Scalable Architecture for Service Function Chain Orchestration

Sahel Sahhaf[1], Wouter Tavernier[1], János Czentye[2], Balázs Sonkoly[2]
Pontus Sköldström[3], Dávid Jocha[4], Jokin Garay[5]
[1] *UGent-iMinds;* [2] *BME;* [3] *Acreo;* [4] *ETH;* [5] *UPV/EHU*

***Abstract*—Network Function Virtualization (NFV) enables to implement network functions in software, high-speed packet processing functions which traditionally are dominated by hardware implementations. Virtualized Network Functions (NFs) may be deployed on generic-purpose servers, e.g., in datacenters. The latter enables flexibility and scalability which previously were only possible for web services deployed on cloud platforms. The merit of NFV is challenged by control challenges related to the selection of NF implementations, discovery and reservation of sufficient network and server resources, and interconnecting both in a way which fulfills SLAs related to reliability and scalability. This paper details the role of a scalable orchestrator in charge of finding and reserving adequate resources. The latter will steer network and cloud control and management platforms to actually reserve and deploy requested services. We highlight the role of involved interfaces, propose elements of algorithmic components, and will identify major blocks in orchestration time in a proof of concept prototype which accounts for most functional parts in the considered architecture. Based on these evaluations, we propose several architectural enhancements in order to implement a highly scalable network orchestrator for carrier and cloud networks.**

I. Introduction

Network Function Virtualization (NFV) [1] enables to implement Network Functions (NFs) such as firewalls or NATs, high-speed packet-processing functions in software which traditionally are dominated by hardware implementations or dedicated middleboxes. NF software may be deployed on generic-purpose servers, e.g., in datacenters. Telecom services (e.g., an Intrusion Detection) which can be decomposed into a Service Graph (SG) of NFs, might now benefit from flexibility and scalability-levels which previously were only possible for web services deployed on cloud platforms such as Amazon EC or Google Compute. Modern control paradigms such as Software Defined Networking (SDN) [2] have the merit of simplifying the service chain provisioning process and reducing the cost in CAPEX and OPEX.

The mapping of NFs of services to infrastructure is one of the core tasks of the orchestrator. This requires that the orchestrator has a view of what are the individual resource requirements of NFs, or even if given NFs might be decomposed to smaller NFs (Service Decomposition). However, as indicated in Section IV, orchestration in realistic service and infrastructure provider contexts quickly involves ten thousands of resource elements, and a multiple of services to be orchestrated.

Existing research related to service orchestration has largely focused on: i) embedding algorithms in small-scale, idealized settings, i.e., limited number of infrastructure nodes, pre-determined SG decomposition, or on ii) NFV architectures, e.g., [1], providing high-level functionality and interfaces for enabling Service Function Chaining with virtual NFs.

Our contribution. In this paper we identify the context and requirements of NFV orchestration over Telecom and datacenter networks. We intend to bridge the gap between abstract embedding algorithms and NFV architectures, by proposing elements of a realistic and scalable resource orchestrator which is able to optimize placement of networking and computing components across infrastructure. Within this state-of-the-art emulation NFV framework [3] we implement an orchestrator supporting decomposition, and identify most significant factors contributing to orchestration time. The latter serves the identification of elements for an improved resource orchestrator framework which is truly scalable, as well as the identification of a set of technologies which are able to implement such design.

The rest of the paper is as follows. Section II describes related work. In Section III we detail the concepts of Service Function Chaining (SFC). Section IV details the problem and context of orchestration of services in a realistic setting and the relation of the orchestration functionality within NFV architectures. An overview of most important orchestration algorithm component is given in Section V, while Section VI identifies experimental performance results and Section VII proposes architecture improvements. Section VIII concludes the paper.

II. Related work

Existing research on NFV orchestration focuses on two aspects: i) design and evaluation of embedding algorithms, and ii) design and implementation of control architectures for production environments. The first relates to the problem of mapping a set of service-related (virtualized) resources to physical infrastructure (e.g., servers and switches). Requested resources might involve networking (bandwidth), as well as node-level resources (computing or memory). The latter is referred to as the Virtual Network Embedding Problem (VNEP), for which a survey can be found in [4]. Recent approaches build on this state of the art, for example by integrating the concept of service decomposition [5].

