Essentially, the second term yields 1 if all NF-graph edges $(h,k) \in A^F$ are mapped onto single-hop links $(p,q) \in A^I$. The normalization of the second term provides a balance against the first term in the objective function.

We further discuss the constraints of the MIP model. Condition (7) ensures that each VNFc h is mapped exactly to one server p. Constraint (8) enforces flow conservation, i.e., the sum of all inbound and outbound traffic in switches and servers that do not host VNFcs should be zero. Constraints (9) and (10) ensure that the allocated computing and bandwidth resources do not exceed the residual capacities of servers and links, respectively.

Finally, condition (11) expresses the binary domain constraint for the variables y_p^h and z_p while constraint (12) ensures that the flows f_{pq}^{hk} are always positive. We further assume that the first element in V^F represents the virtual gateway which we bind to the physical gateway GW by setting $f_{GW}^{V_F(1)} \leftarrow 1$.

3.4. VNF Scheduling over an NFV Infrastructure

As described in previous sections, ETSI NFV defines NSs as entities composed of virtual network functions, which are the actual components performing the specific operations. Typically, network traffic associated to a given NS goes through several network functions. As authors state in the previously cited work [19], that means a set of network functions is specified and the flows traverse these functions in a specific order so that the required functions are applied. This implies precedence requirements between functions in the same service, which is known as the formalization of the function chaining.

The management and orchestration layer within the NFV stack, i.e. TeNOR in T-NOVA, is responsible for the deployment and operation of the different network services. The mapping problem, as it has been described in the previous sections – i.e. where the virtual network functions will be allocated in the NFVI infrastructure - becomes the fundamental challenge to solve. Different servers in the NFVI could have different processing capabilities, or different hardware characteristics, which will affect at the end the NS performance. While this is true for all the virtual network functions that process traffic continuously, e.g., deep packet inspection (DPI), there are other specific control functions which are only executed during a certain period of time, like the virtual path computation element (PCE) [32] or the multi-domain virtual forwarding function, as presented in [33]. These control functions can be also virtualized together with the actual traffic-handling functions, but a new challenge comes into the arena for the last group of virtual network functions; it is the scheduling, i.e., *when* is it better to execute each function in order to minimize the total execution time without, at the same time, degrading the service performance and respecting all the precedencies and dependencies among the functions composing the service.

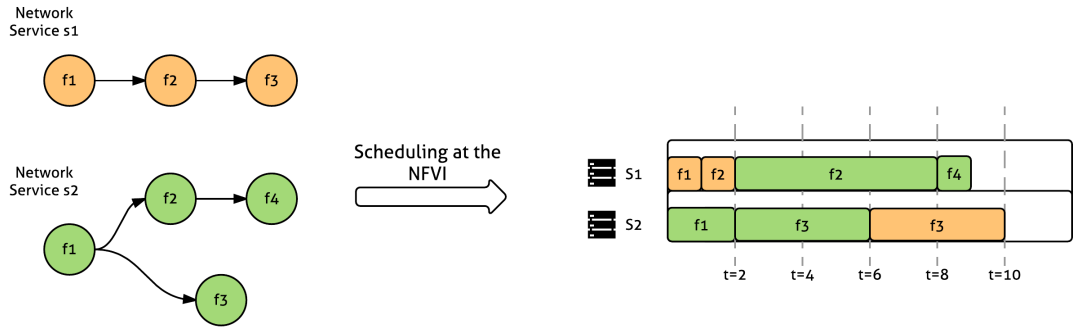

Figure 11 Network service (control functions) scheduling onto NFVI

Discussion on whether these specific virtual control functions must be considered as virtual network functions is left out the scope of this deliverable. The NFV scheduling problem has been defined within T-NOVA and published in [34], [25], as well as it has been accepted as one of the problems to be solved, amongst others in [23]. The following NFV scheduling algorithm has been proposed by the "Fundacio Privada I2CAT, Internet I Innovacio Digital A Catalunya" (I2CAT).

Problem Description

In the model a number of sets are used. F represents virtual network functions, N represents network services, which are composed of different chains of virtual network functions, and S represents servers, which are the elements responsible for processing different functions and

services. Each network function $f \in F$ belongs to one of the network services $n \in N$. In fact, each network service is made of a number of network functions. The relation is modelled by sets $F(n)$, which, for each network service, contains all functions that constitute it. The notion of network service is also indirectly modelled by sets $F(f)$ that define relations between network functions, i.e., for each function we define a set (possibly empty) of network functions that cannot be initiated before the considered network function is successfully executed. The sets are listed below:

- F network functions.
- N network services.
- $F(n)$ network functions belonging to network service n.
- $F(f)$ network functions that cannot executed before finishing the execution of network function f.
- S servers.

There is also a set of constants in the problem, which are relevant for the calculations:

- $t(f,s)$ time needed by server s to execute function f.
- A weight of the finish time of the last served network function.
- M infinity; a sufficiently large constant in practice.

Constant $t(f,s)$ is responsible for a server classification. By setting different execution times for different network functions on different servers, it is possible to easily create classes of servers. In addition, setting appropriate constants $t(f,s)$ to sufficiently large numbers, we can easily block some functions from being executed on particular servers. However, such an operation can be done more easily by forcing appropriate values to be equal to zero. Another constant is A. It serves as a weight to indicate which part of the objective function is more important: the finish time of the last served network service, or the sum of the finish times of all network services. The last constant is the so-called "big-M" constant frequently used in mixed integer linear programs to express relations between binary and real variables.

Finally, the variables considered for the problem model are:

- z finish time of the last served network service.
- z_n finish time of network service n.
- v_f starting time of executing network function f.
- e_{f_s} binary variable; 1 if network function f is executed at server s; 0 otherwise.
- a_{ff}, binary variable; 1 if network function f is executed after f'; 0 otherwise.

The objective function is defined as follows

$$min\ Az + \sum_{n \in N} z_n$$

In the formulation, the objective function consisting of two components is minimized. The first component is the time needed to execute all network services, while the second component is the sum of the times needed to execute each network service. The following constraints are taken into account in the problem.

$$z \geq z_n \qquad \forall n \in N \tag{1}$$

$$v_f + \sum_{s \in S} t(f,s)\, e_{f_s} \leq z_n \qquad \forall n \in N, \forall f \in F(n) \tag{2}$$

$$v_f + \sum_{s \in S} t(f,s)\, e_{f_s} \leq v_f, \qquad \forall f \in F, \forall f' \in F(f) \tag{3}$$

$$v_f + t(f,s) \leq v_{f'} + M\left(a_{ff'} + 2 - e_{f_s} - e_{f'_s}\right),\ \forall f, f' \in F, \forall s \in S \quad (4)$$

$$a_{ff'} + a_{f'f} = 1, \qquad \forall f, f' \in F : f \neq f' \quad (5)$$

$$\sum_{s \in S} e_{f_s} = 1, \qquad \forall f \in F \quad (6)$$

Variable *z* represents a moment in time when all network services are already executed; thus, it cannot be smaller than the finish time of any network service, which is expressed by (1). On the other hand, the finish times of single network services cannot be smaller than finish times of network functions forming them. The relation is modelled by (2). Notice that in (2) the term $\sum_{s \in S} t(f,s)\, e_{f_s}$ is just a time needed to execute function f on a selected server represented by e_{f_s}. We consider network services that impose various constraints on network functions they are built of. This fact is represented by (3), in which time of executing network function f' that follows another network function f, i.e. $f' \in (P(f))$, has to be greater than the finish time of executing network function f. The next constraint, namely (4), prevents network services from being executed in parallel on the same server; it is in fact assumed that each server can process only a single network function at a time. In other words, consider two network functions f and f', assume that f is executed before f' (thus $a_{ff'} = 0$) and both are executed on server s. If all the conditions are met, constraint (4) reduces to $v_f + t(f,s) \leq v_{f'}$, which means that the starting time of executing network function f' has to be after the finish time of executing f. On the other hand, when at least one of the mentioned conditions is not satisfied ($a_{ff'} = 1$ or $e_{f_s} = 0$ or $e_{f'_s} = 0$), constraint (4) reduces to $a \leq b + cM$, where $cM \gg a, b \geq 0$; hence it is always satisfied regardless the values of the variables. Obviously, constant M has to be sufficiently large; in the considered problem it could be for instance equal to the minimum time needed to execute all functions on the fastest server. Finally, constraint (5) ensures that not all the a_{ff} variables can be equal to 1, while constraint (6) ensures that all network functions will be executed.

The scheduling problem is still under investigation from the theoretical perspective, and future work will analyse the relationship between network functions scheduling and job/task scheduling in traditional computer science (i.e. as processor oriented research), where huge work on scheduling has been performed. On the other hand, there is at least one additional initiative in the literature, which integrates both mapping and scheduling of network functions into one complex optimization problem [35]. TeNOR prototype does not include scheduling of network functions.

In order to solve the problem, different methods could be utilized. For example, a heuristic as a variation of a greedy approach that in each iteration schedules a network function which minimizes the overall time assuming that the remaining network functions are scheduled on the fastest finishing servers. Such a scheduling problem could be utilized to determine the performance of different DC distribution, such as edge-computing, with micro-DCs at the edge, and traditional Cloud computing in order to execute specific virtualized control functions.

3.5. Summary of the Features of the Algorithms

Table 4 below reports a summary of the main features of the algorithms, for ease of comparison.

Table 4 Compared view of main algorithms' features

Proposed Approach	ILP Based Algorithms	Reinforcement Learning based approach	Multi-stage Network Service Embedding	VNF Scheduling over an NFV Infrastructure
Objectives	Cost minimization or load balancing	Revenue maximization	Cost minimization or load balancing	Cost minimization (minimal makespan)
First level or second level[4], exact or approximate	First level. 1st level: exact or heuristic	First level. 1st stage: approximate	First level and second level. 1st level: exact 2nd level: exact or heuristic	Second level. Exact or Heuristic
Online/Offline	Online	Online (also, the algorithm could be trained offline)	Online	Online
Resource constraints	SLA, node resources, link resources	Node resources, link resources	CPU, bandwidth, location	Infrastructure resources available
Node/link mapping	Coordinated	Uncoordinated[5]	Coordinated[6]	-
Dependencies	Optimizer, e.g. GLPK[7] or CPLEX	-	Optimizer, e.g. CPLEX[8]	-

[4] First level mapping: the problem of assigning VNFs to PoPs. Second level mapping: the problem of assigning VNFcs to servers in DCs.

[5] The single iteration of the algorithm is uncoordinated. In the long run, learning is such that the node and link mapping are actually coordinated decisions.

[6] Node mapping and link mapping takes place simultaneously, i.e., we can rethink node mapping if the overall cost is lower. In contrast, uncoordinated means the node mapping is fixed before we start mapping the links, even at the risk of rejection.

[7] GLPK is an open source tool for solving linear mathematical optimization problems (see https://www.gnu.org/software/glpk/).

[8] CPLEX, developed by IBM, is one of the most efficient commercial tools available for solving mathematical optimization problems (see http://www.ibm.com/software/commerce/optimization/cplex-optimizer/).

4. SERVICE MAPPING MODULE IMPLEMENTATION AND INTEGRATION

This section give details on how the T-NOVA service mapping module has been implemented and integrated into TeNOR, the T-NOVA orchestrator.

The documentation of the service mapping module API, and of the data model and the format used to exchange information between the service mapping module and the other TeNOR modules is reported in the annexes.

4.1. Implementation

Similar to the other main orchestrator components, the service mapper has been implemented as a microservice application, which is composed by:

- A REST service (written in Ruby [36]).
- A compiled application (written in C++).

The two modules have each a specific task: in a nutshell, the compiled application implements the actual solver of the service mapping mathematical problem, while the REST service acts as an interface between the other microservices in the orchestrator and the solver (i.e. it gathers the inputs needed to build the mathematical mapping problem, passes them to the solver and returns the solution to the orchestrator).
Details of the REST service are provided in the next section of this document, devoted to the integration work. A schema of the whole microservice architecture is given in Figure 12 below.

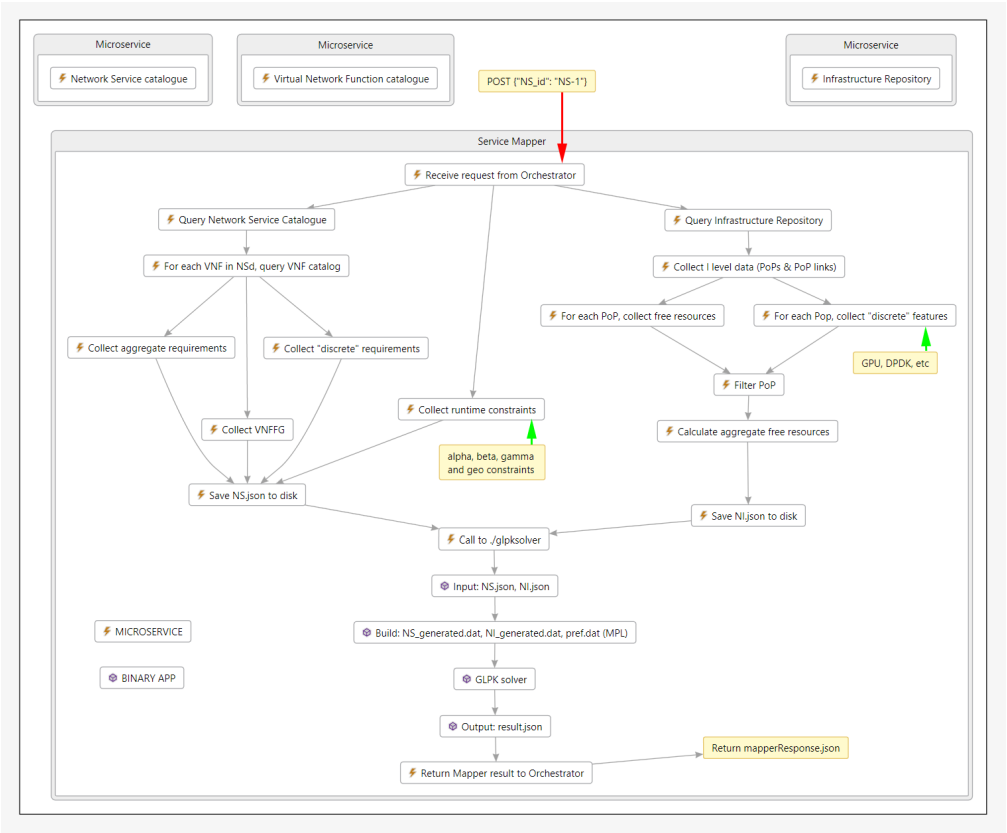

Figure 12 Details of the T-NOVA Service Mapper microservice.

As represented in the figure, when the service mapping module is called by the orchestrator, the operations are divided into three main flows:

1. Querying of the network service catalogue (left branch in the figure).

2. Querying of the infrastructure repository (right branch in the figure).
3. Collection of constraints (central branch of the figure) varying with each mapping instance/request, such as: solver parameters – which can be chosen to tune the behaviour of the solver –, geographical constraints on the mapping, etc.

