CUSCO: A Customizable Solution for NFV Composition

Vinicius Fulber-Garcia[1(✉)], Marcelo Caggiani Luizelli[2], Carlos R. Paula dos Santos[3], and Elias P. Duarte Jr.[1]

[1] Federal University of Paraná, Curitiba, Brazil
{vfgarcia,elias}@inf.ufpr.br
[2] Federal University of Pampa, Alegrete, Brazil
marceloluizelli@unipampa.edu.br
[3] Federal University of Santa Maria, Santa Maria, Brazil
csantos@inf.ufsm.br

Abstract. Although Network Function Virtualization (NFV) has multiple advantages in comparison with traditional hardware middleboxes, there are still many open problems. Some of the major challenges are related to the service deployment process (composition, embedding, and scheduling). In particular, current solutions for network service composition are limited, in the sense that they are not customizable, neither in terms of the evaluation setup nor the operational behavior. In this paper, we propose a new adaptive service composition solution that takes into account multiple specific requirements of network operators. The proposed solution uses a statistical method to conciliate different metrics, disparate granularities, and conflicting objectives, and returns a composition result that maximizes the cost-benefit. We present a case study and experiments to show the feasibility of the proposed solution.

1 Introduction

The Network Function Virtualization (NFV) paradigm aims to decouple the network functions from dedicated hardware, employing instead a software plane [10]. Most important, NFV allows the creation of virtualized network services through the connection of multiple network functions in a traffic forwarding/processing structure called Service Function Chain (SFC) [12]. One of the most important processes related to SFCs is their *deployment*. The deployment process of a network service onto a virtualized environment consists of a series of inter-related tasks [4]: composition, embedding, and scheduling. Several works have been proposed to tackle the particular challenges of the deployment tasks [1,3,5,11].

Most existing service deployment solutions consider just the network topology, including information about the physical resources available, and some other predefined parameters. Thus, individual needs of clients and network operators are disregarded. In particular, the existing composition solutions, such as [2,3,9,11,13,14] all include a pre-defined and hard-coded evaluation setup. The evaluation setup requires specific information about the services to deploy

L. Barolli et al. (Eds.): AINA 2020, AISC 1151, pp. 204–216, 2020.
https://doi.org/10.1007/978-3-030-44041-1_19

including the operational behavior of the multiple network functions. Frequent examples of evaluation setups include the computational resources requirements, bandwidth requirements, traffic ratio, service chain size, and priority levels. It is important to notice that these limited composition solutions are inflexible, restrictive, and, will eventually become obsolete. Furthermore, the network operators are required to adapt their needs to the evaluation capacities of the available solutions, and not the other way around, as it should be.

Composition solutions should deal with different, and even conflicting, needs of the network operators (*e.g.*, minimizing the traffic, maximizing the computational capability, minimizing the energy consumption) and heterogeneous functions/services (*e.g.*, routing, security, load balancing). In this work, we present a CUstomizable Service COmposition (CUSCO) solution that applies innovative models to enable on-demand customization of the evaluation setup. CUSCO uses a dynamic statistical method that can process these customized evaluation setups. In order to conciliate the multiple needs of network operators, the proposed solution computes a weighted function representing the cost-benefit of each composition candidate, defining a unified index that reflects the suitability of each alternative.

The rest of this paper is organized as follows. Section 2 presents an overview of the composition task as well as relevant related works. In Sect. 3, we describe and specify the proposed CUSCO solution. Section 4 presents a case study and experiments of CUSCO applied to customize a service chain composition. Finally, Sect. 5 concludes the paper.

2 Network Service Composition in a Nutshell

In NFV, network services are deployed on a virtualized infrastructure as a sequence of interconnected network functions. The deployment process consists of three tasks: composition, embedding, and scheduling. Particularly, composition solutions receive as input **`network service requests`** (document containing every necessary information to execute the composition task) and output the best **`composition result`** (including the strict order of the network functions and connections) according to an evaluation setup.

Currently, the evaluation setup of existing composition solutions is statically defined and allow one or more predefined **`evaluation metrics`** (specific information related to the network functions) used by a hard-coded **`objective function`** (an optimization function is based on the evaluation metrics). The objective functions compute **`candidates`** (possible composition results – which are called service topologies). When objective functions are based of multiple metrics, the evaluation result of a single metric is called a **`partial result`** and the partial results are jointly considered to find the final result.