The second category of research is driven by Telecom research and industry. The NFV Industry Specifications Group (NFV ISG) of European Telecommunications Standards Institute (ETSI), is driven by Telecom operators to strategically steer NFV-related activities. The ISG produced and publicly

released documents on NFV terminology, use cases, requirements and architectures. NFV Management and Orchestration (MANO) is the ETSI-defined framework for the management and orchestration of computing, networking, storage and virtual machine resources in cloud and carrier networks. Multiple community-driven and private software initiatives, such as OpenMANO (https://github.com/nfvlabs/ openmano) or vConductor [6], are developing proof of concepts of proposed ETSI specs. These recent efforts involve single domain designs, architectures and prototypes where scalable orchestration is not the first focus. OpenStack is a cloud infrastructure management framework in which Heat is the component responsible for orchestration. Network connectivity orchestration here relies on Neutron and is heavily focusing on L3 and above, lacking fine-grained forwarding (L2) flexibility required for NFV orchestration. Larger frameworks such as OPNFV (http://www.opnfv.org/) intend to combine existing cloud and network control frameworks (e.g., OpenStack, OpenDayLight) to build a reference implementation for NFV management/control.

EU-funded projects such as FP7 T-NOVA or Mobile Cloud Networking (MCN) also investigate the merits of NFV in Telecom. T-NOVA focuses on an 'NFV Marketplace" composed of: i) a 'Network Function Store" including NFs by several 3rd-party developers and ii) a Brokerage platform enabling customers to trade with the T-NOVA service provider and 3rd-party function developers. MCN investigates NFV as an enabler for increased flexibility in the backhaul of mobile networks. Both projects define some kind of orchestrator for mapping virtual resources to physical infrastructure and managing the life-cycle of virtualized resources. These are ongoing, and do not directly focus on the design and implementation of a scalable, potentially multi-domain orchestrator, involving recursivity in the control and orchestration layers as the proposed approach.

III. Service Function Chaining

In order to easily introduce joint programmatic interfaces for controlling different types of resources, such as compute, storage and networking ones, we have defined a common model to be used at different reference points called Network Function Forwarding Graph (NF-FG) [7]. It provides support for functionalities such as resource orchestration or service decomposition, on the one hand, and features such as scalability, dynamicity or support for DevOps in Service Provider environments, on the other. The model is capable of storing service description as SG, resource information as Resource Graph (RG) and mapping of requests to resources as NF-FG (see Fig. 1, at left the SG is shown on top and NF-FG with mapping is shown at bottom).

The SG defines the service functions and their logical connectivity, the Service Access Points (SAPs) to the service and the Service Level Specification to meet the Service Level Agreement. It is only used as a standalone element when there are no resources involved yet.

The RG describes the (virtual) resources that will be used to deploy the requested services. It provides a homogeneous representation of the (virtualized) infrastructure, in terms of both capacities and capabilities, at the defined abstraction level. For example, in domains with hierarchical orchestration processes, the RG in the higher level orchestrators has a wider scope and abstracts away the finer grain details of the underlying resources, whereas the RG in the lower level orchestrators has a fine grain detail of the resources.

The NF-FG contains the assignment of NFs to the virtualized software resources; the definition of the forwarding behavior in the virtualized network resources and the service requirements which can be evaluated at the network and software abstraction layers.

The NF-FG evolves from its original definition as SG to RG mapping. On the one hand, while the NF-FG progresses down through the architecture it will be further characterized, the service decomposition process will also decompose the components (e.g. NFs) and it will be split into smaller subgraphs if deployed in different infrastructure domains. On the other hand, during the service lifecycle the NF-FG will also evolve from the initial deployment as a consequence of internal re-optimization processes, modifications to the service requested to the orchestrator or external changes (e.g. infrastructure updates, auto-scaling).