Details on network infrastructure and service catalogue querying are reported in the next section, discussing the integration of the service mapping module. After the querying phase, all the relevant data input to the service mapping mathematical problem are stored in two distinct JSON files:

1. *NI.json*, a file storing all the data from the infrastructure repository which enter the service mapping problem mathematical formulation.
2. *NS.json*, a file storing all the data related to the network service to be mapped and relevant for service mapping (also, NS.json file is enriched with the above mentioned runtime information on algorithm parameters, additional user constraints, etc.).

After the above inputs are gathered, they are passed to the actual solver of the service mapping problem.

Here the focus is on the implementation provided by UniMi of the ILP mapping approach proposed in Section 3.1. Other mapping approaches could be similarly integrated in the future, by adding to the mapper microservice a routing logic which enables the mapping algorithm to be called.

In the above figure, the call to the solver is represented by the block "call to ./glpksolver", since an open-source solver, the GNU Linear Programming Kit (GLPK), has been deployed to solve the ILP service mapping problem.

GLPK needs two basic structures for correctly instantiating the optimization problem:

1. *A .mod file,* containing the Mathematical Programming Language (MPL) program that implements the abstract mathematical model described in the previous sections on algorithms' description.
2. *At least one .dat file* containing the inputs to the mapping problem, collected by the REST service (i.e. the inputs from the infrastructure repository and the service catalogue, etc.).

The developed C++ application – which spans, in the above figure, from the "call to ./glpksolver" block up to the "Output:result.json" block – acts as a wrapper for the GLPK solver; this application is invoked and executed by the REST service once all the network infrastructure and network service data required by the solver have been collected and locally saved to disk into the two input files NI.json and NS.json.

The application is composed by two distinct binary executables, which are sequentially invoked by the REST service:

1. *jsonConverter*, which parses and further processes both files in order to create the .dat files required by GLPK; data extracted from the NI.json and NS.json files are roughly divided into three .dat files:
    a) *NI_generated.dat* file*,* containing the snapshot of the network infrastructure, i.e. the list of PoPs and their connection topology, as well as additional information regarding each PoP and each link between them (i.e. computing power and capabilities, bandwidth and delay of network connections, etc.).
    b) *NS_generated.dat* file, which contains the description of the network service and its requirement, i.e., the list of VNFs and their connection graphs, as well as minimal hardware and bandwidth requirements, etc.
    c) *pref_generated.dat* file, which contains a list of variables that are not strictly related to NI or NS, but are used by the solver to adapt the solution to any

need specified by the customer (i.e. locking a VNF to a specific PoP) or by the network status / flavour / service characteristics by steering the solution in order to meet those requirements (i.e. preferring delay minimization instead of PoP spreading); these variables are NI and NS independent and may vary at each allocation request.

2. *solver*, which acts as a wrapper for the GLPK solver:once the .dat files are successfully imported, this application allocates an MPL problem, fills its workspace with both the model and the data files, and invokes the actual GLPK solver.

The last task of the solver builds and returns the solution to the REST service, in json format. A response is provided even if the problem has no feasible solution, while error messages are generated if something goes wrong when creating and instantiating the MPL problem or the GLPK workspace.
The returned feasible solution (if successfully found) states the target PoP of allocation for each VNF and the path of allocation for each link between VNFs.

Data transfers between the REST service and the C++ application package is achieved by locally storing files in json format. Since ANSI C++ has no built-in support for parsing json files or streams, the Daniel Parkers's "jsoncons" open-source (under Boost license) library[9] for processing data in this format has been used.

It is worth noting that by dividing the json-to-MPL conversion phase from the actual solver may reduce the future amount of work needed for the implementation of different solvers than GLPK.

4.2. Integration

The REST service mentioned in the previous section takes care of the integration of the service mapper with the Orchestrator components that play a role in service mapping, such as sources of information needed to build the mathematical mapping problem or as entities responsible for enforcing the mapping decisions.
By logically separating the interface with the orchestrator (by means of the REST service) from the actual service mapping solution phase (the C++ applications), a strategy for providing a common platform for data collection and aggregation has been devised. So different solving algorithms can be potentially integrated into the platform either by changing the core .mod file that reports the formulation of the optimization problem to be solved (if the mapping algorithm is still in the family of linear programming), or by changing the solver and the needed parts of the C++ apps (in case of a mapping algorithm not based on linear optimization). Moreover, such integration scheme provides a common testbed so that evaluation of different algorithms may be performed on the same set of data, even in operational environment.

Briefly, the main interactions of the REST service with the other involved TeNOR modules are:

- Receive and validate instantiation requests from the orchestrator.

9 https://github.com/danielaparker/jsoncons/wiki/json

- Collect data needed by the mapping algorithms, by querying the infrastructure repository, the network service catalogue and virtual network function catalogue APIs.
- Save the collected data into the NI.json and NS.json input files.
- Invoke the mapping solver (the Unimi C++ application).
- Return the solution to the orchestrator and to the visualization service.

The service mapper REST service responds to explicit POST commands coming from the orchestrator, with the body payload containing, in particular, the ID of the network service to be instantiated, so that it is possible to properly query the service catalogue to retrieve the needed parameters of the network service.

Once the request has been checked, the operation flow proceeds according to the following steps (the reader may refer for convenience to Figure 12):

1. The REST service queries the network service catalog API to get a descriptor of the network service; once successfully received, this NSD is parsed and the relevant data are extracted (composing the virtual network functions IDs and their forwarding graphs, as well as any other first level requirements).
2. For each virtual network function ID collected in the previous step, the virtual network function descriptor is requested from the virtual network function catalog and, once successfully received, the relevant data are extracted from the descriptors: each VNFd is parsed, looking for the requested flavour (or the default one). Since every VNF may be composed by different VDUs[10], we chose to collect the total *aggregated requirements* of each VNF (e.g., for the memory requirement, the aggregated virtual memory requirement of a VNF is the sum of the "virtual_memory_resource_element" field of each VDU composing the VNF). Currently, collected aggregate data regard the number of virtual CPUs, the amount of virtual memory and the required bandwidth; also in addition, the maximum allowed delay between VNFs is collected too. Also the virtual forwarding graph (as well as the descriptors of the virtual link involved) which interconnects the VNFs is collected.
3. All the data collected by parsing the NSd and the VNFds are saved into a NS.json file on disk; this file may contain additional parameters that are NS/VNF independent and vary on allocation basis (at run-time), i.e. geographical location requirements, solving strategy, additional parameters to be passed to the solver, etc.
4. Once NS collection finishes, the REST service begins querying the infrastructure repository APIs, in order to build a snapshot of the network infrastructure status. Data collected include, but are not limited to, PoP IDs, PoP description and status, PoP links IDs, PoP links descriptions and statuses, etc.; then, for each PoP, every hypervisor is queried in order to extract the total aggregated resource available for that PoP.
5. Aggregated resource of each PoP is calculated by summing the available resources: data regarding the total amount of available CPUs, RAM, HD and bandwidth is collected in this same way. Enhanced Platform Awareness (EPA) requirements are treated in a similar way: the current implementation counts the total number of DPDK-enabled NICs and the total amount of GPU accelerator for each PoP, but other EPA features can be added as well. All network infrastructure collected data are saved to disk in the NI.json file.
6. Once both NS.json and NI.json are saved, the solver is invoked with a system call. For the service mapping algorithm implemented, the one proposed by Unimi, it consists

[10] Recall that a Virtualisation Deployment Unit (VDU) [5] is a construct to describe the deployment and operational behaviour of a VNFc.

in a sequence of system calls to the C++ applications detailed in the above section. The solver response is saved to disk in the "MapperResponse.json" file and data related to mapping visualization are built and added to the json; finally, this json response is returned to the Orchestrator.

As mentioned above, the two key T-NOVA subsystems queried by the mapping microservice are the infrastructure repository and the service catalogue.

Essential information on the two subsystems is provided in the following two sections.

Details on the service mapping module API and on the format and content of the data exchanged between the mapping module and the orchestrator are reported in the annexes.

4.2.1. Interaction with the Infrastructure Repository

This section reports the key features of the infrastructure repository instrumental to service mapping. Full details on the infrastructure repository design can be found in D3.2 Infrastructure Resource Repository [37].

The infrastructure repository is the subsystem of the T-NOVA Orchestration layer which provides, to the resource mapping module, infrastructure related information collected from the Virtualized Infrastructure Manager (VIM) and from the NFVI components of the Infrastructure Virtualisation and Management (IVM) layer. The design of the repository subsystem addresses the challenges of assimilating infrastructure related information from sources within the IVM, namely the cloud infrastructure and data centre network environments. This subsystem comprises a number of key elements including a data model, resource information repositories and access mechanisms to the information repositories. The subsystem also augments the information provided by cloud and SDN environments through a resource discovery mechanism.

OpenStack until recent versions exposed a limited set of infrastructure related information, and that was problematic for the resource mapping module. For example in the Juno release of OpenStack the NOVA service database contained almost no infrastructure information beyond limited CPU details such as manufacturer, speed and CPU status flags. Subsequent releases of OpenStack have improved the level of infrastructure information stored. However inclusion of fine grained infrastructure information remains a work in progress and will not be fully addressed in the lifetime of the T-NOVA project.

Generation of a comprehensive view of the network infrastructure has been another challenge for the infrastructure repository. While OpenStack Neutron service database maintains information regarding the VM related network connections there is no view of the physical network topology i.e. what compute node NIC is connected to an SDN enable switch. In order to address this limitation it is necessary to collect these information from the DC SDN controller i.e. OpenDaylight. ODL provides a REST API which can be used to retrieve information regarding the topology of the SDN switches and of the compute nodes attached to the switch ports. From a resource mapping module perspective using a separate interface to retrieve network topology information creates unwanted complications. Therefore from a design and implementation perspective the repository implements a single interface which abstracts the retrieval of infrastructure information from all sources within the IVM layer.

Addressing these shortcomings was the key goal for the repository subsystem design and implementation. The key components of the infrastructure repository as shown in Figure 15 are the following:

- **Enhanced Platform Awareness (EPA) Agents** – Python based software agent running on the compute nodes of the NFVI-PoP. A central EPA controller service provides aggregation of data from each agent and persists the data to a central database.
  - **Infrastructure Repository Database** – Collected infrastructure data is stored in a graph database where resources are represented as nodes with associated properties. Edges between the nodes store information on their relationship.
  - **Listener Services**- Two separate listener services are specified within the architecture. The OpenStack Notification listener service is designed to intercept messages from the OpenStack notification.info queue and to provide notifications to the controller. The EPA agent listener service intercepts EPA agent messages and notifies the controller of the messages in order to trigger processing of received data files and to use the data to carry out an update database action.
  - **EPA Controller –** The controller is responsible for updating the infrastructure database based on information received from the listener services and data files sent by the EPA agents. One instance of the controller runs in each NVFI-PoP. The service runs on a compute node within the NFVI.
  - **API Middleware Layer** – Provides a common set of API calls that can be used by all the functional entities of the T-NOVA Orchestration layer including the resource mapping module.

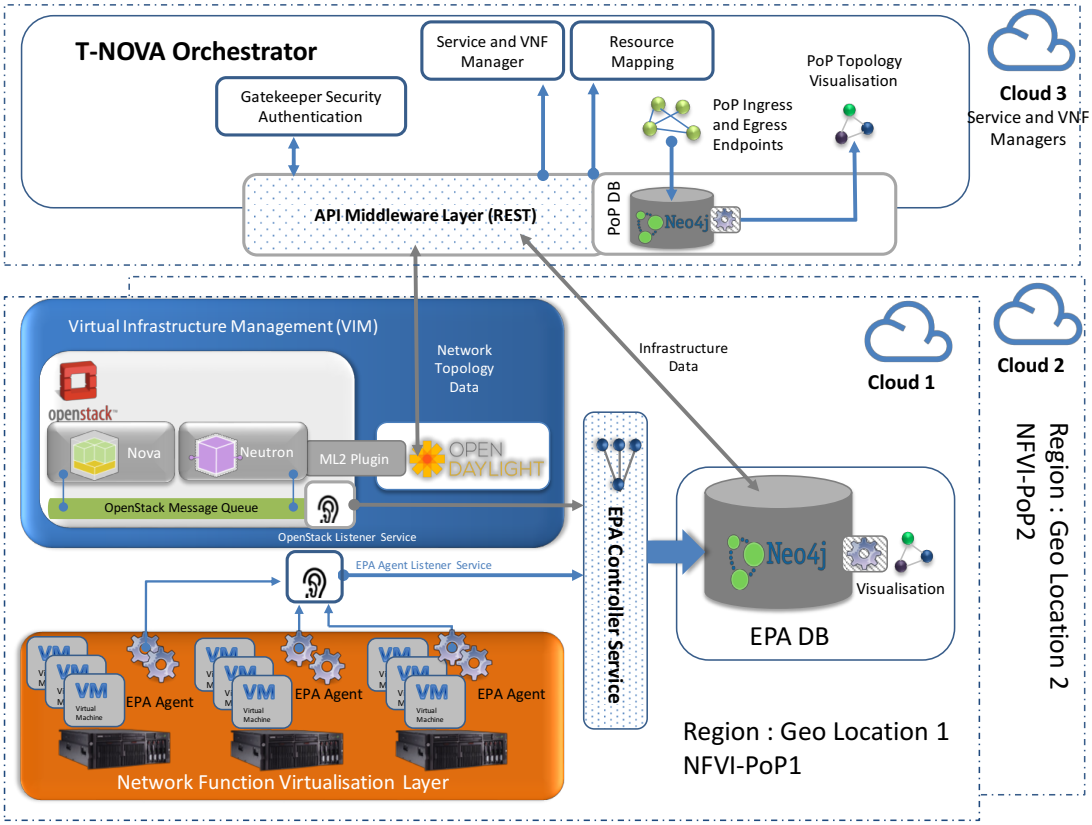

Figure 13 Infrastructure Repository sub-system architecture

4.2.1.1. Infrastructure Repository Database

Regarding the implementation of the infrastructure repository database it was considered useful and important to encode the relationship between the resources and associated parameters. In order to achieve this goal a graph database approach was adopted (see the T-

NOVA deliverable 3.2 [37] for more details). Graph databases are NoSQL (Not only SQL) database systems which commonly use a Directed Acyclic Graph (DAG) to store data relationships. A graph database stores information using vertices/nodes and edges/relations.

The use of graphs maps conveniently to the hierarchical structure of compute, storage and network elements within the NFVI. The NFVI can be decomposed into either physical or virtual resources that can be mapped directly to a node structure. The relationships of virtual-virtual, physical-physical and virtual-physical are captured in the relevant connections between the resources. Virtual resources have an implicit dependency on physical resources, i.e. a virtual resource cannot exist without a physical host. Therefore in a graph construct, virtual resources must at some point of the graph be connected to a physical resource

Figure 14 shows a simple graph for a server, where it has two sockets, each socket has one CPU and each CPU has multiple cores.