Finally, the solutions are either based on exhaustive search or heuristic search. The **`exhaustive search`** model generates and evaluates all the possible service toplogies that are candidates that solve the composition problem. The **`heuristic search`** model iteratively composes a single service topology that

is the composition result. Although the heuristic search model typically reduces the computational complexity of the composition task, it does not guarantee the optimal result. On the other hand, the exhaustive model requires more processing time and computational resources to find the optimal result, and can only be used if the search space is small, which is typically the case as services often consist of a few funtions.

Related work includes several composition solutions proposed in recent years. For example, in [9] and [2], partially ordered service topologies (*i.e.*, topologies with segments of network functions that can be located in different positions) are processed to create candidates. In [9] both exhaustive and heuristic (greedy) models are employed to compose a service that reduces the overall traffic ratio. [2] employs a Pareto simulation model to minimize the service topology size, traffic ratio, and computational resources requirements.

Other solutions explore the possible/allowed relationships between network functions to do the service composition. In [11], an exhaustive search solution presented to minimize the expected traffic ratio and total computational resources requirements. The tabu search meta-heuristic proposed in [3], in turn, aims to minimize the bandwidth requirements to compose a requested network service. Furthermore, the authors of [14] present an exhaustive search solution to minimize link overload while maximizing the multi-tenant network functions usage. Finally, in [13] an automatic composing solution is proposed. This solution identifies functions dependencies based on a description of their operational behavior, thus iteratively creating and evaluating candidates in a Hasse diagram through a priority objective function.

The state-of-art composing solutions, regardless of the performance for finding composition results related to their objective functions, do not enable the network operators to request a custom evaluation of particular objective functions. Currently, multi-objective optimization *a posteriori* methods are usually employed to achieve some flexibility in the evaluation of the candidates. For example, a weighted-sum is used in [2] to define on-demand the importance of each metric in the objective function. However, it is not possible to choose the evaluation metrics themselves, neither the mathematical operations that are applied in their evaluation.

3 CUstomizable Service COmposing

In this section we describe the proposed CUstomizable Service COmposing (Available at https://github.com/ViniGarcia/NFV-FLERAS) (CUSCO), a novel solution for composing network service topologies. This section describes the design of CUSCO, including its operation, the statistical evaluation methodology adopted, and the request data model.

`Request Data Model`. The request data model used by CUSCO is implemented as a YAML document and has four main blocks: `METADATA`, `OBJECTIVES`, `SERVICE`, and `SPECIFICATION`. Each block contains objects

and attributes that provide data about the network operator requirements, service structures, and benchmark information to be computed during the composing task. Figure 1 shows a sketch of the CUSCO request data model.

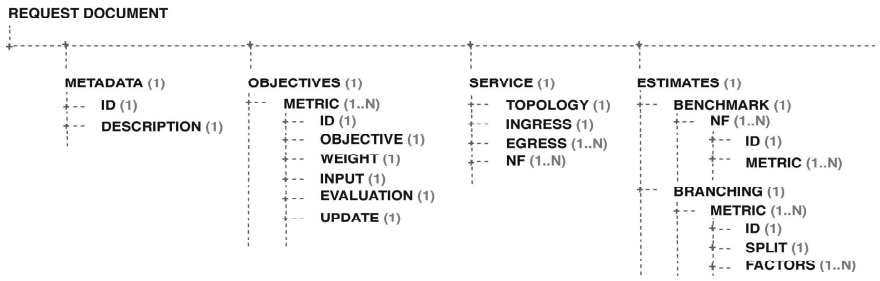

Fig. 1. The CUSCO request data model

The first block, `METADATA`, is used for the identification of a particular request through two attributes: an `ID` (*e.g.*, UUID) and a high-level text `DESCRIPTION`. The `OBJECTIVES` block, in turn, defines the metrics that must be evaluated in the composing task. This block is composed by one of more `METRIC` objects, each of which contains six attributes: an exclusive metric `ID`; the optimization `OBJECTIVE` (*i.e.*, maximization or minimization); the relative importance of the metric in the overall objective function as a `WEIGHT` ($0 < WEIGHT <= 1 | \sum WEIGHT = 1$); an initial numerical value `INPUT` to be iteratively evaluated and updated during the composition; and, finally, the `EVALUATION` and the `UPDATE` operations ($+$, $-$, *, or /) to be applied between the input and the network functions benchmark (discussed later).