Service decomposition is the process of transforming an NF-FG containing abstract NF (s) to NF-FG (s) containing less abstract, more implementation-close NF (s). This can also include dividing the functionality of a complex NF to several, less complex NFs. This allows for a step-wise translation of high-level (compound) NFs into more elementary NFs, which can eventually be mapped onto the infrastructure. During the decomposition of an NF, the external interfaces remain unchanged. Formally, a decomposition rule can be seen as a NF $\rightarrow$NF-FG mapping. There can be multiple decompositions for an NF.

A sample NF decomposition for an Intrusion Detection System (IDS) service is shown at the right of Fig.1. It can be implemented with a hardware appliance or a monolithic Virtual Machine (VM). The IDS control logic is decomposed into an IDS Control VM, a Firewall (FW) component to block the identified malicious traffic and a traffic analyzer. The FW may be mapped to a Forwarding Element (FE) and the traffic analysis is realized by a generic Deep Packet Inspection (DPI) VM component.

IV. Service Chain Orchestration

An orchestrator is responsible for the service management and orchestration. The main functionalities of an orchestrator are: i) optimal mapping of Virtualized Network Functions (VNFs) across infrastructure, ii) instantiating VNFs at reasonable locations, iii) keeping track of VNFs location, iv) assigning and scaling resources to the VNFs and v) serviceVNFs monitoring.

Requirements. Orchestration process is impacted by scale of Telecom operator network and the number of service requests. Based on the discussions with network operators and information available on Telecom operator networks such as BT[1] and datacenters info in UK[2], some parameters and requirements for network topology, customers and their requests