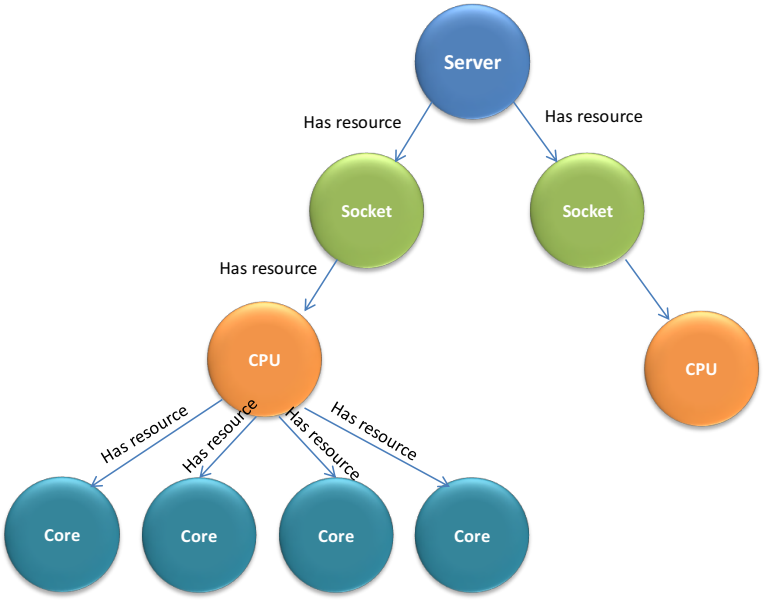

Figure 14 Simple server resource graph

Middleware API

To achieve a unified view of the infrastructure information among multiples PoPs, the infrastructure repository implements a middleware layer. The main responsibilities of the middleware layer are:

- Defining a common view for all information sources (OpenStack, EPA Agents and OpenDaylight SDN controller);
- Dispatching user requests to the required PoP.

The middleware layer exposes the API calls to manage the PoPs (Create, Retrieve, Update, Delete). When a request is sent to the middleware layer, the ID of the PoP that the user wants to access must be specified. In this way the middleware layer can retrieve the URLs and query the appropriate PoP level information sources. From the perspective of a component using the interface the location of the data and the underlying complexity in forming the query response is abstracted. To be compliant with the other Orchestrator interfaces a REST type approach was adopted to implement the interfaces (see deliverable D3.2 Section 4.5.1 for more details). Some sample calls to retrieve specific infrastructure information available to the resource mapping module are shown in Table 5.

Table 5 API calls for specific infrastructure information retrieval

Kind	Endpoint url	Description	Actions
Net	/pop/{pop_id}/net/	Neutron network	GET
Port	/pop/{pop_id}/port/	Neutron port	GET
Hypervisor	/pop/{pop_id}/hypervisor/	Hypervisor used by Nova Compute	GET
Machine	/pop/{pop_id}/machine/	Physical machine	GET
NUMA Node	/pop/{pop_id}/numanode/	NUMA node	GET
NUMA node link	/pop/{pop_id}/numanode/link/	Link between NUMA node and Bridge or Socket	GET
PCI Bridge	/pop/{pop_id}/bridge/	PCI Bridge	GET
Bridge link	/pop/{pop_id}/bridge/link/	Link between PCI bridge and PCI devices connected to it	GET
PCI Device	/pop/{pop_id}/pcidev/	PCI Device	GET
PCI Device link	/pop/{pop_id}/pcidev/link/	Link between PCI Device and respective OS device	GET
OS Device	/pop/{pop_id}/osdev/	OS Device (allowed type: Compute, Network, Storage)	GET
Socket	/pop/{pop_id}/socket/	Socket	GET
Cache	/pop/{pop_id}/cache/	Cache	GET
Core	/pop/{pop_id}/core/	Physical Core	GET
Core link	/pop/{pop_id}/core/link/	Link to Process Units node	GET
PU	/pop/{pop_id}/pu/	Processing unit	GET
Switch	/pop/{pop_id}/switch/	Physical SDN switch	GET
Switch Interface	/pop/{pop_id}/switch-interface/	Switch interface controlled by ODL controller	GET

Technical details on the sequence of actions performed to query the infrastructure repository and an example of the resulting NI.json file are reported in Annex 11.

4.2.2. Interaction with the T-NOVA NSD

The SLA specification in T-NOVA is done in the marketplace by the Service Provider (SP) [38] and will be available to TeNOR by means of the T-NOVA NSD that is stored in the service catalogue [39]. The SP will define in the marketplace expected values for the different service

performance metrics, considering the performance metrics of VNFs that are part of the service. Also other metrics can be defined for the end-to-end service considering the links between VNFs, for instance the delay added by the links.

Therefore, each SLA specification is associated by means of the NSD to the VNFDs of the VNFs that constitute the service. These VNFDs describe the VDU requirements defined by the Function Providers (FPs) for the corresponding VNF flavor in each case.

The marketplace will also define, as part of the SLA specification, the rewarding for the customer, like billing discounts, in case the SLA agreed between the SP and customer is not met. This is relevant for the commercial activity in the T-NOVA marketplace and not for TeNOR itself, but it represents the relevance for the service mapping to achieve the SLA specified for the service.

The NSD and VNFD fields relevant for the service mapping are reported in the following tables.

Table 6 NSD:SLA

Identifier	Description
SLA_Id	ID of the service SLA
assurance_parameters	Assurance parameter or a combination of multiple assurance parameters with a logical relationship between them and values against which this SLA is being described. The parameters should be present as a NSD: monitoring_parameter, and they can be: - generic metrics common to all the network services that will be monitored (e.g. network throughput metrics). The basic metrics will have an associated threshold for action initiation. - Flavour_key parameters
flavour_key	Parameter or a combination of multiple assurance parameters with a logical relationship between them against which this flavour is being described.
constituent_vnf	Represents the characteristics of a constituent flavor element.

Table 7 NSD: SLA: assurance_parameters

Identifier	Description
name	Name of the metric
value	Range of values the metric should take to meet the SLA: LT(50) → lower than 50
unit	Unit of the values: %, Mb, Gb, etc
formula	Formula that aggregates the metrics of the VNFs in order to calculate the value above.
violations	Definition of SLA violations

Table 8 NSD: SLA: assurance_parameters: violation

Identifier	Description
breaches_count	Amount of times the thresholds are breached
interval	Time interval

Table 9 NSD: SLA: constituent VNFs

Identifier	Description
vnf_reference	Reference to a VNFD declared as VNFD in the network service via vnf:id.

vnf-flavour_id_reference	References a VNF flavor (vnfd:deployment_flavour:id) to be used for this service SLA.
redundancy_model	Represents the redundancy of instances, for example, “active” or “standby”
affinity	Specifies the placement policy between this instance and other instances, if any.
number_of_instances	Number of VNF instances satisfying this service assurance.

Table 10 VNFD: deployment flavor

Identifier	Description
id	ID of the VNF flavour
assurance parameters	A set of elements related to a particular monitoring parameter.
constraint	Constraint that this deployment flavour can only meet the requirements on certain hardware
constituent_vdu	Represents the characteristics of a constituent flavour element Examples include Control-plane VDU & Data-plane VDU & Load Balancer VDU Each needs a VDU element to support the deployment flavour of 10k calls-per-sec of vPGW, Control-plane VDU may specify 3 VMs each with 4 GB vRAM, 2 vCPU, 32 GB virtual storage etc. Data-plane VDU may specify 2 VMs each with 8 GB vRAM, 4 vCPU, 64 GB virtual storage etc.

Table 11 vnfd:deployment_flavour:constituent_vdu

Identifier	Description
vdu_reference	References a VDU which should be used for this deployment flavour by vnfd:vdu:id
number_of_instances	Number of VDU instances required
constituent_vnfc	References VNFCs which should be used for this deployment flavour by vnfd:vdu:vnfc:id

Technical details on the sequence of actions performed to query the service catalogue and an example of the resulting NS.json file are reported in Annex 12.

5. SIMULATION TESTS

The aim of this section is to present and discuss simulation tests for each service mapping algorithm proposed in T-NOVA, with the objective of highlighting the peculiarities of each approach, its strengths and the possible drawbacks and areas for future works.

5.1. Integer Linear Programming based approach

We performed a computational evaluation of the method both to understand the relative impact of parameters choice in our models and to assess the scalability of the solvers, thereby validating the practical applicability of our framework. For this task we built a synthetic simulator, working as follows.

First, we populated the simulator with the Network Infrastructures (NI) and the Network Services (NS) included in the dataset [40] described in [41], a popular reference in the literature of mapping algorithms. In particular, such a dataset consists of 210 base instances, partitioned in 7 classes of increasing size networks, each one including 30 instances. Any instance contains a graph describing the NI and a number of smaller graphs describing the NSs. NS graphs belong to a limited number of topologies, representing different service types. NI (respectively, NS) graph nodes are annotated with one integer, indicating the CPU availability (respectively, consumption) on that node. NI graph nodes are annotated also with an additional integer, indicating the allocation cost on that node, which is independent on the NS node to be allocated. Similarly, NI (respectively, NS) graph arcs are annotated with two integers, one indicating the amount of available (respectively, consumed) bandwidth, and the other indicating the expected (respectively, required) delay on that arc. Furthermore, for each NS node a vector of compatibilities to NI nodes is given, indicating which mappings are feasible due to specific resources needed by the NS VNF on the host. We considered each arc in the NS graph (and only them) as critical paths, whose delay constraints have to be respected.

In Table 12 the features of this dataset are reported: for each class, we detailed the number of nodes in the NI, the average number of nodes in the corresponding NSs, the average number of NS types, the average fraction of NI-NS node mappings which are feasible, and the average ratio between the overall amount of available CPU at NI nodes and the overall required CPU of NS nodes.

Table 12 Instance details

Size	Average VNF nodes	VNF types	Average compatibility ratio	Available CPU / required CPU
20	5,52	4	18,98%	3,38
30	6,91	4	16,56%	3,88
50	9,87	4	13,81%	4,29
100	17,43	4	10,03%	5,1
Total Result	9,93	4	14,84%	4,16

These base instances represent static mapping problems, while those in TNOVA are dynamic. That is, each NS allocation request arrives at a certain point in time and has a certain duration: NI resources are allocated, if possible, when each request is issued, and released when the request is over.

Therefore, we completed our synthetic simulator considering two running modes: simulation and stress test.

In simulation mode, we model the arrival of NS allocation requests as a Poisson process, whose average interarrival time is denoted as λ. We also assume the duration of NS allocation requests to be random, following a normal distribution whose mean is denoted as μ and whose standard deviation is denoted as σ. A simulation run consists of selecting a particular base instance (that is, a NI and the corresponding pool of possible NSs), and then to (a) iteratively choose one NS at random, selecting it with uniform probability from the NS pool, (b) randomly draw an interarrival time and a duration for the corresponding allocation request, (c) possibly release the NI resources assigned to allocations requests terminating within the drawn interarrival time, (d) ask the mapping algorithm to find a suitable allocation of the drawn NS to the NI, (e) either reject the allocation request or accept it, and subsequently mark NI resources as used, according to the mapping provided by the algorithm. The simulation ends as soon as the sum of interarrival times exceeds a given time horizon τ, that represents the time length of the simulation. Values λ, μ, σ, and τ, and the base instance to be used, are parameters of the simulator.

In stress test mode, instead, we consider for a given base instance all NS allocation requests to arrive following the order in which they appear in the instance file, with a negligible fixed interarrival time, and a duration equal to τ. That is, in stress test mode all NS allocation requests arrive one after another, they are either rejected or allocated, and in the second case the assigned resources are never released.

For the sake of interpretability, we always set the simulation length τ = 168 hours (that is, one week), the average allocation request duration μ = 24 hours, and the corresponding standard deviation σ = 2 hours.

5.1.1. Test 1: Evaluating solvers scalability as the NI size increases

First, we verified that the approach is computationally effective enough to be embedded in TeNOR. We therefore considered two options: to use either the open source solver GNU GLPK or the commercial one IBM CPLEX to perform service mapping. In both cases, the solver is invoked to optimize the service mapping model described in Section 3.2.

A time limit of 60s was given to each solver run. The simulations were restricted to instances with up to 100 NI nodes. The parameters α, β and γ of the model were all set to 1.

The synthetic simulator was set in simulation mode. We analysed two scenarios: mild and high average NI load, setting in the former case $\lambda = \mu / (0.50 * (L / l))$, and in the latter case $\lambda = \mu / (0.75 * (L / l))$, where L is the overall available CPU resources in the NI, and l is the average amount of CPU resource required by each NS in the corresponding pool. That is, if CPU were the only scarce resource, and NS node fragmentation were possible, in the mild (respectively, high) average NI load scenario we would expect to have about 50% (respectively, 75%) of overall CPU resources always allocated.

We considered two performance measures: the percentage of accepted NS allocation requests, and the average computing time per allocation request.

Table 13 Solvers scalability test, 60s time limit

Solver	GLPK

Average load	Size	Acceptance rate	CPU time (accept)	Percentage Timeout (accept)	CPU time (reject)	Percentage Timeout (reject)
0,5	20	80,55%	0,02	0,00%	0,03	0,00%
	30	75,31%	0,21	0,19%	0,43	0,57%
	50	71,43%	0,49	0,20%	0,48	0,34%
	100	58,91%	2,69	0,78%	1,95	0,63%
0,75	20	67,52%	0,02	0,00%	0,02	0,00%
	30	62,12%	0,21	0,12%	0,25	0,25%
	50	57,26%	0,51	0,19%	0,32	0,08%
	100	47,57%	2,57	0,74%	1,87	0,54%
Avg.		**65,08%**	**0,84**	**0,28%**	**0,67**	**0,30%**
Solver		**CPLEX**				
Average load	**Size**	**Acceptance rate**	**CPU time (accept)**	**Percentage Timeout (accept)**	**CPU time (reject)**	**Percentage Timeout (reject)**
0,5	20	80,57%	0,02	0,00%	0,01	0,00%
	30	75,42%	0,04	0,00%	0,02	0,00%
	50	71,07%	0,09	0,00%	0,04	0,00%
	100	59,52%	0,31	0,00%	0,16	0,00%
0,75	20	67,73%	0,02	0,00%	0,01	0,00%
	30	62,70%	0,04	0,00%	0,02	0,00%
	50	58,75%	0,08	0,00%	0,04	0,00%
	100	47,07%	0,29	0,00%	0,15	0,00%
Avg.		**65,35%**	**0,11**	**0,00%**	**0,06**	**0,00%**

Table 13 contains two horizontal blocks, one for each NI load scenario (column *avg. load*). Each block contains one row for any instance class, reporting average values over instances of that class. Instance classes are sorted by increasing NI size (column *size*). The actual results are reported in the subsequent two vertical blocks, one for each solver option (as indicated in the leading row). Each of them is composed by two sub-blocks, restricting to results on allocation requests that were accepted (respectively, rejected). In each sub-block we reported, in turn, the average percentage of accepted (respectively, rejected) allocation requests, the average CPU time spent in mapping for those requests that were accepted (respectively, rejected), the percentage of those runs that exceeded the time limit.