The `SERVICE` block carries information about the requested service topology and its elements. In this block, the attribute of `TOPOLOGY` specifies a service through a context-free grammar, called Service ChAin Grammar (SCAG - Available at https://github.com/ViniGarcia/NFV-FLERAS/tree/master/SCAG). SCAG allows the creation of a myriad of services from straightforward linear topologies to topologies with terminal and non-terminal branching structures. We call a "branching structure" the part of the topology that has branches (from the function that precedes the branches to a terminal or intersection point). Most important, SCAG enables the network operators to define partially ordered segments where the network functions included can have their position and connections exchanged among themselves. Other attributes in the `SERVICE` block are `INGRESS`, `EGRESS`, and `NF` that specify the symbols of, respectively, the ingress data node, egress data nodes, and network functions.

The last block, `ESTIMATES`, consists of two objects (`BENCHMARK` and `BRANCHING`) that provide estimate information used to evaluate the objective function and compose the service topology. The `BENCHMARK` object carries tests of individual network functions that employ the evaluation metrics requested in the `OBJECTIVES` block. This object has a sub-object called `NF` that, in turn, has

two attributes: a network function `ID` previously defined in `SERVICE.NF`; and a list `METRIC` with tuples relating each `OBJECTIVES.METRIC.ID` to a benchmark value of the respective network function `ID`. Observe that every network function requested in the `SERVICE` block must have a `BENCHMARK.NF` object in `ESTIMATES`.

Still in the `ESTIMATES` block, the `BRANCHING` object specifies the inputs of the different branches for branching structures. This object uses the sub-object `METRIC` that defines the attributes `ID`, `SPLIT`, and `FACTORS`. The `ID` attribute identifies an evaluation metric in the `BRANCHING`, every evaluation metric must have a corresponding `BRANCHING.METRIC` object. The `SPLIT` attribute defines an operation ($+$, $-$, *, and /) to be applied between an `OBJECTIVES.METRIC.INPUT` and the splitting factor of each branch segment in a branching structure, specified as tuples of the *FACTORS* attribute. Each tuple represents a particular branching structure and its elements are the splitting factor of each branch in that branching structure.

`Operational Settings.` CUSCO executes the composing task in two phases: (i) topologies expansion and (ii) topologies evaluation. In the topology expansion phase, a set of candidates is created through the processing of partial ordering permutations and branching remodeling. The other phase, the topologies evaluation, is responsible to iteratively process (*i.e.*, function to function) the evaluation metrics for every available candidate, evaluate the partial results, creating the candidates' Suitability Index (SI), and compare the final results to rank the candidates.

Topologies Expansion. The topologies expansion phase is sub-divided in two procedures: (i) partial ordering permutations and (ii) branching remodeling. The partial ordering permutations procedure acts on solving partially ordered segments of the requested service topology (specified with SCAG). A permutation with constraints is executed for each partially ordered segment. We consider the constraints as the network functions dependencies (ordering and coupling) that limit the search space. CUSCO generates all possible permutations for each partially ordered segment, thus verifying all of them and removing improper results which violate some constraint. Finally, a Cartesian product is executed on the valid permutations of different partially ordered segments. This procedure, summarized in Fig. 2, returns an exhaustive set of valid service topologies with all network functions pinned on a specific position.

Branching remodeling, in turn, consists of the processing of branching structures with fully ordered service topologies (*i.e.*, with pinned functions). This procedure aims to find identical network functions pinned in the same position among every branch of a particular branching structure. Therefore, these network functions can be reduced to a single instance and be allocated to a common position in two cases: when they precede a branching structure and at the end of a non-terminal branching structure. Observe that the network functions involved in branching remodeling, besides the identical positions, must have compatible administrative domain dependencies (if they exist) to be merged in a single instance. Figure 3 depicts an example of the described procedure.