[1] http://www.kitz.co.uk/adsl/21cn_network.htm

[2] http://www.datacentermap.com/united-kingdom/

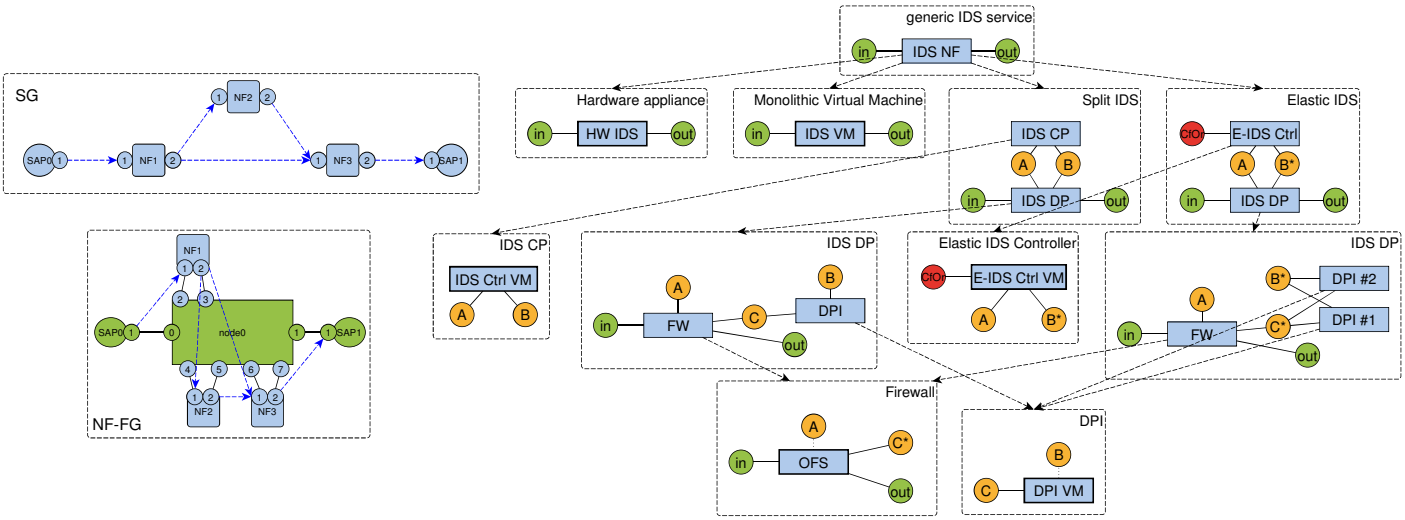

Fig. 1. Example of SG, NF-FG and Network Function decomposition

were identified. A typical Telecom operator network has a hierarchical structure with a dense core router meshed network consisting of inner and outer core Points of Presence (PoPs). The end customers are interconnected to this core network via a hierarchy of tree-structured access- and metro aggregation networks. Considering BT, such an infrastructure consists of almost $50K$ devices at different parts in the network and datacenters excluding CPEs. In case of including CPEs, almost $10M$ devices are needed to be orchestrated. The number of new customers per day is equal to almost $2.5K$ and the number of service requests is estimated around $5-10K$ per day. There are several IETF drafts on use cases for Service Function Chain (SFC). Based on these drafts, a service chain request is typically a Directed Acyclic Graph with topologies such as a simple path or a forking path.

Capability. In order for the orchestrator to fully exploit the capabilities of the servers, all the features/capabilities available in the infrastructure should be identified. Devices such as Acceleration Hardware (AH) (e.g., FPGAs, GPUs and MICs) and advanced network interfaces cards can improve the performance of many NFs by offloading several performance-critical tasks in an NF to these devices. The challenges occur when the complexity of NFs increases which makes the implementation of NFs infeasible due to resource limitation in AH. A solution to this issue is the support of service decomposition in the orchestration process (see Section III). Complex NFs can be decomposed to more elementary NFs and there should exist the hardware description of performance-critical NFs to be installed on AH. An important task is to design the NFs in such a way that efficient usage of specific capabilities is ensured. Note that for the mapping of NFs to the infrastructure, both hardware static metrics (e.g., location and supported features) and dynamic metrics (e.g., CPU utilization and current available memory) should be known to the orchestrator. Besides, values of resource demands should be coherent with node specification constraints. Depending on the form of the exposed resources, comparison of demands and available resources can be a challenging task. To address this issue, different profiling tools (e.g., GNU gprof or Tuning and Analysis Utilities (TAU)) can be used to measure the application's performance. Benchmarks such as the one provided by Standard Performance Evaluation Corporation (SPEC) enables direct comparison of processors' performance. Additionally, analytical techniques (e.g., Amdahl's law and Gustafson's law) can be used to model the performance of multi-core CPUs. In spite of existence of several profiling tools, it is challenging to have a benchmarking with high-accuracy.

Interface. A generic API for an orchestrator has to support the following operations: i) instantiate/tear down/change NF-FG: once an NF-FG arrives at the orchestrator, it tries to execute it based on its global resource view. Changing a request includes operations such as modifying the NF demands and inserting/removing NFs in the NF-FG ii) get/send virtual resource info: the orchestrator provides resources, capabilities and topology information iii) notification/alarm: any failure or unexpected event can be reported by the orchestrator iv) get/send observability info: measurement reports on Key Quality Indicators (KQIs) related to NFs can be provided by the orchestrator v) start/stop/restart NFs and switches vi) connect/disconnect NFs to switches and vii) configure switches. Possible option for addressing the last operations is using different protocols at the southbound interface of the controllers such as OpenFlow, NETCONF and OFconfig.

A. *ESCAPE framework*

We have established a prototyping framework called ESCAPE[3] including 3 layers of Infrastructure Layer (IL), Orchestration Layer (OL), Service Layer (SL) and demonstrated the first version in [3]. The main goal of ESCAPE is to support the development of several parts of the service chaining architecture including VNF implementation, traffic steering, virtual network embedding, etc. However, here we focus on the orchestration part. ESCAPE is (mainly) implemented in Python on top of POX (OpenFlow controller) platform and Mininet. The modular approach and loosely coupled components make it easy to change several parts and evaluate own algorithms. The system architecture of the next version of ESCAPE (without the Mininet based IL) is shown in Fig. 2.