We first observe that computing time is not a critical issue: the commercial solver CPLEX never exceeds the time limit, and the average response times is as low as 0.3s even for large NIs. The open source solver GLPK yields computing times that are one order of magnitude larger than those of CPLEX; this was expected, giving the benchmarking results from the literature. Still, the average query time is always less than 3s. Allocation rejection is almost always produced as the result of the solver detecting infeasibility (timeout is observed on average in 0.3% of the runs, and only for GLPK); rejections are on average faster to report than allocations.

Summarizing, both solvers scale well in terms of computing time: embedding either CPLEX or GLPK would likely yield systems whose bottleneck is not the service mapping optimization algorithm. Embedding CPLEX yields almost real-time performances.

Second, we observed that the percentage of accepted requests decreases as the size of the NI increases. This might highlight a possible point of improvement for the mapping model. At the same time, GLPK and CPLEX provide almost identical results, even if GLPK incurs more often (0.3% of the runs) in timeouts.

To further check the behaviour of the solvers when very fast response is required, we repeated the experiment by setting a time limit of 3s instead of 60s. Our results are reported in Table 14; for simplicity only the NS allocation requests acceptance rate is reported for both solvers; as a reference, we include also the acceptance rate obtained during the previous experiment. On average, no worsening is observed.

Table 14 Solvers scalability tests, overall acceptance rate.

Timeout		3 seconds		60 seconds	
Average load	Size	GLPK	CPLEX	GLPK	CPLEX
0,5	20	80,48%	80,25%	80,55%	80,57%
	30	74,92%	76,15%	75,31%	75,42%
	50	70,97%	71,30%	71,43%	71,07%
	100	60,16%	60,12%	58,91%	59,52%
0,75	20	67,81%	67,82%	67,52%	67,73%
	30	62,60%	62,52%	62,12%	62,70%
	50	57,41%	57,59%	57,26%	58,75%
	100	47,94%	47,76%	47,57%	47,07%
Avg.		**65,29%**	**65,44%**	**65,08%**	**65,35%**

Therefore, also in terms of acceptance rate, embedding either CPLEX or GLPK, even by imposing tight time limits, has no strong impact on the performances of the overall system.

5.1.2. Test 2: Evaluating solvers scalability as the average datacenter load increases

Second, we analysed the performances of our mapping algorithm as the average DC load changes. In particular, we restricted our tests to instances whose NIs contain 20 nodes, and considered simulations in which the average CPU load of NI nodes, denoted as δ, ranges from 0.1 to 1.2, considering each step of 0.1 points, and setting for each of them $\lambda = \mu/(\delta * (L / l))$.

Figure 15 below depicts the average acceptance rate obtained using CPLEX as the average CPU load changes.

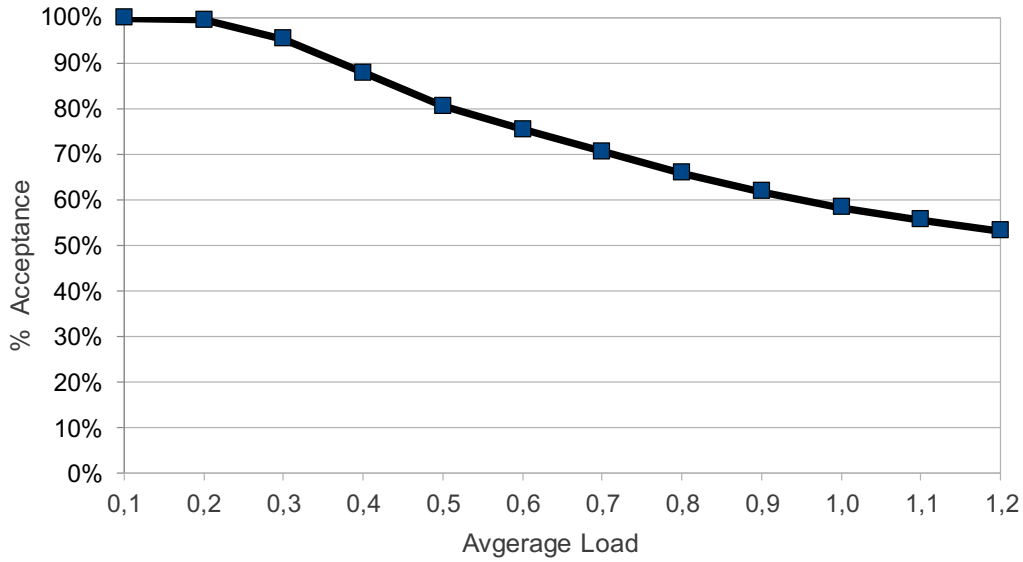

Figure 15 Percentage acceptance as average load increases.

Figure 16 has a similar structure and reports the average computing time per allocation request that was subsequently accepted (blue line) or rejected (orange line).

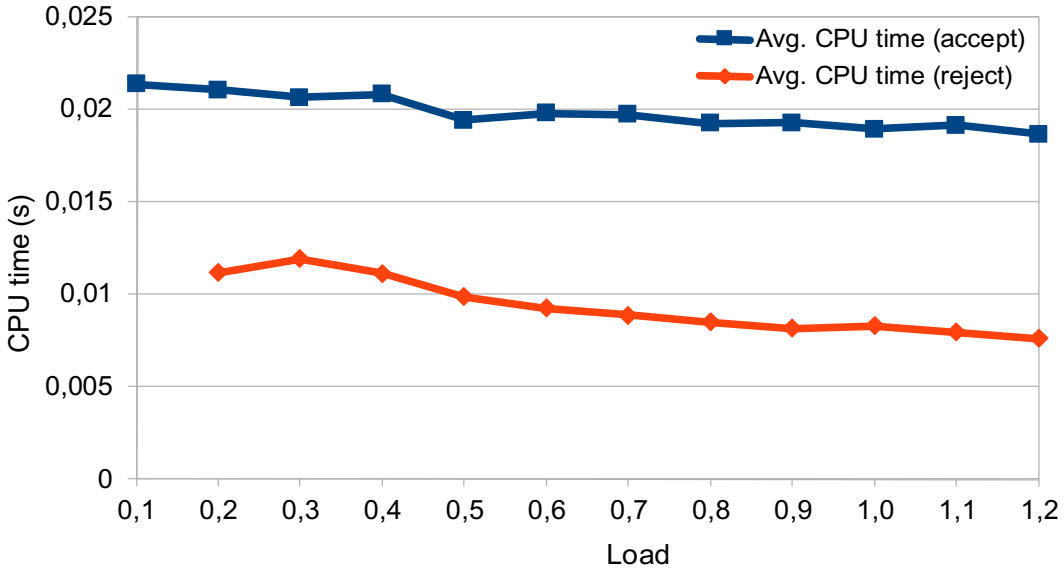

Figure 16 CPU time as average load increases

As expected, the acceptance rate drops as the average load increases, but the system does not collapse even under heavy stress (rightmost part of the charts). Average load seems to have very little effect on solver computing time.

No substantial difference was observed by using GLPK, and therefore the corresponding results are omitted.

5.1.3. Test 3: Fine tuning of model parameters

Finally, we tried to assess the behaviour of the service mapping algorithm as the parameters of the corresponding model change. To this purpose, we set the synthetic simulator in stress test mode, and we considered only instances whose NIs contain 20 nodes. These always

include 40 NSs. We considered eight configurations, one for each possible choice of either value 1.0 or value 0.0 for each of the model parameters α, β and γ. Figure 17 depicts the acceptance rate (vertical axis) with respect to the NS allocation request arrival rank (horizontal axis) obtained using CPLEX. One line is included for each configuration: thick lines encode the choice α = 1.0 (while thin ones encode α = 0.0), black lines encode β = 1.0 (while grey ones encode β = 0.0), continuous lines encode γ = 1.0 (dashed ones encode γ = 0.0). Before rank 20 the acceptance rate was always 100%, and therefore the corresponding portion of the Chart is omitted.

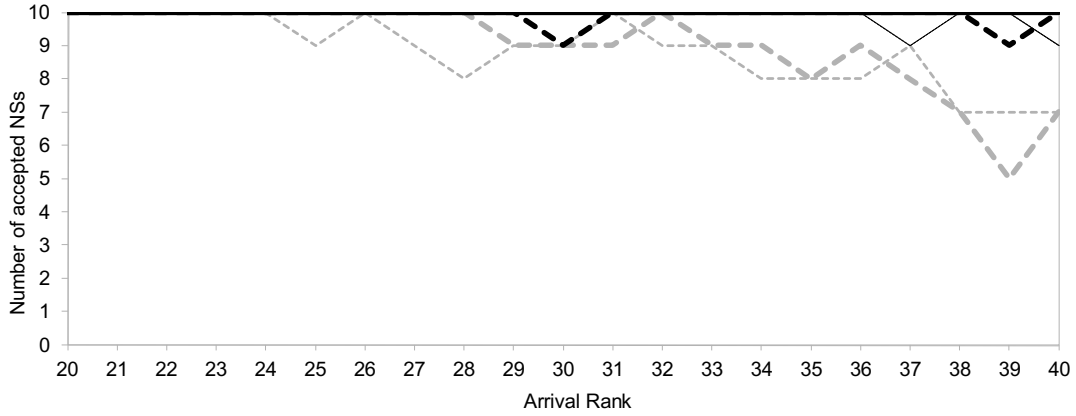

Figure 17 Acceptance rate in function of the arrival rank

We note that moving from β = 0.0 (grey lines) to β = 1.0 (black lines) substantially improves the acceptance rate; the same applies moving from γ = 0.0 (dashed lines) to γ = 1.0 (continuous lines). Moving from α = 0.0 (thin lines) to α = 1.0 (thick lines) has still some impact, when combined with settings β = 1.0 and γ = 1.0. This matches our modelling aim, in which objective terms related to β and γ directly affect allocation feasibility, while that related to α is useful only for diversification, and therefore inter PoP balancing. Overall, setting all parameters to 1.0 gives the best results.

Figure 18 depicts the average computing time per allocation request (vertical axis) with respect to the NS allocation request arrival rank (horizontal axis). One line is included for each configuration, respecting the encoding described above. While up to request 20 the allocation sub problems turn out to be trivial for the solver, as even a simple greedy solution may be optimal, from request 21 onwards solving the sub problem becomes not trivial and requires an additional effort, even if the solver response time remains low.

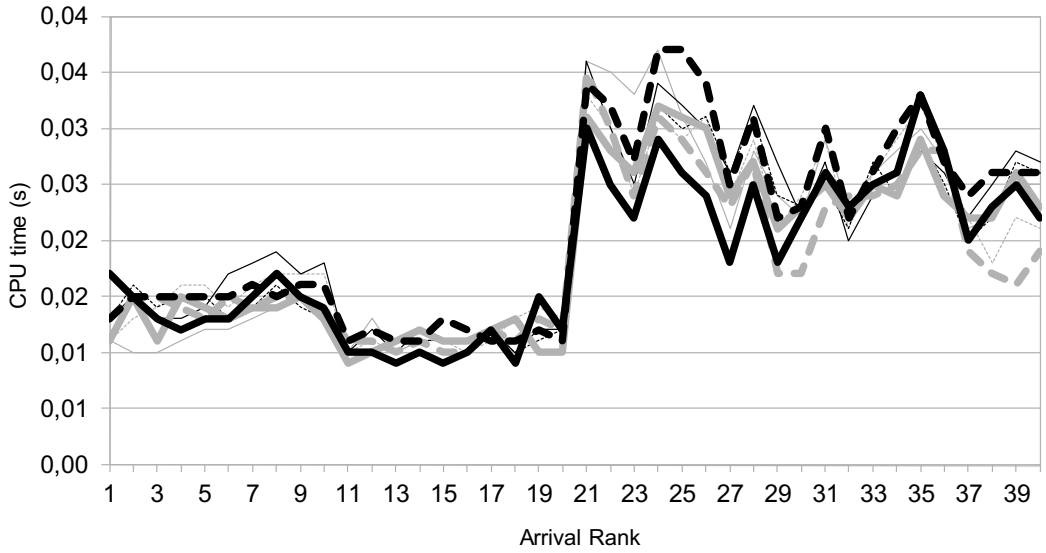

Figure 18 Average computing time in function of the arrival rank (on stress test)

No configuration yields substantial changes in the average solver computing time. No substantial difference was observed by using GLPK, and therefore the corresponding results are omitted.

5.2. Reinforcement Learning Based Approach

In the following subsection simulation tests for the two RL mapping algorithms proposed are provided (i.e. for the VNF mapping strategy in Section 3.2.1 and the complete NS mapping Strategy in Section 3.2.2). The approach has been implemented in Matlab, with artificial infrastructure and service repositories created to build the simulation scenarios.

5.2.1. Mapping of Single VNFs

The following simulation scenario is proposed to highlight the peculiarities of the approach. Three types of NSs are considered: "light", "medium" and "heavy", in terms of resource requirements. NSs are characterized by the same arrival ($\lambda^a = 0{,}02$) and termination ($\lambda^t = 0{,}0001$) Poissonian rates. NSs are composed each by a single VNF. VNFs can require up to five different types of resources. A network with 10 PoPs is considered. Infrastructure and NSs' resource specifications have been chosen based on [41] (PoP resources in the order of $10^4 - 10^5$ and NS cumulative resources of 400, 500 and 600 respectively for the three NS types). The rewards for mapping the first two NSs are set to 1, the reward for mapping the "heavy" NSs is set to 10. The remaining simulation parameters are $\eta = 0.9, \gamma = 0.7, \alpha_\tau = 1/(1.01 + \lfloor \tau/1000 \rfloor)$. This simple simulation setting is aimed at showing how RL manages to learn environment dynamics and maximize the allocation reward, in respect of infrastructure and NS constraints. Figure 26 shows the total reward achieved in time ($3{\times}10^5$ simulation steps) by two different policies: a greedy policy and the proposed Q-Learning strategy.