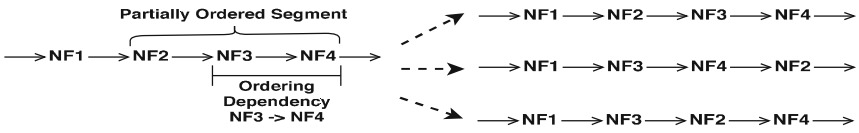

Fig. 2. Partial ordering permutation process

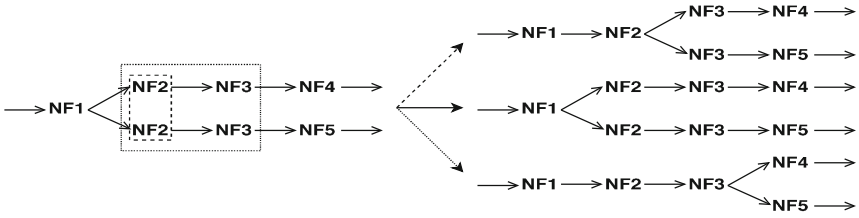

Fig. 3. Branching remodeling process

Topologies Evaluation. In the topologies evaluation phase, the candidates are processed with multiple metrics to generate partial results and, finally, evaluate the objective function. First, in this phase, all the partial results are computed to every candidate. Next, CUSCO generates the candidates' suitability indexes and ranks them. Thus, the topology evaluation is composed of two procedures: (i) partial results generation and (ii) candidates evaluation. Before starting to compute the candidates' partial results in the first procedure, CUSCO maps each requested evaluation metric to a mathematical function. This is done by creating variables to receive the metric input and the network function benchmark, as well as establishing the math operation that is executed between them. A similar process is done to create the metric update functions, thus mapping them to another mathematical function. Figure 4 shows an example of the described mapping process to a Traffic Ratio (TR) metric.

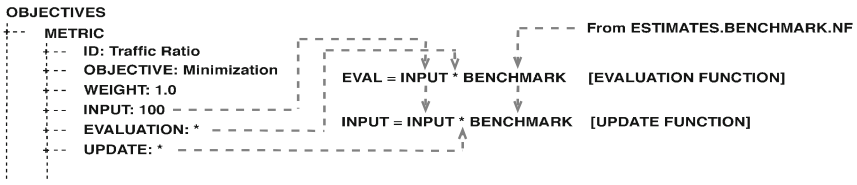

Fig. 4. Evaluation metric mapping example

Both evaluation and update functions are used to generate the partial results iteratively for each evaluation metric. The metrics' inputs are evaluated in a service topology node by node (*i.e.*, corresponding to network functions) and updated edge by edge (*i.e.*, corresponding to virtual connections). Furthermore, if a branching structure is reached during a metric update, an extra operation is

triggered. In this case, estimations (provided in the service request) are used to split the metric input among the upcoming branch segments. While traversing a branching structure a single evaluation/update iteration consists of multiple parallel sub-operations in the different branch segments, this is repeated until the service topology ends in egress nodes (terminal branching) or the branching structure ends in an intersection point (non-terminal branching). In the case of non-terminal branching, another extra update operation is executed at the intersection point. Thus, previously split metric inputs are re-aggregated in a single value. At the end of this procedure, the partial results are formed by the sum (metric by metric) of the evaluation results of every iteration. Figure 5 shows the partial results generation process using the previously presented TR metric.