[3]Extensible Service ChAin Prototyping Environment using Mininet, Click, NETCONF and POX (ESCAPE)

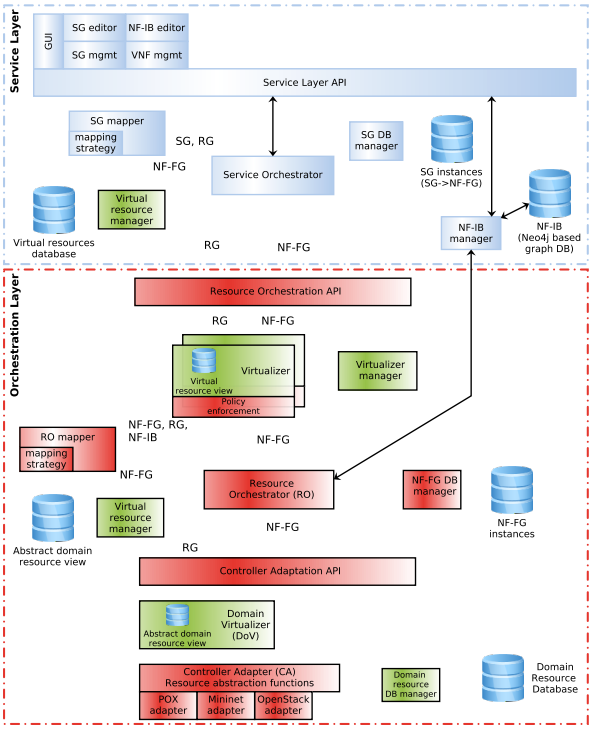

Fig. 2. System architecture of ESCAPE

SL contains an API and a GUI at the top level where users can request and manage services and NFs. The API is capable of formulating SG from the request and passes that to a dedicated service orchestrator which is responsible for gathering resource information (RG) from Virtual resource manager. This is the virtual view provided by the Virtualizer of the lower layer. Mapping of SG to RG is delegated to the SG mapper module which constructs an NF-FG storing the request, the virtual resources and the mapping between NFs and infrastructure nodes.

OL encompasses the most important components of the resource orchestration process which replaces the ETSI's Virtualized Infrastructure Managers (VIMs). An API is set up on the top centralizing the interaction with the upper layer. On the one hand, the request coming as an NF-FG is forwarded to the Resource Orchestrator (RO) via the corresponding Virtualizer (which is responsible for policy enforcement as well). On the other hand, the virtual view created and managed by the Virtualizer is provided as an RG to the upper layer. RO is the key entity managing the components involved in the orchestration. The input is an NF-FG which should be mapped to the abstract domain view provided by the Domain Virtualizer. RO collects and forwards all required data to RO mapper. More specifically, the NF-FG, the domain view (as an RG) and the Network Function Information Base (NF-IB) are passed to the RO mapper which invokes the configured mapping strategy and interacts with the Neo4j graph database containing information on NFs and decomposition rules[4] (see Section IV-B). The outcome is a new NF-FG which is sent to the Controller Adaptation part. The role of Controller Adapter (CA) is twofold. First, it gathers technology specific information on resources of different domains then builds an abstract domain view. The interaction with different types of technology domains are handled by adapters (e.g., OpenStack adapter for clouds managed by OpenStack). Second, the incoming NF-FG request is decomposed according to the low-level domains and delegated to the corresponding adapters.

[4]NF-IB corresponds to "VNF Catalogue" in NFV MANO with the difference of supporting service decomposition.

B. Network Function-Information Base (NF-IB)

The NF-IB is the entity responsible for storing the NF models/abstractions, NF relationships, NF implementation image(s) and NF resource requirements (see Fig. 2). The NF-IB supports the definition of abstract NFs such as a FireWall, referring to a type, a potential number of ports/interfaces, as well as dependencies to other NFs. As explained in Section III, abstract NFs might be implemented through more refined NFs or might be decomposed themselves into multiple NFs interconnected into an NF-FG with the same external interfaces as the higher-level NF. The NF-IB is capable of storing these relationships into a tree-like data structure in support of the decomposition process (cfr. Figure 1). The leaves of the decomposition tree are NFs for which low-level implementation and deployment information is available such as images, provisioning scripts, resource requirements in terms of CPU, memory and storage.

We have implemented the NF-IB in Neo4j database. As this database is capable of storing key-value pairs for nodes and edges, for each NF we have stored the explained tree-like structure with all the corresponding information of nodes and links. Several modules have been implemented to enable i) updating of the database and ii) retrieval of all possible decompositions of a given SG.