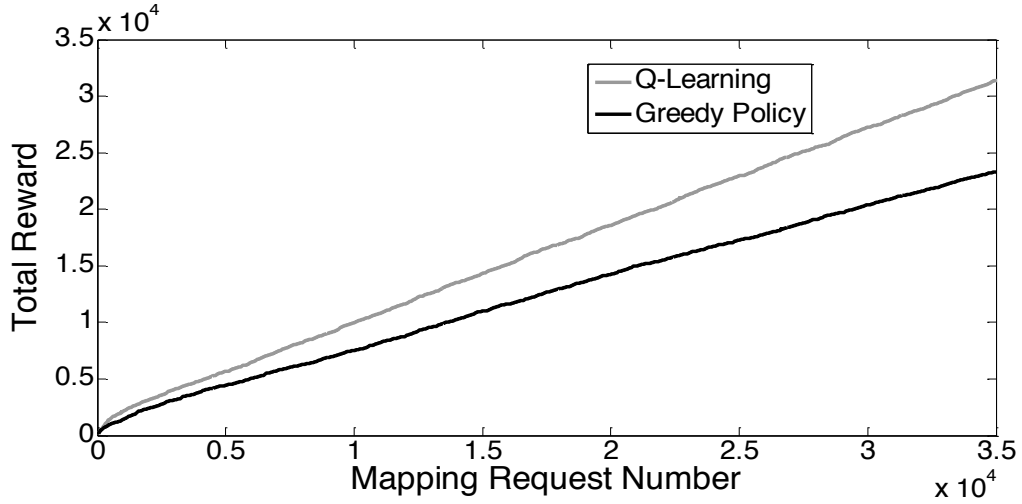

Figure 19 Revenue evolution: greedy and Q-Learning policies compared

The greedy policy is a typical policy used for benchmark purposes in RL applications. According to the greedy policy, at each time τ, the action is chosen which maximizes the *current* achievable reward r_τ, compatibly with the PoPs' level of occupancy. The figure shows that the adopted Q-Learning policy largely overcomes in time the greedy one. The Q-Learning strategy ensures higher performances by learning system's dynamics (e.g. arrival and termination rates, PoP resources availabilities and NS requirements). Also, it acts having as objective the performance of the system over a time horizon, rather than by looking only at the immediate reward (thus implicitly implementing the "lookahead" property [16]). In this sense, allocation of NSs is managed by the controller in such a way as to give some precedence to the most rewarding services, as it can be seen in the next Figure 27. The figure shows the performance of the two policies in terms of average acceptance rate. For both policies, as natural, the acceptance rate decreases as the NS size increases.

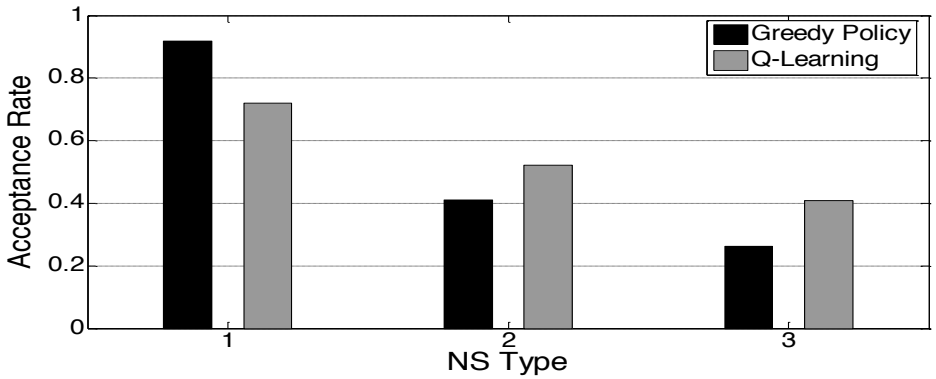

Figure 20 Average NS acceptance rates

However, it can be noticed that, under the Q-Learning policy, the acceptance rate of the "large" NS (the one associated with the greatest reward) significantly increases (+56%), while the acceptance rate of the "small-size" NS decreases. Notice that this is done not by explicitly rejecting the less rewarding services (even though explicit rejection could be modelled as well), but rather by mapping them to sub-optimal PoPs, in order to keep available space for the rewarding NSs. As a result, also the overall acceptance rate increases (+4%), resulting into higher utilization of PoP resources[11] (Figure 28).

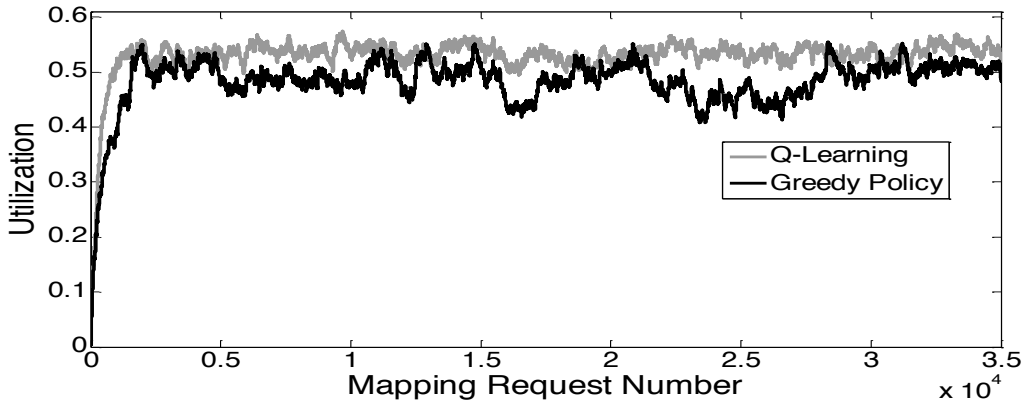

Figure 21 Utilization of PoP resources

The simulation scenario above corresponds to a challenging load factor[12] of 1.05. Furthermore, the fact that the different NSs have different resource requirements poses significant

[11] Utilization is defined here as the ratio of the average used PoP resources over the total available PoP resources.

[12] Recall that load factor is defined here as the ratio of the average requested resources to the available ones.

constraint to the mapping process (making impossible in practice to achieve high utilization factors).

Concluding, the following strengths can be mentioned regarding the proposed RL approach: (i) the behavior of the algorithm can be tuned in accordance with the control objectives by choosing the right reward function, (ii) the mapping algorithm works by simply updating the equations (9) and (10), thus having negligible execution times, (iii) the learning property of the RL algorithm makes possible to maximize not just the immediate reward, but rather the expected reward in the long run, which is important when acting in an uncertain environment (e.g. stochastic service request times and types). In particular, points (ii) and (iii) are distinctive features with respect to the proposed ILP approaches.

5.2.2. Mapping of Complete NSs

We finally provide a simple simulation test of the RL algorithm designed to map NSs composed by more than one VNF (i.e. the most general case – refer to Section 3.2.2). To this end, a scenario is considered in which requests randomly arrive according to a Poisson distribution. At each request time, a NS is randomly picked from a pool of three possible ones. Topology and node/link resource requirements of the three NSs are reported in the next figure.

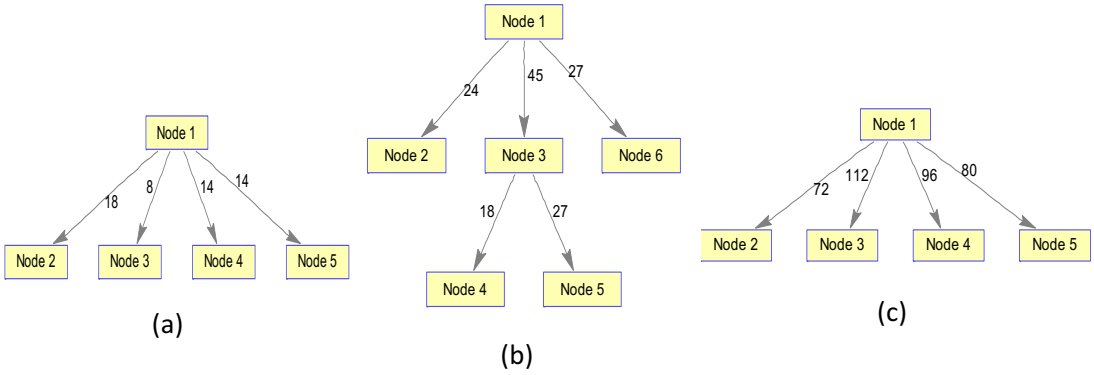

Figure 22 NSs topologies and bandwidth requirements for the NS1 (a), the NS2 (b) and the NS (c)

The algorithm is tested on the 20 node NI from [41], the same used previously in Section 5.1.2. The network graph is reported in the next figure.

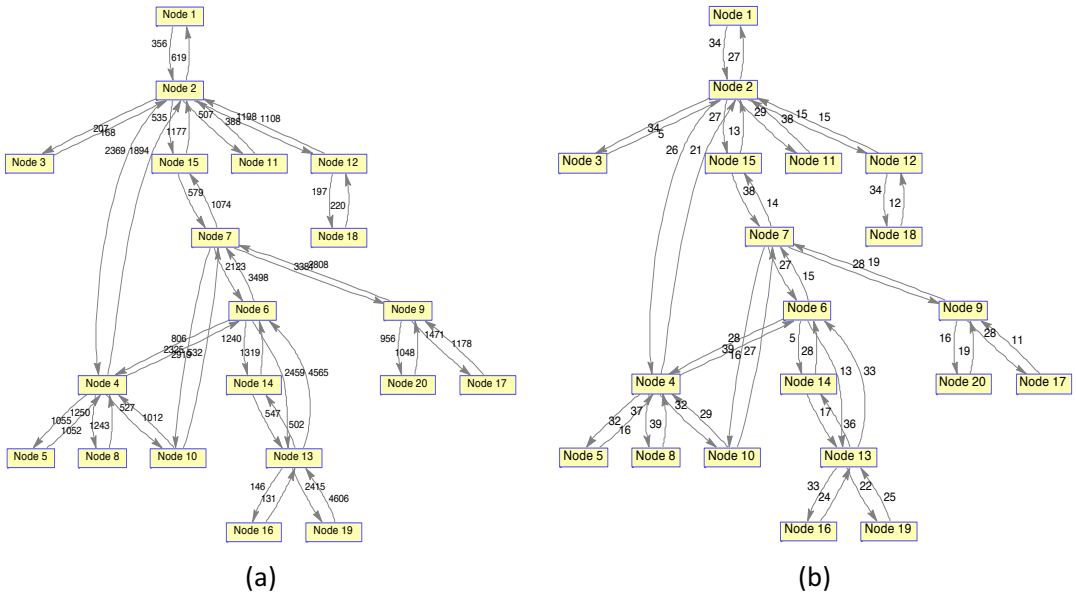

Figure 23 NI topology (20 nodes) and bandwidth availability (a), link delay (b)

The figure below reports the resulting acceptance rate as a function of the load of requests, defined as the ratio between the average cumulative resources asked to the total available resources. We refer to the PoP load. To compute the average cumulative resources asked, we refer to the Little law [42], according to which the average number of NSs outstanding requests is equal to the average arrival rate times the average dwelling time (which is in turn given by the ratio of the arrival rate to the departure rate, for Poisson distributions). The figure reports the acceptance rate for the three NSs. The reward function in this test is set to maximise NSs acceptance. As for the previous VNF case, the rates are decreasing and the heaviest NS is associated with the lowest acceptance rate.

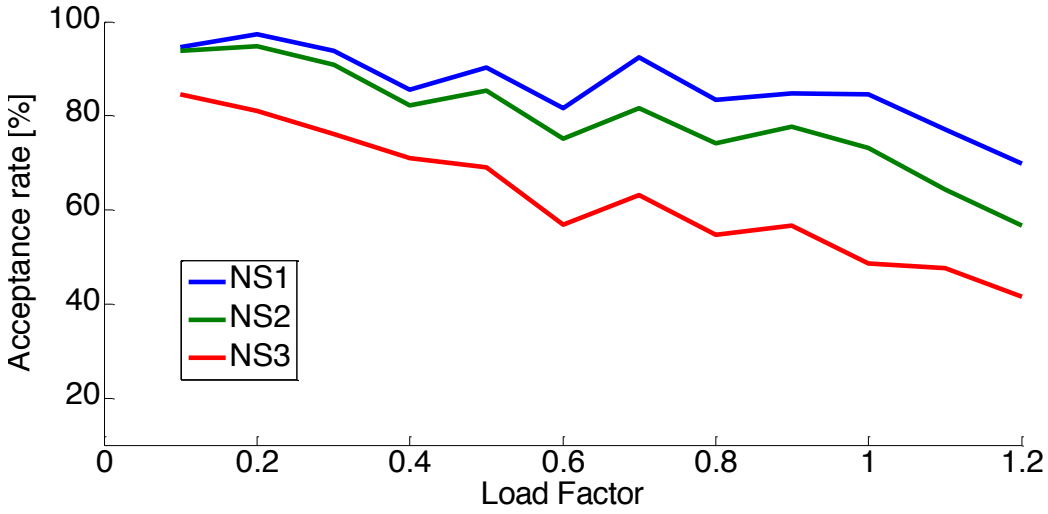

Figure 24. Acceptance rate in function of the load (same reward for all NS types)

Concluding, the above simulations have shown the applicability of the reinforcement learning concept to the service mapping problem, with encouraging results which show that learning systems' dynamics allows to effectively implement a form of lookahead property[13] different from the one previously discussed in literature [16], which consisted "simply" in mapping together more than one request.

Future research work will regard in particular:

- Investigation of new definitions for the state space, in order to: (i) improve scalability through improved state aggregation, (ii) improve the details on the actual state of the infrastructure conveyed by the state definition.
- Investigate new definitions for the reward function, in line with the goals of the actors involved.
- Investigate possible combinations with the ILP approaches proposed in this deliverable.

[13] That is, the ability of a mapping algorithm to map more than one request at the same time.

5.3. Multi-stage Network Service Embedding

In this section, we evaluate the efficiency of the proposed two-stage service chain embedding approach. We particularly focus on the ILP-based weight-minimized service chain request partitioning for which we employ the IBM ILOG CPLEX optimizer. In addition, we use a greedy algorithm as baseline. This algorithm binds each NF with one of the end-points, depending on the NF location constraint or the order in the service chain (for NFs without location dependencies), and assigns each NF to the DC which is most proximate to the corresponding end-point. We run simulations with 50 homogeneous DCs, each containing 200 servers. We generated service chains with each 10 to 20 NFs and a source traffic rate of 10 to 100 Mbps.

Figure 25 depicts the evolution of load balancing level across the DCs. Since the greedy selection of DCs close to the end-points does not lead to load balancing[14], we focus on the load balancing level of the ILP variant. According to Figure 25, weight-minimized request partitioning converges to near-optimal load balancing after 100K requests, exploiting the DC utilization levels disclosed via the link weights.
More specifically, after 250K requests the highest server load is 5.3% above the average DC utilization for weight-minimized request partitioning.