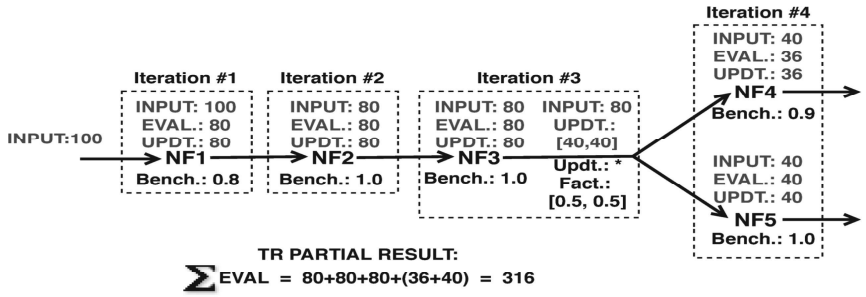

Fig. 5. Partial results generation example

The candidates evaluation procedure, in turn, is responsible for executing a series of processes that transform the multiple partial results into a single suitability index. First, the partial results retrieved in the last procedure are normalized to a common range. In addition to avoiding that the granularity of different metrics creates a biases on the candidates' suitability indexes, the normalization also keeps the SI in a known range of values regardless of the requested evaluation metrics set. To do that, the partial results set regarding each evaluation metric mtc is mapped from the range $[\alpha_{mtc}^{max}, \alpha_{mtc}^{min}]$ to the range $[0, 1]$. Note that α_{mtc} is the set of raw partial results of metric mtc, α_{mtc}^{max} is the highest partial result found for the mtc evaluation of the available candidates, while α_{mtc}^{min} is the lowest result. We employ the technique called Proportion Of Maximum Scaling (POMS) [6] to do this mapping process and create the normalized set of partial results called β_{mtc}.

To reduce the complexity of the suitability index, we designed it as a maximization problem (mono-objective). However, the evaluation metrics can be assigned with a minimization objective in the service request. To circumvent this objective incompatibility, we transform the minimization problem to a maximization problem. It is executed through the complementation of every partial result b in β_{mtc} ($\forall b \in \beta_{mtc} : 1 - b$). Finally, a weighted-sum [8] process is

applied to the pre-processed partial results regarding each particular candidate, thus creating its suitability index. The higher a candidate's SI is, better is the composed service topology considering the requested evaluation metrics, their objectives, and weights. At the end, CUSCO returns the ranked candidates as the solution final output.

4 Case Study

In this section we describe an empirical evaluation of the CUSCO solution for the composition of a HTTP/S-based network service topology. The case study service uses seven different network functions, two of them were developed with the Click Modular Router framework: Protocol/Port Filter - `PPF` and Traffic Classifier - `TC`. The other five functions were developed with Python 3 and the Scapy library: HTTPS Signature Inspector - `HSI`; HTTP Content Inspector - `HCI`; Markup Filter (MF); HTTP/S Intrusion Prevention System - `HIPS`; and Load Balancer - `LB`.

The HTTP/S security service balances HTTP/S requests among clients and available servers. It is tailored to process only HTTP/S traffic, thus everything different from these protocols is dropped by the PPF function. TC recognizes the HTTP and HTTPS requests and forwards them to be processed by different branch segments of the service topology. Functions in these segments search for prohibited signatures in HTTPS packets (HSI) and forbidden content in the HTTP packets (HCI). If search returns positive results, the packets are marked and then dropped by the MF function. Finally, HIPS searches and discards anomalous packets that are probably malicious. Non-malicious packets are forwarded to the LB function and then to the HTTP/S servers. Figure 6 depicts the case study service topology (i) and its SCAG specification (ii).

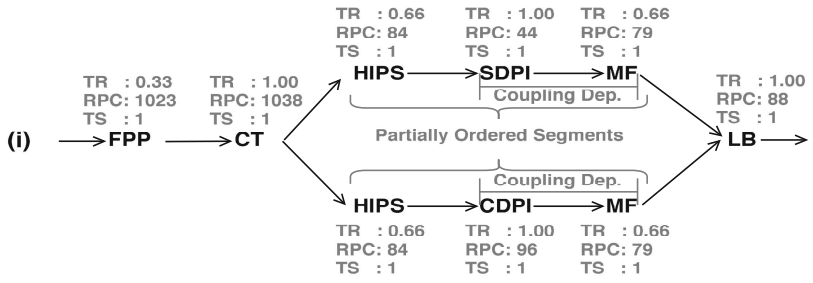

Fig. 6. Case study network service

Observe that the requested service specification has two partially ordered segments, each in a different branch of the topology. The first partially ordered segment includes network functions HSI, MF, and HIPS, while the second includes

functions HCI, MF, and HIPS. The inspector functions (HSI and HCI) are both coupled to the MF. Therefore, the MF must receive traffic from the inspectors. As as example, consider three evaluation metrics forming the objective function: minimization of the traffic ratio (TR), maximization of the HTTP/S request processing capacity (RPC), and minimization of service topology size (TS – number of network function instances). The traffic ratio specifies how much traffic is discarded according to characteristics of the functions and the expected incoming traffic. The http-perf tool (https://www.npmjs.com/package/http-perf) is employed of obtain a profile of the network functions in terms of their request processing capacity. Finally, the topology size is computed by counting the number of instances of each candidate. Without loss of generality, we set up the same weight (*i.e.* 0.3333) for each evaluation metric. The metrics specification (Experiment setup available at: https://github.com/ViniGarcia/NFV-FLERAS/tree/master/CUSCO/Experiments) for each network function is shown in Fig. 6.