V. Embedding Algorithm

We have implemented a proof of concept embedding algorithm which supports service decomposition using the Neo4j-based NF-IB explained in Section IV-B. It is implemented in Python in compliance with the ESCAPE framework. Given an NF-FG to this module, it retrieves all possible decompositions from the NF-IB and selects a suitable decomposition which is mapped to the network infrastructure. In order to connect to the Neo4j-based NF-IB from Python, we have used Py2neo library. The embedding algorithm was proposed in [5] but was only evaluated in terms of service acceptance ratio. As detailed in [4], there exist several algorithmic approaches in the literature to solve the embedding problem. Importantly, the implemented embedding algorithm is different from the existing approaches in the sense that service decompositions are taken into account at the time of embedding and a resource-aware selection is made. We briefly explain this algorithm in this section.

The objective of the algorithm is to minimize the embedding cost which is achieved by minimizing the resources consumed in the infrastructure to map a request. This allows accepting more requests over time and increases the acceptance ratio. As service decompositions are known from the design time, we can make a resource-aware decomposition selection which would certainly improve the performance of the embedding as a reasonable decomposition is selected which corresponds to the existing resources and thus leads to better placement of the NFs.

The algorithm is based on a backtracking mechanism and is composed of two phases: i) Decomposition selection and ii) Mapping.

In the first phase, given an NF-FG all of its possible decompositions are retrieved from the Neo4j-based NF-IB. For each decomposition a cost is calculated based on: i) number of NFs in the decomposition, ii) number of candidate physical nodes with sufficient capacities which can potentially host the NFs in the decomposition and iii) Cluster Factor (CF) which is calculated as follows: the NFs with similar types which are directly connected (without intermediate NFs with other types) are grouped in a same cluster. The number of clusters in the decomposition is the CF of that decomposition.

NFs types refer to the implementation of the NFs as they can be implemented through different techniques such as: Virtual Machine (VM) images in different virtualization techniques (e.g. Xen, Vmware), process in a container, packet I/O drivers (e.g. DPDK), or hardware appliances. It is of great importance to take NFs types into account at the time of the embedding because not all physical nodes of the infrastructure support all types.

CF is taken into account in the cost function to enable more efficient resource consumption. The more the number of NFs with the same type, the more NFs might be mapped into a same physical node. This leads to less resource consumption, if the similar-type NFs are interconnected directly.

The minimum cost decomposition is selected in the first phase of the algorithm. The mapping phase is based on a backtracking mechanism which tries to minimize the resource consumption of the mapping. The NFs of each cluster (see explanation for CF calculation) are sorted based on their requirements in descending order and the mapping of the NFs of the cluster with maximum requirement starts first. For each of the unmapped NFs, we sort its corresponding candidate physical nodes based on their distance (hop count) to the used physical nodes in ascending order. Every time a physical node is selected to host an NF it is checked if all connected links to the NF can be mapped as well. If not, another candidate physical node is investigated. If none of the nodes can host the NF the algorithm backtracks to the previous mapped NF and selects another candidate node. For more detailed explanation of the algorithm, we refer the interested readers to [5].

VI. Performance Evaluation

The goal of experiments in this paper is to identify the major blocks in orchestration time in the implemented proof of concept prototype. Additionally, we see the effect of increase in the topology size, SG size and number of service decompositions on the performance of the embedding.

In our recent work [5], we have evaluated the proposed embedding algorithm in terms of cost and acceptance ratio. We refer the interested readers to [5] to see the added value of considering service decompositions at the time of the embedding and the impact of the service decomposition choices on the resource footprint.

As our intention is to evaluate the embedding execution time on physical topologies with different sizes, we have generated random regular networks with 100-1000 nodes with degree 3. For each of the generated topologies, the resources of nodes such as memory, storage and CPU and the links bandwidth and delay are numbers uniformly distributed between 100-300. The SGs are also generated randomly and each pair of nodes is connected with probability 0.5. The resource demands of NFs and links within an SG are numbers uniformly distributed between 1-20. Each scenario is iterated 50 times and the average value is reported.