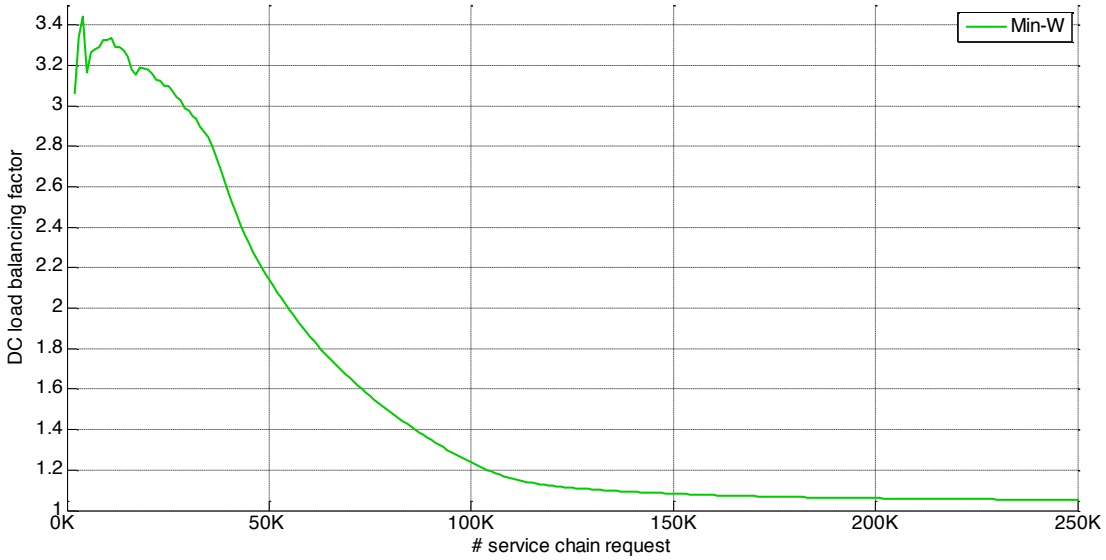

Figure 25: Load balancing level across the DC servers with weight-minimized request partitioning

Figure 26 shows the request acceptance rates for the weight-minimized and for the greedy request partitioning methods. Optimizing DC selection based on the disclosed weights inhibits the assignment of NF-subgraphs to highly utilized DCs, which usually leads to request rejections. As such, weight-minimized request partitioning yields a higher request acceptance rate while the greedy algorithm suffers from a large number of rejections, due to the restrictions in DC selection.

[14] The DC load balancing factor is defined by the maximum over the average server utilization.

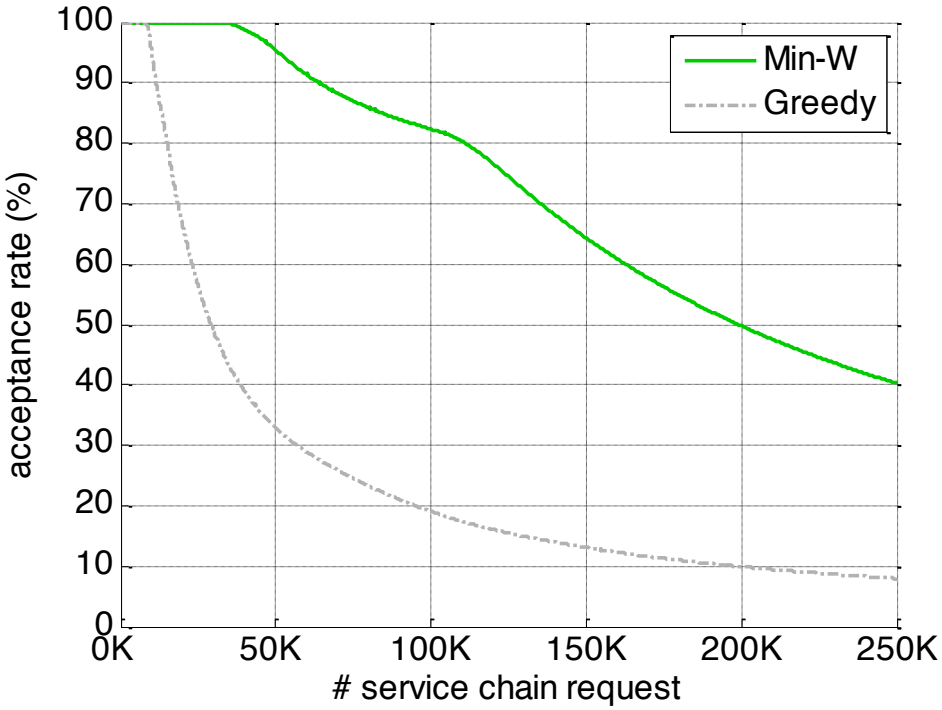

Figure 26: Acceptance rate with weight-minimized and greedy request partitioning

Figure 26 and Figure 27 show a strong correlation between the acceptance rate and generated revenue. The ILP variant generates substantially higher revenue from CPU and bandwidth, compared to the greedy algorithm. The high acceptance rate with weight-minimized request partitioning translates to higher revenue which is up to three times higher than with the greedy variant. This essentially designates weight-minimized request partitioning as the preferred method.

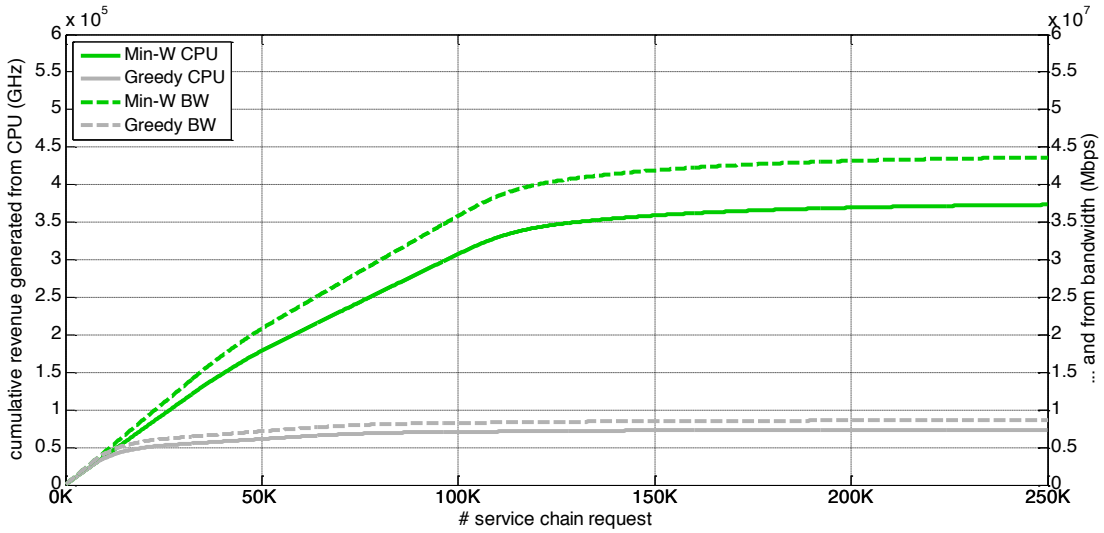

Figure 27: Cumulative revenue with weight-minimized and greedy request partitioning

Finally, we measure the acceptance rate of weight-minimized request partitioning with 250K expiring requests and diverse arrival rates. Figure 28 shows that acceptance rates always converge to a steady state. This further indicates the efficiency of the proposed ILP-based methods for service chain embedding across multiple DCs.

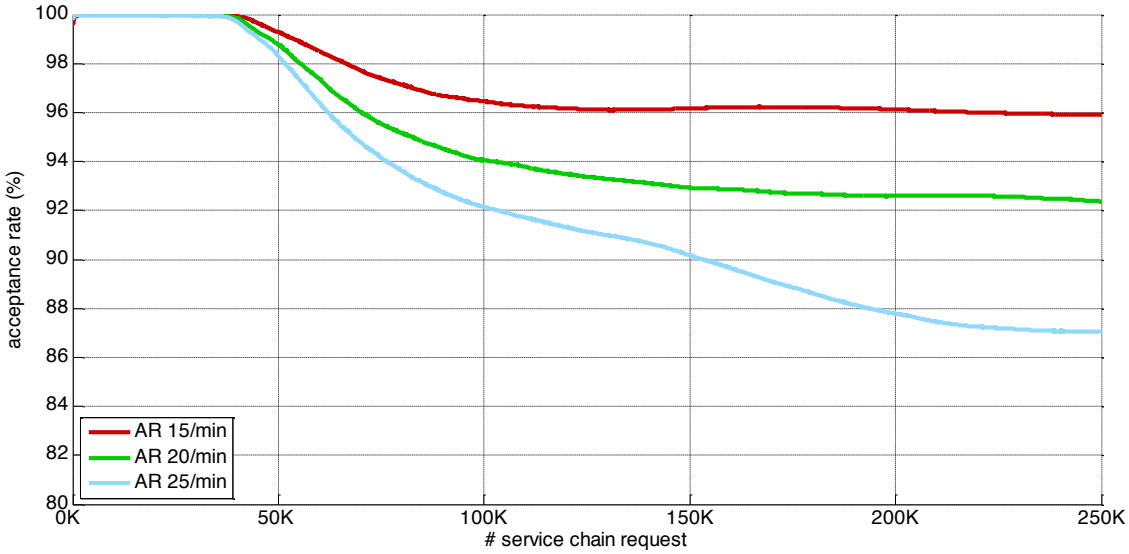

Figure 28: Acceptance rate with diverse arrival rates of expiring service chain requests with weight

6. CONCLUSIONS

This deliverable has reported the outcomes of the Task 3.3 “Service Mapping” of the T-NOVA project.

The service mapping problem has been put into the context of the architecture developed in T-NOVA for virtualized network function orchestration, and a common modelling framework for the first level mapping problem (i.e. mapping of virtual network functions to data centres) and the second level mapping problem (i.e. assignment of virtual machines to hardware devices inside the data centre) has been proposed, based on graph theory. Different mathematical methodologies for service mapping have been explored, mainly borrowing from pure mathematical optimization theory (integer linear optimization) and from stochastic control theory (reinforcement learning). The properties of such proposed mapping algorithms have been discussed by means of simulations carried out in emulated environment.

A second key outcome of the Task 3.3, reported in this deliverable, is the design and the integration of the service mapping module (the module actually hosting the service mapping algorithm) in the T-NOVA system. A key outcome of the integration design is that the mapping module can potentially integrate different mapping algorithms, provided that they comply to the documented interfaces specifications. Such interfaces mainly regard the interaction of the module with the infrastructure repository and with the service catalogue, which are the two key subsystems providing inputs to the mapping problem. The integer linear programming approach proposed by Unimi has been integrated in TeNOR, the T-NOVA orchestrator. The integration has been first tested on a mock-up system, built based on the interfaces specification, and then integrated into the actual T-NOVA system.

The integration and implementation style adopted and the results of algorithms research open the way for further promising works. As mentioned, the so built platform allows to easily integrate advancements in the mapping algorithms, which are possible especially in the direction of integrating the theoretical approaches presented (exact optimization versus stochastic control) towards an enhanced algorithm. Such enhanced algorithm could capture the strengths of both the theoretical approaches, that is, (i) the ability of exact methods to accurately tackle the service and infrastructure requirements, thanks to the inclusion of related constraints, and, (ii) the ability of the stochastic methods to learn environment dynamics in order to came up with mapping decisions that account for the behaviour of the system in the long run.

7. REFERENCES

[1] T-NOVA Consortium, "Deliverable D2.1 System Use Cases and Requirements." pp. 1–62, 2014.

[2] European Telecommunications Standards Institute (ETSI), "GS NFV-MAN 001 - V1.1.1 - Network Functions Virtualisation (NFV); Management and Orchestration," vol. 1, pp. 1–184, 2014.

[3] T-NOVA Consortium, "Deliverable D3.01 Interim Report on Orchestrator Platform Implementation." pp. 1–92, 2014.

[4] European Telecommunications Standards Institute (ETSI), "Network Functions Virtualisation (NFV); Architectural Framework," vol. 1, pp. 1–21, 2013.

[5] European Telecommunications Standards Institute (ETSI), "Network Functions Virtualisation (NFV);Terminology for Main Concepts in NFV," vol. 1, no. ETSI GS NFV 003 V1.2.1, pp. 1–13, 2014.

[6] European Telecommunications Standards Institute (ETSI), "Network Functions Virtualisation (NFV); Use Cases," *ETSI GS NFV 001 V1.1.1*, pp. 1–50, 2013.

[7] H. Ballani, P. Costa, T. Karagiannis, and A. Rowstron, "Towards predictable datacenter networks," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 41, no. 4, p. 242, 2011.

[8] J. Lee, M. Lee, L. Popa, Y. Turner, S. Banerjee, P. Sharma, B. Stephenson, J. C. Mogul, J. Mudigonda, J. R. Santos, H. P. Labs, and P. Alto, "CloudMirror : Application-Aware Bandwidth Reservations in the Cloud," in *Proceedings of the 5th USENIX Workshop on Hot Topics in Cloud Computing*, 2013, pp. 1–6.

[9] C. Guo, G. Lu, H. J. H. J. Wang, S. Yang, C. Kong, P. Sun, W. Wu, and Y. Zhang, "SecondNet: a data center network virtualization architecture with bandwidth guarantees," in *Proceedings of the 6th International Conference*, 2010, no. Vdc, pp. 15:1–15:12.

[10] A. Gember, R. Grandl, and A. Anand, "Stratos: Virtual middleboxes as first-class entities," *ONS summit*, 2013.

[11] S. Oechsner and A. Ripke, "Flexible Support of VNF Placement Functions in OpenStack," in *Proceedings of the 2015 1st IEEE Conference on Network Softwarization (NetSoft)*, 2015, pp. 1–6.

[12] A. Fischer, J. F. Botero, M. T. Beck, H. De Meer, and X. Hesselbach, "Virtual network embedding: A survey," *IEEE Commun. Surv. Tutorials*, vol. 15, no. 4, pp. 1888–1906, 2013.

[13] H. Xin, S. Ganapathy, and T. Wolf, "A scalable distributed routing protocol for networks with data-path services," in *Proceedings - International Conference on Network Protocols, ICNP*, 2008, pp. 318–327.

[14] A. Abujoda and P. Papadimitriou, "MIDAS: Middlebox discovery and selection for on-path flow processing," in *2015 7th International Conference on Communication Systems and Networks (COMSNETS)*, 2015, pp. 1–8.

[15] R. Riggio, T. Rasheed, and R. Narayanan, "Virtual Network Functions Orchestration in Enterprise WLANs," in *Integrated Network Management (IM), 2015 IFIP/IEEE International Symposium on*, 2015, pp. 1220–1225.

[16] 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 *Proceedings of the Network Operations and Management Symposium (NOMS)*, 2014, pp. 1–7.

[17] S. Sahhaf, W. Tavernier, D. Colle, and M. Pickavet, "Network service chaining with efficient network function mapping based on service decompositions," in *Proceedings of the 2015 1st IEEE Conference on Network Softwarization (NetSoft)*, 2015.

[18] S. Sahhaf, W. Tavernier, D. Colle, and M. Pickavet, "Network service chaining with efficient network function mapping based on service decompositions," *Comput. Networks*, vol. 93, pp. 492–505, 2015.

[19] S. Mehraghdam, M. Keller, and H. Karl, "Specifying and Placing Chains of Virtual Network Functions," *arXiv Prepr. arXiv1406.1058*, vol. 22, no. 5, pp. 20–25, 2014.