After CUSCO processed the required service and evaluation metrics the output consisted of nine candidate of which three were selected due to their results: (i) the candidate with highest suitability index (IN FPP HIPS TC SDPI/CDPI MF LB EN; SI 0.666, TR 0.333, RPC 0, and TS 0.333); (ii) the candidate with the best request processing capacity (IN FPP TC HIPS SDPI MF/HIPS CDPI MF LB EN; SI 0.500, TR 0.167, RPC 0.333, and TS 0); and (iii) the candidate the worst suitability index (IN FPP TC SDPI MF/CDPI MF HIPS LB EN; SI 0.327, TR 0, RPC 0.161, and TS 0.166).

The candidates were validated on two hosts connected on a Gigabit Ethernet, one for clients (8GB RAM DDR3, Core I3 4410U, Ubuntu 14.04, and KVM hypervisor) the other for the network service and HTTP/S servers (8GB RAM DDR3, Core I5 3330, Debian 8, and KVM hypervisor). Each virtual machine was configured with 512MB RAM and a single virtual processing unit, and they were connected with Linux bridges. Clients and servers are in the same /24 network. Four clients were deployed in this scenario, two sending HTTP requests and two sending HTTPS requests. The network service was deployed between the clients and servers and processes all the clients' requests. We used the http-perf tool to create valid HTTP/S requests. Other types of traffic were generated and injected into the system by using the nping tool [7]. The selected candidates were deployed with the minimum number of virtual machines and lasted until legitimate clients have completed 5000 requests.

We used the previously presented setup to conduct tests in three different scenarios: (i) a scenario in which four clients, two making legitimate HTTP and two legitimate HTTPS requests at the maximum feasible rate; (ii) a scenario with both HTTP/S and non-HTTP/S traffic in which two clients make legitimate requests (one for HTTP and another for HTTPS) and the other two send UDP packets (1450 bytes) at a rate of 10Mbps; (iii) a scenario with intensive processing of payload under a low-rate DDoS attack. The last scenario presents two clients sending legitimate requests while the other two fake HTTP/S traffic with anomalous requests of 1450 bytes at a rate of 10Mbps that are not recognizable by the web servers, thus the malicious clients aim to undermine the entire

service as it makes unavailable the network functions that process the packet payload.

Figure 7 presents the results of the HTTP/S requests ratio. Observe that, in scenarios (i) and (ii), results are very similar. It occurs because the system is underloaded and the request ratio does not exceed the individual processing capacity of the network functions. Also, all the HTTP/S requests are non-malicious. The highest request ratio noted in the second scenario is a consequence of the lower number of successful clients, remember that UDP traffic is dropped.

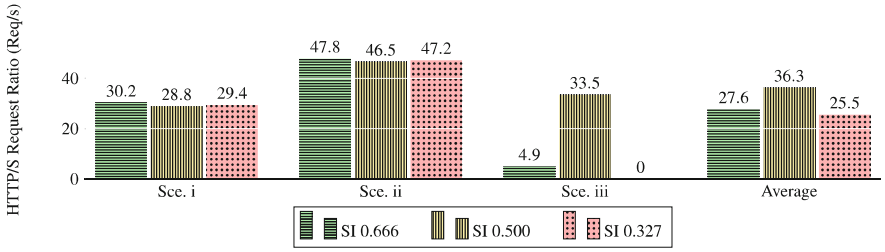

Fig. 7. HTTP/S requests ratio results

Scenario (iii) presents several different results regarding the request ratio. In this scenario, the malicious traffic overloads all the selected candidates. However, due to the duplication of the HIPS function in the first positions of the branch segments (paralleling the processing of malicious traffic) and the high-capacity of the FPP and TC functions, candidate SI 0.500 presented a higher request ratio than the other candidates. Candidate 0.666 completes all the received non-malicious requests, but at a smaller rate. This low request processing rate is a consequence of the existence of a single HIPS function that drops the malicious traffic early in service topology. Finally, candidate SI 0.327 is not able to establish HTTP/S connections. In this case the HIPS function is after the branching structure, thus all malicious traffic is processed by functions SDPI and CDPI that present intense payload processing.