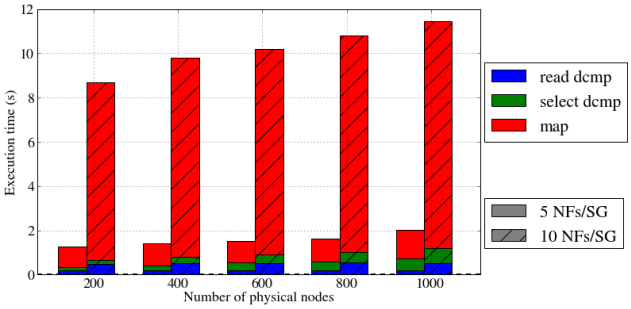

Fig. 3. Embedding execution time for SGs with one decomposition

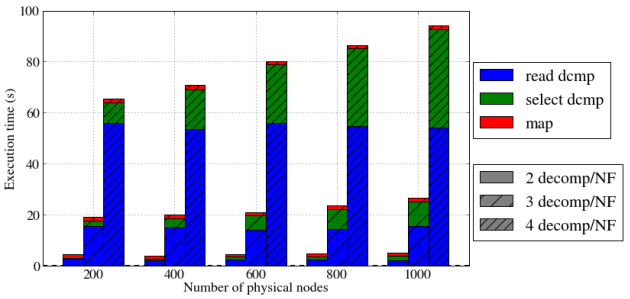

Fig. 4. Embedding execution time for SGs with 5 NFs

In the first experiment, we have evaluated the execution time of the embedding of an SG into physical networks of different sizes for two scenarios: i) SGs with 5 NFs and ii) SGs with 10 NFs. Each SG has only one decomposition. Fig. 3 reports the execution time of different blocks in the embedding algorithm. These blocks include: i) retrieving/reading of all decompositions from the NF-IB (read dcmp), ii) decomposition selection (select dcmp) and iii) mapping of the selected decomposition (map). Based on the results, the mapping is the dominant block and it increases significantly with the increase in the number of NFs, when only one decomposition exists for an SG.

The next experiment evaluates the effect of increase in the number of decompositions for an SG. Fig. 4 reports the execution times for 3 scenarios in which the number of decompositions per NF in an SG changes from 2 to 4. As there are 5 NFs in each SG, the number of decompositions in each scenario is: 5^2, 5^3 and 5^4. As we see 'map' and 'read dcmp' blocks seem to scale quite well, whereas 'select dcmp' is the block which scales poorly with increasing number of nodes in the network. For small topologies with few nodes, 'read dcmp' is the dominant block while for larger topologies (1000 nodes and more) 'select dcmp' seems to be the main concern. It is worth mentioning that the 'read dcmp' block includes the time for reading NFs decomposition from the NF-IB and the time needed for calculating the service decompositions. This is because only NFs decompositions are stored in the NF-IB

and possible service decompositions should be calculated upon request.

The experiments in this section clearly identified the major blocks in the orchestration time in the proposed embedding approach. Knowing these blocks, in the next section, we propose several architectural enhancement to improve the scalability of the orchestrator.

VII. Discussion: Towards a scalable orchestration framework

In this section, we propose several architectural enhancements and explain existing challenges in order to implement a highly scalable orchestrator and meet the requirements mentioned in Section IV.

Parallel/distributed embedding. Based on the results, we identified the 'read dcmp' block to be the most time consuming block in the embedding in smaller topologies while 'select dcmp' block seems to be a major issue in larger topologies. Changing the embedding algorithm to a distributed approach in which costly calculations are done in parallel can improve the performance of the embedding significantly. A possibility to parallelize the 'read dcmp' block of the algorithm is to use Neo4j which supports High-Availability (HA) by distributing the full database onto multiple nodes, resulting in database read performance that scales near linearly with the number of nodes in the cluster. Using the Neo4j HA the 'select dcmp' block can simply be computed in parallel as the cost calculation of each decomposition is independent of others. However, parallelizing the 'map' block is a challenging task. This phase is equivalent to the typical VNEP as SGs composed of atomic NFs are similar to virtual networks which should be mapped to a physical infrastructure and thus similar solutions to VNEP can be considered for the mapping phase (e.g. [8]). The options for mapping parallelization are: i) considering all possible combination of NFs mapping to the physical nodes and selecting the minimum cost mapping. The feasibility/cost of each mapping can be checked in parallel. This approach is feasible only in small topologies (o(100) nodes) as the number of combination increases drastically with a small increase in the topology size, ii) selecting the first-fit physical node for mapping of NFs and finding the shortest path between nodes. NFs mapping/path calculation can be done in parallel. If the first-fit mapping is not successful, the next one is selected. A challenge is to avoid different threads reserving the same resource. Batch scheduling is a solution in which each job gets dedicated access to the resources.