[20] V. Abedifar, M. Eshghi, S. Mirjalili, and S. M. Mirjalili, "An optimized virtual network mapping using PSO in cloud computing," in *2013 21st Iranian Conference on Electrical Engineering (ICEE)*, 2013, pp. 1–6.

[21] K. Giotis, Y. Kryftis, and V. Maglaris, "Policy-based orchestration of NFV services in Software-Defined Networks," in *Proceedings of the 2015 1st IEEE Conference on Network Softwarization (NetSoft)*, 2015, pp. 1–5.

[22] T-NOVA Consortium, "T-NOVA Document of Work." pp. 1–164, 2013.

[23] R. Mijumbi, J. Serrat, J. L. Gorricho, N. Bouten, F. De Turck, and R. Boutaba, "Network Function Virtualization: State-of-the-art and Research Challenges," *IEEE Commun. Surv. Tutorials*, no. c, pp. 1–28, 2015.

[24] D. Dietrich, A. Rizk, and P. Papadimitriou, "Multi-domain virtual network embedding with limited information disclosure," *IEEE Trans. Netw. Serv. Manag.*, vol. 12, no. 2, pp. 188–201, 2015.

[25] J. F. Riera, X. Hesselbach, E. Escalona, J. A. García-espín, and E. Grasa, "On the Complex Scheduling Formulation of Virtual Network Functions over Optical Networks," in *16th International Conference on Transparent Optical Networks (ICTON '14)*, 2014, pp. 1–5.

[26] S. Battilotti, F. Facchinei, A. Giuseppi, G. Oddi, A. Pietrabissa, and V. Suraci, "Resource Management in Multi-Cloud Scenarios via Reinforcement," in *Control Conference (CCC), 2015 34th Chinese*, 2015, pp. 9084 – 9089.

[27] J. Riera, J. Batallé, J. Bonnet, M. Días, M. J. McGrath, G. Petralia, F. Liberati, A. Giuseppi, A. Pietrabissa, A. Ceselli, A. Petrini, M. Trubian, P. Papadimitrou, D. Dietrich, A. Ramos, M. Javier, G. Xilouris, A. Kourtis, T. Kourtis, and M. Evangelos, "TeNOR - A Resource and Service Mapping Approach to Orchestrating VNF Deployments," in *Submitted to the Proceedings of the 2016 2nd IEEE Conference on Network Softwarization (NetSoft)*, 2016.

[28] Openstack, "Scheduling," 2015. [Online]. Available: http://docs.openstack.org/kilo/config-reference/content/section_compute-scheduler.html.

[29] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, vol. 3, no. 9. Cambridge MA: The MIT Press, 2012.

[30] M. L. Puterman, *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014.

[31] D. Dietrich, A. Abujoda, and P. Papadimitriou, “Embedding Network Services across Multiple Providers with Nestor,” in *Proc. IFIP Networking*, 2015.

[32] A. Farrel and J. P. Vasseur, “RFC 4655,” *Netw. Work. Gr.*, no. 1, pp. 1–5, 2006.

[33] J. Batallé, J. Ferrer-Riera, E. Escalona, and J. A. García-Espín, “On the implementation of NFV over an OpenFlow infrastructure : Routing Function Virtualization,” in *Future Networks and Services (SDN4FNS), 2013 IEEE SDN*, 2013, pp. 1–6.

[34] J. F. Riera, E. Escalona, J. Batalle, E. Grasa, and J. A. Garcia-Espin, “Virtual network function scheduling: Concept and challenges,” in *2014 International Conference on Smart Communications in Network Technologies (SaCoNeT)*, 2014, pp. 1–5.

[35] R. Mijumbi, J. Serrat, J. Gorricho, N. Bouten, F. De Turck, and S. Davy, “Design and Evaluation of Algorithms for Mapping and Scheduling of Virtual Network Functions,” in *Proceedings of the 2015 1st IEEE Conference on Network Softwarization (NetSoft)*, 2015.

[36] “Ruby Programming Language,” 2015. [Online]. Available: https://www.ruby-lang.org/. [Accessed: 15-Dec-2015].

[37] T-NOVA Consortium, “Deliverable 3.2 Infrastructure Resource Repository.” pp. 1–105, 2015.

[38] T-NOVA Consortium, “Deliverable 6.4 SLAs and Billing.” 2015.

[39] T-NOVA Consortium, “Deliverable D6.1 Service Description Framework.” 2015.

[40] Algorithms and complexity group, “Virtual Network Mapping Problem Instances,” 2015. [Online]. Available: https://www.ac.tuwien.ac.at/research/problem-instances/. [Accessed: 15-Dec-2015].

[41] J. Zhu, “Benchmarking Virtual Network Mapping Algorithms,” University of Massachusetts - Amherst, 2014.

[42] J. D. C. Little and S. C. Graves, “Chapter 5 Little ’ s Law,” *Oper. Manag.*, vol. 115, pp. 81–100, 2008.

8. LIST OF ACRONYMS

Acronym	Explanation
API	Application Programming Interface
CIMI	Cloud Infrastructure Management Interface
DAG	Directed Acyclic Graph
DC	Datacenter
DMI	Desktop Management Interface
DMTF	Distributed Management Task Force
DPDK	Data Plane Development Kit
EPA	Enhanced Platform Awareness
ETSI	European Telecommunications Standards Institute
GPU	Graphics Processing Unit
GW	Gateway
HDFS	Highly Distributed File System
HTTP	Hyper-Text Transfer Protocol
ILP	Integer Linear Programming
IPMI	Intelligent Platform Management Interface
IVM	Infrastructure Virtualisation and Management
JSON	JavaScript Object Notation
MANO	(ETSI NFV) Management and Orchestration
MDP	Markov Decision Problem
MIF	Management Information Format
MIP	Mixed Integer Programming
ML	Modular Layer
MPL	Mathematical Programming Language
NAT	Network Address Translator
NF	Network Function
NFS, NF Store	Network Function Store
NFV	Network Functions Virtualization
NI	Network Infrastructure
NIC	Network Interface Controller
NS	Network Service
NSD	Network Service Descriptor

Or-Vi	Interface between the Orchestrator and the VIM
PoP	Point of Presence
QoS	Quality of Service
RCSP	Resource Constrained Project Scheduling Problem
REST	Representational State Transfer
SLA	Service Level Agreement
SM	Service mapping
SMM	Service mapping Module
SP	Service Provider
SR-IOV	Single Root I/O Virtualization
T-Ac-Or	Interface between T-NOVA Accounting (Marketplace module) and the Orchestrator
T-Br-Or	Interface between T-NOVA Brokerage (Marketplace module) and the Orchestrator
T-Da-Or	Interface between T-NOVA Dashboard (Marketplace module) and the Orchestrator
T-Sla-Or	Interface between T-NOVA Servile Level Agreement (Marketplace module) and the Orchestrator
UC	Use Case
VCPU	Virtual CPU
VDU	Virtualisation Deployment Unit
VIM	Virtual Infrastructure Manager
VM	Virtual Machine
VN	Virtual Network
VNE	Virtual Network Embedding
VNF	Virtual Network Function
VNFc	Virtual Network Function component
VNFD	Virtual Network Function Descriptor
VNFM	Virtual Network Function Manager
Vnfm-Vnf	Interface between the VNF Manager and VNFs
vNIC	virtual Network Interface Controller

9. LIST OF MATHEMATICAL SYMBOLS

Symbol	Definition/Explanation
A	Set of links of the graph $G(NS)$, modelling the set of links among the VNFs composing the NS.
A^D	Set of links of the graph $G(DC)$, modelling the connections among the PoP apparatuses.
A^F	Set of links of the graph $G(VNF)$, modelling the set of links among the VNFc composing the VNF.
A^I	Set of links of the graph $G(NI)$, modelling the set of links among the PoPs composing the NI.
α	Weighting parameter of the target function of the proposed ILP mapping approach.
Act	Space of the possible mapping actions (reinforcement learning approach)
β	Weighting parameter of the target function of the proposed ILP mapping approach.
c_p^h	Cost of assigning the VNF h to the PoP p
δ_{pq}	Delay associated with each arch (p, q) in $G(NI)$.
Δ_π	Maximum allowed delay associated to each path $\pi \in P$.
f_{pq}^{hk}	Amount of flow over the link (p,q) for the NF-graph edge (h,k).
γ	Weighting parameter of the target function of the proposed ILP mapping approach.
$G(DC) = (V^D, A^D)$	Graph modelling each PoP infrastructure.
$G(NI) = (V^I, A^I)$	Graph modelling the Network Infrastructure.
$G(NS) = (V, A)$	Graph modelling the Network Service.
$G(VNF) = (V^F, A^F)$	Graph modelling the Virtual Network Function.
λ	Learning rate of the Q-Learning update rule.
NT	Set of resource types available at Network Infrastructure nodes.
P	The set of paths connecting pairs of VNFs in a NS.
$\pi \in P$	Generic path in P (i.e. sequence of arcs in the graph $G(NS)$).
Π	Policy function providing a mapping of states to actions (i.e. $\Pi(s,a)$ denotes the probability that action a is taken when the system is in state s.
$Q_\pi(s, a)$	State-action value function. Estimate of the value associated to the state-action couple (s, a) (i.e. expected future reward achieved when starting from state s, taking action a and following policy Π).

r	Reward function associated with the Markov decision process defined for the SM approach based on reinforcement learning.
RA_p^t	Amount of resources of type $t \in NT$ available at PoP node $p \in V^I$.
RA_{pq}^t	Amount of resources of type $t \in NT$ available at Network Infrastructure link $(p,q) \in A^I$.
RR_h^t	Amount of resources of type $t \in NT$ required by VNF node $h \in V$.
RR_{hk}^t	Amount of resources of type $t \in NT$ required by VNF link $(h,k) \in A$.
S	State space of the Markov decision process defined for the SM approach based on reinforcement learning.
T	State transition matrix characterizing the Markov decision process defined for the SM approach based on reinforcement learning.
T	Time horizon over which the reinforcement learning problem is defined.
$t(s,a,s') \in T$	Probability to go to state s' when starting from state s and performing action a
V	Set of vertices of the graph $G(NS)$, modelling the set of VNFs composing the NS.
V^D	Set of vertices of the graph $G(DC)$, modelling the connections among the apparatuses of the PoP.
V^F	Set of vertices of the graph $G(VNF)$, modelling the set of VNFc composing the VNF.
V^I	Set of vertices of the graph $G(NI)$, modelling the set of PoPs in the Network Infrastructure.
$V_\Pi(s)$	Value function. Estimate of the value associated to state s (i.e. expected future reward achieved when starting from state s and following policy Π)
w_{pq}	Weight value associated with link (p,q).
x_{pq}^{hk}	Binary decision variable. $x_{pq}^{hk} = 1$ if the link (h,k) in graph $G(NS)$ is mapped on a path in the infrastructure which passes through the link (p,q) of graph $G(NI)$.
y_p^h	Binary decision variable. $y_p^h = 1$ if VNF h is assigned to PoP p.
z_p	Binary decision variable. $z_p = 1$ if server *p* is used.

Annexes

10. ANNEX A: DOCUMENTATION OF THE SERVICE MAPPING MODULE API

This annex reports a documentation of the REST API of the developed service mapping module. The API provides to the orchestrator the access to the service mapping module's services.

The information provided may be subject to future changes as the integration, deployment and testing work is refined during the course of the project.

Supported HTTP Method

POST

Current Accepted Request Parameters

The accepted parameters are reported in the table below.

Table 15 Service mapping request parameters (body of the POST call)

Name	Description	Example
NS_id	Id of the Network Service to be instantiated	"NS_id" = "demo1"
NS_sla	Flavour of the Network Service to be instantiated	"NS_sla" = "gold"
tenor_api	Address of the Tenor API	"tenor_api" = "http://1.2.3.4:5544"
infr_repo_api	Address of the Infrastructure Repository	"infr_repo_api" = "http://1.2.3.4:5544"
ir_simulation	If true, dummy IR data is used for testing purpose	"ir_simulation" = "false"
ns_simulation	If true, dummy NS data is used for testing purpose	"ns_simulation" = "false"

alpha	Optional parameter to be passed to the solutor: weight of alpha parameter in objective function (see notes)	"Alpha" = "0.6"
beta	Optional parameter to be passed to the solutor: weight of beta parameter in objective function (see notes)	"Beta" = "0.25"
gamma	Optional parameter to be passed to the solutor: weight of gamma parameter in objective function (see notes)	"Gamma" = "0.15"
fixVnf01	Optional parameter that forces the allocation of the first VNF of the NS to the PoP specified by the "toPoP01" parameter (for testing purpose)	"fixVnf01" = "vnf_id_a012fe31"
toPoP01	Optional parameter that states in which PoP should be allocated the VNF specified by the "fixVnf01" parameters (for testing purpose)	"toPoP01" = "126594f3a2a1"
solver	Chooses the solver application	"solver" = "unimi"

Notes on Alpha, Beta and Gamma parameters: these optional key/value pairs are passed by the Orchestrator to the UniMi solver and they are used as weights for the three components of the objective function. By altering the default values, the solver tries to optimize the allocation of the VNF so that cost is minimized (alpha > gamma + beta), the aggregate delay is minimized (beta > alpha + gamma) or the number of infrastructure links for the allocation of each NS path is minimized (gamma > alpha + beta).

Request Payload Example

```
{
        "NS_id":"demo1",
        "NS_sla":"gold",
        "tenor_api":"http://10.20.30.40:5454",
        "infr_repo_api":"http://1.2.3.4:5544",
        "ir_simulation":"true",
        "ns_simulation":"false",
        "solver":"unimi",
        "Alpha":"0.5",
        "Beta":"0.3",
        "Gamma":"0.2",
        "fixVnf01":"",
        "toPoP01":""
```

```
}
```

Return Value Example (successful mapping)

```
{
    "status": 0,
    "created_at":"Thu Nov  5 10:13:25 2015",
    "links_mapping":
    [
        {
            "vld_id":"vld0",
            "maps_to_link":"/pop/link/85b0bc31-dff0-4399-8435-
4fb2ed65790a",
        },
        {
            "vld_id":"vld1",
            "maps_to_link":"/pop/link/85b0bc32-dff0-4399-8435-
4fb2ed65790a",
        },
        {
            "vld_id":"vld0",
            "maps_to_link":"/pop/link/85b0bc33-dff0-4399-8435-
4fb2ed65790a",
        },
        {
            "vld_id":"vld1",
            "maps_to_link":"/pop/link/85b0bc34-dff0-4399-8435-
4fb2ed65790a",
        }
    ],
    "vnf_mapping":
    [
        {
            "maps_to_PoP":"/pop/55ef7cce-1e9b-4b8f-9839-d40ceeb670f7",
            "vnf":"vnf_demo1_0"
        },
        {
            "maps_to_PoP":"/pop/55ef7cce-1e9b-4b8f-9839-d40ceeb670f7",
            "vnf":"vnf_demo1_1"
        },
        {
            "maps_to_PoP":"/pop/55ef7cce-1e9b-4b8f-9839-d40ceeb670f6",
            "vnf":"vnf_demo1_2"
        }
    ]
}
```

Return Value Example (mapping failure)

```
{
    "status": 1,
    "error":"Error in MIP problem",
    "info":"MIP solution is undefined",
    "created_at":"Thu Nov  5 10:11:37 2015"
}
```

Return Value Codes

All return messages are in json format and, with the only exception for return code 0, each message has the keys "status" (a numerical value) and an associated "error" (a string). The complete list of statuses and errors is the following:

Status	Error	Description
0	-	All ok / valid mapping found
1	"No feasible solution found"	The solver was unable to find a feasible
-1	"No matching NSd in NS catalog"	-
-2	"No matching SLA in NSd"	-
-3	"No matching VNFd in VNF catalog"	-
-4	"No matching flavour in	-
-60	"No matching NSd in dummy NS catalog"	-
-61	"No matching VNFd in dummy VNF	-
-120	"NS.json not found / not generated"	-
-121	"NI.json not found / not generated"	-
-122	"Invalid json request: no ns_id found"	-

11. ANNEX B: DETAILS ON INFRASTRUCTURE REPOSITORY QUERYING

Details on the sequence of actions performed for querying the infrastructure repository are reported in the table below.