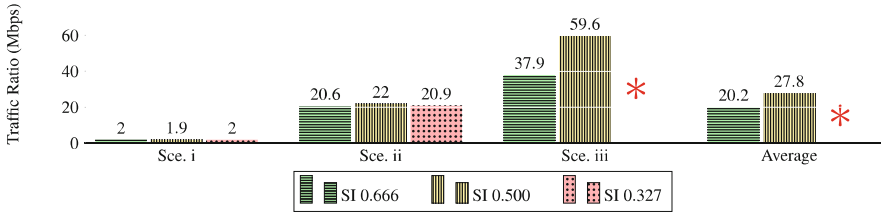

Fig. 8. Traffic ratio results

Figure 8 shows the results for the traffic ratio. Non-malicious HTTP/S requests consist of small packets (60 to 80 bytes). Thus, as in the first scenario

as all requests are non-malicious, the average traffic ratio is low for all candidates. In the second scenario, in turn, the extra traffic from the ingress node to the FPP function (20mbps of UDP traffic) causes the average traffic ratio to increase, but the results is similar for all candidates. These results are due to the fact that in the first scenario the traffic traverses all the service topologies while in the second scenario, the FPP function that drops the invalid traffic is located in the first position of the topologies of all candidates.

In the third scenario, the malicious traffic is dropped in different positions of the selected candidate topologies. Candidate SI 0.327 is supposed to drop the malicious traffic at the first position after the branching segment, thus malicious traffic is processed by all network functions except the LB. Candidate 0.500 drops the malicious traffic at the first positions of the branch segments, therefore it is processed by the FPP and CT functions. In candidate 0.666, in turn, the malicious traffic is processed only by the FPP function and is dropped at the second topology position. The HTTP/S service is denied when candidate SI 0.327 is used due to the processing of malicious traffic by the payload-intensive functions SDPI and CDPI. The other candidates resist to the DDoS. Candidate SI 0.500 forwards the malicious traffic through more network functions, thus increasing its average traffic ratio in the service. Finally, candidate SI 0.666 drops the malicious traffic early and achieves the best average traffic ratio result.

Figure 9 presents the computational resources needed to do the minimal deployment (*i.e.*, minimal required resources without executing any scaling out or scaling up) of the selected candidates. Observe that, as the minimal required computational resources to instantiate a virtual machine are the same regardless of the network function, the measurements in Fig. 9 are directly proportional to the number of virtual instances in each candidate service topology. Thus, minimizing the number of virtual instances will further minimize the amount of computational resources required. Candidate SI 0.500 requires more computational resources, as HIPS and MF functions are in both branch segments. Candidate SI 0.327 also employs dedicated virtual instances, but only for the MF function. The last candidate, SI 0.666, minimizes the computational resource requirements by using common virtual instances for the HIPS and MF functions.

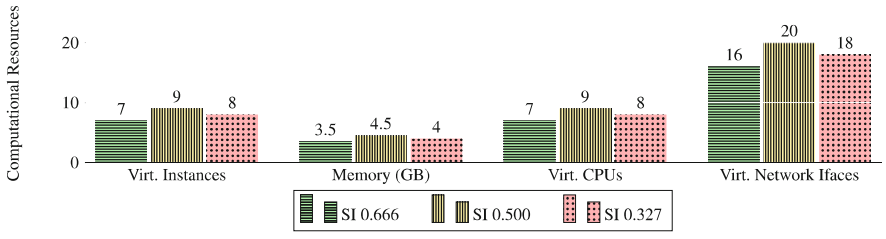

Fig. 9. Computational resource requirements

It is important to notice that the results matched the CUSCO partial results and suitability indexes. Next, we discuss results considering the average RPC in