Hierarchical embedding. The other alternative to achieve a scalable orchestration process is to have a hierarchical embedding process. In this process, SG can be divided into different subgraphs using service decompositions available in the NF-IB and each subgraph can be given to a different domain to be orchestrated locally. The main challenge in such distributed embedding relates to the amount of resource and infrastructure information that needs to be advertised to the upper layer orchestrators to facilitate an efficient embedding process. Each domain may expose to upper layers only high-level and aggregated information such as total available capacities and capabilities or aggregated PoP-level information instead of detailed router-level topologies. Such incomplete information in the higher layer orchestrator might lead to inefficient embeddings with performance far from the optimal solution. It is a challenging task to identify the trade-off between the efficiency of the embedding and the amount of infrastructure information exposed by each domain.

Pre-defined service chains. Another enhancement option, independent of the embedding approach, is to have pre-defined service chains with pre-defined decomposition templates. With such templates different parts of the embedding can be done proactively.

VIII. Conclusion

In Service Function Chaining (SFC), virtualized Network Functions (NFs) are chained to compose a network service. This paper has focused on the design of an adequate resource orchestrator to steer the control of SFCs. The main goal of an orchestrator is to map network functions of a requested service (i.e., service function chain) to infrastructure network and compute resources. Orchestration might involve thousands of requests in the period of one business day to be mapped on one or more infrastructure provider networks involving ten thousands of network elements. Scalability is therefore an important characteristic of an orchestrator component. The system architecture, related components and a new service representation model were explained in detail. Important elements of mapping algorithms were characterized and an algorithm supporting service decomposition was implemented as a proof of concept. The key time consumers within the implemented PoC were identified, and a scalable distributed orchestrator architecture, as well as related technologies were proposed based on these findings.

Acknowledgment

This work was conducted within the framework of the FP7 UNIFY project, which is partially funded by the Commission of the European Union.

References

[1] ETSI, "White Paper: Network Functions Virtualisation (NFV)," 2013. [Online]. Available: http://portal.etsi.org/NFV/NFV_White_Paper2.pdf

[2] ONF, "Open networking foundation," 2014. [Online]. Available: https://www.opennetworking.org/

[3] A. Csoma, B. Sonkoly, L. Csikor, F. Nemeth, A. Gulyas, W. Tavernier, and S. Sahhaf, "ESCAPE: Extensible Service ChAin Prototyping Environment using Mininet, Click, NETCONF and POX. Demonstation." in *ACM SIGCOMM 2014*, 2014.

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

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

[6] W. Shen, M. Yoshida, K. Minato, and W. Imajuku, "vconductor: An enabler for achieving virtual network integration as a service," *Communications Magazine, IEEE*, vol. 53, no. 2, pp. 116–124, 2015.

[7] W. Tavernier, S. Sharmaa, S. Sahhaf, R. Szabó, D. Jocha, P. Sköldström, J. Matias, J. Garay, G. Agapiou, B. Sonkoly, M. Rost, T. Jungel, A. Rostami, and X. Cai, "D3.1 Programmability framework," UNIFY Project, Deliverable 3.1, Oct. 2014. [Online]. Available: https://www.fp7-unify.eu/files/fp7-unify-eu-docs/Results/Deliverables/UNIFY_D3.1%20Programmability%20framework.pdf

[8] Q. Yin and T. Roscoe, "Vf2x: fast, efficient virtual network mapping for real testbed workloads," in *Testbeds and Research Infrastructure. Development of Networks and Communities*. Springer, 2012, pp. 271–286.