An example of the resulting JSON file storing the queried parameters is reported (i.e. NI.json file).

Table 16 Sequential steps in querying the infrastructure repository

Task	Description
ir_simulation == true?	▪ Check if ir_simulation is true ▪ set ir_simulation_requested accordingly
request PoP list	▪ GET to */pop/* to retrieve list of PoP; ▪ store list to pop_id_array
request PoP detail	▪ for each pop, GET to */pop/<pop-id>* to retrieve the deatil of PoP ▪ clear unused PoP information ▪ store PoP status in pop_detail_array
request link list	▪ GET to */pop/link* to retrieve list of links between PoPs ▪ store list to pop_link_id_array
request link detail	▪ for each link, GET to */pop/link/<link-id>* to retrieve the detail of link ▪ store link status in pop_link_detail_array ▪ convert both bw_Gps and bw_util_Gps into Mbps (modyfing the key accordingly) ▪ note that available bw will be calculate as the difference between occi.epa.pop.bw_Mbps and occi.epa.pop.bw_util_Mbps ▪ also, link delay will be extracted from occi.epa.pop.roundtrip_time_sec
request list of hypervisors	▪ for each PoP, GET to */pop/<pop-id>/hypervisor* to retrieve the list of the hypervisors ▪ store everything in the hash of arrays hypervisors_list_hash
collect data from each hypervisor and build the aggregate resource of each PoP	▪ for each PoP, for each hypervisor in this pop, GET to */pop/<pop-id>/hypervisor/<hypervisor_id>* to retrieve the hypervisor detail ▪ from this hypervisor detail extract the data: ▪ cpus = attributes -> occi.epa.attributes -> vcpus

	▪ vcpu_used = attributes -> occi.epa.attributes -> vcpu_used ▪ ram = attributes -> occi.epa.attributes -> memory_mb ▪ ram_used = attributes -> occi.epa.attributes -> memory_mb_used ▪ hdd = attributes -> occi.epa.attributes -> local_gb ▪ hdd_used = attributes -> occi.epa.attributes -> local_gb_used ▪ calculate the amount of free resource as algebraic difference and accumulate on PoP basis ▪ count the how many cpu have the AES-NI acceleration from attributes -> occi.epa.attributes -> cpu_info -> features ▪ save free resources of each PoP into hypervisors_detail_hash
collect GPU / DPDK data	▪ for each PoP_id: GET to */pop/<pop-id>/pcidev/?dpdk=true* to get list of DPDK-enabled NIcs ▪ store number of DPDK-enabled NICs in PoP_aggregate_resources -> dpdk_nic_count ▪ for each PoP_id: GET to */pop/<pop-id>/osdev/?category=compute* to get list of GPUs (**actually, compute devices?**) ▪ store number of GPUs in PoP_aggregate_resources -> gpu_count
create final hash and save to disk	▪ create a new hash containing pop_id_array, pop_detail_array, pop_link_id_array, pop_link_detail_array, hypervisors_detail_hash ▪ save to disk the hash in json format. create NI.json

An example of the resulting JSON file storing the queried parameters is reported in the following (i.e. NI.json file).


```
{
  "PoP_id": [
    "/pop/55ef7cce-1e9b-4b8f-9839-d40ceeb670f4",
    "/pop/55ef7cce-1e9b-4b8f-9839-d40ceeb670f5"
  ],
  "PoP_detail": [
    {
      "attributes": {
        "occi.epa.pop.graph_db_url":
"http://neo4j:intel_tnova@134.191.243.7:7474/db/data/",
        "occi.epa.pop.lat": "53.3720513",
        "occi.epa.pop.lon": "-6.5130686999999625",
        "occi.epa.pop.name": "Intel Ireland's Leixlip Campus, Kildare,
Ireland",
        "occi.epa.pop.odl_name": "admin",
        "occi.epa.pop.odl_password": "admin",
```

```
        "occi.epa.pop.odl_url":
"http://134.191.243.7:9001/restconf/operational/",
        "occi.epa.timestamp": 1434538341.071382
      },
      "identifier": "/pop/55ef7cce-1e9b-4b8f-9839-d40ceeb670f4",
      "links": [

      ]
    },
    {
      "attributes": {
        "occi.epa.pop.graph_db_url":
"http://neo4j:intel_tnova@134.191.243.7:7474/db/data/",
        "occi.epa.pop.lat": "23.3727512",
        "occi.epa.pop.lon": "18.5199025",
        "occi.epa.pop.name": "Googre Corporation, datacenter 00f3a",
        "occi.epa.pop.odl_name": "admin",
        "occi.epa.pop.odl_password": "admin",
        "occi.epa.pop.odl_url":
"http://134.191.243.7:9001/restconf/operational/",
        "occi.epa.timestamp": 1434449042.2
      },
      "identifier": "/pop/55ef7cce-1e9b-4b8f-9839-d40ceeb670f5",
      "links": [

      ]
    }
  ],
  "PoP_link_id": [
    "/pop/link/85b0bc27-dff0-4399-8435-4fb2ed65790a",
    "/pop/link/85b0bc28-dff0-4399-8435-4fb2ed65790a"
  ],
  "PoP_link_detail": [
    {
      "attributes": {
        "occi.epa.label": "is_connected_to",
        "occi.epa.pop.bw_Mbps": 102400,
        "occi.epa.pop.bw_util_Mbps": 19420,
        "occi.epa.pop.destination": "200.202.200.5",
        "occi.epa.pop.interface": "Interface0",
        "occi.epa.pop.ip_address": "200.202.200.4",
        "occi.epa.pop.netmask": "255.255.255.0",
        "occi.epa.pop.protocol": "MPLS",
        "occi.epa.pop.roundtrip_time_sec": "0.021",
        "occi.epa.pop.source": "Ethernet",
        "occi.epa.pop.type": "Egress",
        "occi.epa.timestamp": 1434701892.961004
      },
      "identifier": "/pop/link/85b0bc27-dff0-4399-8435-4fb2ed65790a",
      "source": "/pop/55ef7cce-1e9b-4b8f-9839-d40ceeb670f4",
      "target": "/pop/55ef7cce-1e9b-4b8f-9839-d40ceeb670f5"
```

```
    },
    {
      "attributes": {
        "occi.epa.label": "is_connected_to",
        "occi.epa.pop.bw_Mbps": 102400,
        "occi.epa.pop.bw_util_Mbps": 15380,
        "occi.epa.pop.destination": "200.202.200.4",
        "occi.epa.pop.interface": "Interface0",
        "occi.epa.pop.ip_address": "200.202.200.5",
        "occi.epa.pop.netmask": "255.255.255.0",
        "occi.epa.pop.protocol": "MPLS",
        "occi.epa.pop.roundtrip_time_sec": "0.023",
        "occi.epa.pop.source": "Ethernet",
        "occi.epa.pop.type": "Egress",
        "occi.epa.timestamp": 1434701892.961004
      },
      "identifier": "/pop/link/85b0bc28-dff0-4399-8435-4fb2ed65790a",
      "source": "/pop/55ef7cce-1e9b-4b8f-9839-d40ceeb670f5",
      "target": "/pop/55ef7cce-1e9b-4b8f-9839-d40ceeb670f4"
    }
  ],
  "PoP_aggregate_resources": {
    "/pop/55ef7cce-1e9b-4b8f-9839-d40ceeb670f4": {
      "aggregate_cpus": 8,
      "aggregate_ram": 4096.0,
      "aggregate_hdd": 160,
      "aggregate_cpu_accel_aes-ni": 3,
      "dpdk_nic_count": 2,
      "gpu_count": 5
    },
    "/pop/55ef7cce-1e9b-4b8f-9839-d40ceeb670f5": {
      "aggregate_cpus": 0,
      "aggregate_ram": 0.0,
      "aggregate_hdd": -2160,
      "aggregate_cpu_accel_aes-ni": 3,
      "dpdk_nic_count": 4,
      "gpu_count": 2
    }
  }
}
```

12. ANNEX C: DETAILS ON THE SERVICE CATALOGUE QUERYING

Details on the sequence of actions performed for querying the service catalogue are reported in the table below.

An example of the resulting JSON file storing the queried parameters is reported (i.e. NS.json file).

Table 17 Sequential steps in querying the service catalogue

Task	Description
set IR and NS/VNF catalog addresses	▪ set IR default address ▪ set NS base address accordingly to ns_simulation parameter
parse of body request and preliminary check	▪ Parse of the request body ▪ Check if NS_id is present, return if not ▪ Check if NS_sla is present; set ns_sla to "gold" as default
query NS catalog	▪ GET to NS catalog for retrieving the NSd ▪ Check if NSd is avaliable in the NS catalog, return if not

store VNF ids associated with selected NS flavour	▪ scan nsd -> sla array looking for an id match
	▪ when found, store for each constituent vnf the
	  ▪ nsd -> sla -> constituent_vnf -> vnf_reference
	  ▪ nsd -> sla -> constituent_vnf -> vnf_flavour_id_reference
	  ▪ nsd -> sla -> constituent_vnf -> number_of_instances

query VNF catalog	▪ -- Loop over constituen_vnfd_array ▪ for each vnf_id in constituent_vnf_array: GET to VNF catalog

flavour management	▪ scan the vnf -> deployment_flavours array for a match with the constituent_vnf -> flavour_id
	▪ once found, store the associated:
	  ▪ vdu_name from vnf -> deployment_flavours -> constituent_vdu -> vdu_reference
	  ▪ num_of_vdu_instances from vnf -> deployment_flavours -> constituent_vdu -> number_of_instances
	▪ return if no match

	▪ vdu_name is used to reference the correct vdu descriptor associated to the selected VNF flavor
collect aggregate reqs for each VNF	▪ scan vnf -> vdu array for a match with vdu_name from the previous step ▪ once found, collect and aggregate the data ▪ tot_cpu with vnf -> vdu -> resource_requirements -> vcpus ▪ tot_ram with vnf -> vdu -> resource_requirements -> memory ▪ tot_hdd with vnf -> vdu -> resource_requirements -> storage -> size ▪ tot_peak_bw with vnf -> vdu -> networking_resources -> peak ▪ tot_aver_bw with vnf -> vdu -> networking_resources -> average ▪ max_bw with vnf -> vdu -> resource_requirements -> network_interface_bandwidth (NOT an aggregate, just max value) ▪ tot_cpu_aesni with vnf -> vdu -> resource_requirements -> cpu_support_accelerator ▪ tot_dpdk with vnf -> vdu -> resource_requirements -> data_processing_acceleration_library ▪ multiply each aggregate data by the number_of_vdus_instance ▪ save the aggregate requirements into vnf_requirements array -- End of loop over constituen_vnfd_array
special requirements management	▪ scan and save the special/additional requirements of each VNF, i.e. GPUs or DPDK-enabled NICs
Forwarding graph management	▪ Scan and save into virtual_links_array array the list of virtual links from nsd -> vld, and extract: ▪ virtual_link_id ▪ root_requirements (converted in MBps) ▪ source ▪ destination ▪ Scan and save into network_forwarding_paths array the list of Network Forwarding Paths from nsd -> vnffgds -> vnffgs -> 0 -> network forwarding path and save: ▪ nfp_id ▪ nfp_graph (list) ▪ connection_points (list)

create the ns_out array and fill with data	▪ store ns_id in ns_out array ▪ store ns_sla in ns_out array ▪ store vnf_id array in ns_out array ▪ store vnf_req array in ns_out array ▪ store virtual_links array in ns_out array ▪ store network_forwarding_paths array in ns_out array
collect dynamic parameters passed by the Orchestrator	▪ check for "alpha", "beta", "gamma" and all the rest in the request body ▪ store them additional parameter in ns_out array
save data to disk	▪ create NS.json, store ns_out in it and save to disk

An example of the resulting JSON file storing the queried parameters is reported (i.e. NS.json file).


```
{
  "ns_id": "demo4",
  "ns_sla": "gold",
  "vnf_id": [
    "/vnf_demo4_0",
    "/vnf_demo4_1"
  ],
  "vnf_req": [
    {
      "vnf_id": "/vnf_demo4_0",
      "req_vcpus": 4,
      "req_ram": 4096.0,
      "req_hdd": 160.0,
      "req_nic_bw": 10000,
      "req_peak_bw": 10,
      "req_aver_bw": 7,
      "req_cpu_accel_aes-ni": 1,
      "req_data_accel_lib_dpdk": 1,
      "vnf_flavour": "vnfflavourid1",
      "vnf_num_of_inst": "1"
    },
    {
      "vnf_id": "/vnf_demo4_1",
      "req_vcpus": 8,
      "req_ram": 4096.0,
      "req_hdd": 160.0,
      "req_nic_bw": 10000,
      "req_peak_bw": 10,
      "req_aver_bw": 7,
      "req_cpu_accel_aes-ni": 1,
      "req_data_accel_lib_dpdk": 1,
      "vnf_flavour": "vnfflavourid1",
```

```
      "vnf_num_of_inst": "1"
    }
  ],
  "virtual_links": [
    {
      "virtual_link_id": "vld0",
      "root_requirements": 10000,
      "source": "data0",
      "destination": "vnf_demo4_0:data0"
    },
    {
      "virtual_link_id": "vld1",
      "root_requirements": 10000,
      "source": "vnf_demo4_0:data1",
      "destination": "vnf_demo4_1:data0"
    },
    {
      "virtual_link_id": "vld2",
      "root_requirements": 10000,
      "source": "vnf_demo4_1:data1",
      "destination": "data1"
    }
  ],
  "network_forwarding_paths": [
    {
      "nfp_id": "nfp0",
      "nfp_graph": [
        "vld0",
        "vld1",
        "vld2"
      ],
      "connection_points": [
        "data0",
        "data1"
      ]
    }
  ]
}
```