Fig. 7, TR in Fig. 8, and the "Virtual Instances" in Fig. 9 (TS). First, candidate SI 0.327 achieved the worst average results for both the RPC and TR. Despite requesting less virtual instances when compared to candidate SI 0.500, the other results show that candidate SI 0.327 – as is suggested by its suitability index – is not an adequate option. For the RPC metric, the CUSCO candidate SI 0.500 achieved the best average result in terms of the request ratio. Furthermore, this candidate also presented the highest number of virtual instances in the service topology and an under-optimized average traffic ratio (worse than candidate SI 0.666). At last, also confirming the CUSCO results, candidate SI 0.666 has both a lightweight service topology and presents the lowest average traffic ratio. As discussed earlier, candidate SI 0.666 presents a lower RPC than candidate SI 0.327. However, the optimized location of network functions in candidate SI 0.666 offsets this problem and makes it possible for this service topology to resist the DDoS attack in the third scenario. Therefore, we conclude that candidate SI 0.666 presents the best service topology to provide the network service according to the requested evaluation metrics and weights.

5 Conclusion

In this work, we proposed CUSCO, a flexible and customizable NFV/SFC composing solution. CUSCO employs a dynamic statistical method to process evaluation metrics defined on-demand, thus returning candidate topologies, which can be compared based on a suitability index. We validated CUSCO through the composition of an HTTP/S security service that involved three custom metrics. Experiments are presented in which the service is deployed according to different alternatives which confirm the CUSCO evaluation results.

Future work includes an evaluation of the CUSCO processing overhead caused by a growing number of metrics. Also, we will investigate other methods for defining the priorities of the different evaluation metrics. The idea is to avoid potential bias caused by the network operators that undervalue and/or overvalue specific metrics. Finally, we will investigate how the Pareto fronts can be used to create tiebreak mechanisms to be applied when two candidates present the same suitability index.

References

1. Cappanera, P., Paganelli, F., Paradiso, F.: VNF placement for service chaining in a distributed cloud environment with multiple stakeholders. Comput. Commun. **133**, 24–40 (2019)
2. Dräxler, S., Karl, H.: Specification, composition, and placement of network services with flexible structures. Int. J. Netw. Manage. **27**(2), 1963:1–20 (2017)
3. Gil-Herrera, J., Botero, J.F.: A scalable metaheuristic for service function chain composition. In: Latin-American Conference on Communications, pp. 1–6. IEEE (2017)
4. Herrera, J.G., Botero, J.F.: Resource allocation in NFV: a comprehensive survey. IEEE Trans. Netw. Serv. Manage. **13**(3), 518–532 (2016)

5. Kulkarni, S.G., et al.: NFVnice: dynamic backpressure and scheduling for NFV service chains. In: ACM Special Interest Group on Data Communication, pp. 71–84. ACM (2017)
6. Little, T.: Longitudinal Structural Equation Modeling. Methodology in the Social Sciences Series. Guilford Press, New York (2013)
7. Lyon, G.F.: Nmap Network Scanning: The Official Nmap Project Guide to Network Discovery and Security Scanning. Insecure, Seattle (2009)
8. Marler, R.T., Arora, J.S.: The weighted sum method for multi-objective optimization: new insights. Struct. Multi. Optim. **41**(6), 853–862 (2010)
9. Mehraghdam, S., Keller, M., Karl, H.: Specifying and placing chains of virtual network functions. In: International Conference on Cloud Networking, pp. 7–13. IEEE (2014)
10. ETSI NFVISG: Network functions virtualization: White paper. Technical report, European Telecommunications Standards Institute (2012)
11. Ocampo, A.F., et al.: Optimal service function chain composition in network functions virtualization. In: International Conference on Autonomous Infrastructure, Management and Security, pp. 62–76. Springer (2017)
12. Quinn, P., Nadeau, T.: Problem statement for service function chaining - RFC 7498. Technical report, Internet Engineering Task Force (2015)
13. Wang, Y., et al.: Enabling automatic composition and verification of service function chain. In: IEEE/ACM International Symposium on Quality of Service, pp. 1–5 (2017)
14. Wang, Z., Zhang, J., Huang, T., Liu, Y.: Service function chain composition, placement and assignment in data centers. IEEE Trans. Netw. Serv. Manage. **16**, 1638–1650 (2019)