INTERNATIONAL JOURNAL OF NETWORK MANAGEMENT
Int. J. Network Mgmt 2017; **e1963**:1–20
Published in Wiley Online Library (wileyonlinelibrary.com/journal/nem). DOI: 10.1002/nem1963

Specification, composition, and placement of network services with flexible structures

Sevil Dräxler*† and Holger Karl

Paderborn University, Germany

SUMMARY

Network function virtualization and software-defined networking allow services consisting of virtual network functions to be designed and implemented with great flexibility by facilitating automatic deployments, migrations, and reconfigurations for services and their components. For extended flexibility, we go beyond seeing services as a fixed chain of functions. We define the service structure in a flexible way that enables changing the order of functions in case the functionality of the service is not influenced by this, and propose a YANG data model for expressing this flexibility. Flexible structures allow the network orchestration system to choose the optimal composition of service components that for example gives the best results for placement of services in the network. When number of flexible services and number of components in each service increase, combinatorial explosion limits the practical use of this flexibility. In this paper, we describe a selection heuristic that gives a Pareto set of the possible compositions of a service as well as possible combinations of different services, with respect to different optimization objectives. Moreover, we present a heuristic algorithm for placement of a combination of services, which aims at placing service components along shortest paths that have enough capacity for accommodating the services. By applying these solutions, we show that allowing flexibility in the service structure is feasible.

KEY WORDS: Network Function Virtualization; Service Function Chaining; Service Placement

1. INTRODUCTION

Softwarization of networks introduces new levels of flexibility in service provisioning and orchestration that are, similar to targets like reducing the costs for network operators [1], hard or impossible to achieve in traditional networks. For example, in a software-defined network hosting virtualized composed services, Virtual Network Functions (VNFs) are highly portable and the service delivery chain can be modified without human intervention.

In this context, services are defined as a composition of multiple network functions that should be traversed by network flows in a specific order [2]. The simplest case for such a service is a linear chain of at least one network function between two specific endpoints in the network. Inserting functions that can split network flows over different paths makes the structure of a service more complicated than a simple chain. Such services can be modeled as directed graphs consisting of network functions as nodes and the connections between pairs of network functions as edges of the graph. We refer to these graphs as *service graphs*.

Different VNFs of a service that are responsible for packet processing in the network can influence the traversing flows in different ways, for example, by modifying data rate or splitting flows over different branches. As an example, we assume network flows need to be processed by VNFs f_1

†née Mehraghdam
*Correspondence to: Sevil Dräxler, Paderborn University, Warburger Str. 100, 33098 Paderborn, Germany. E-mail: sevil.draexler@uni-paderborn.de, holger.karl@uni-paderborn.de

(e.g., a video optimizer function) and f_2 (e.g., a firewall) as part of a service. The functionality of these VNFs and their effect on the flows is independent from each other. Therefore, traversing these functions with the order of $f_1 \rightarrow f_2$ or $f_2 \rightarrow f_1$ gives the same *processing result*. In this example, f_2 is a function that can block certain incoming flow and therefore, reduce the data rate of traversing flows. Consequently, the two options for the structure of this service result in different *requirements* for the service: $f_2 \rightarrow f_1$ connection requires less link capacity than $f_1 \rightarrow f_2$. For example, if deploying the service as $f_1 \rightarrow f_2$ is not possible because of insufficient link capacity, the network orchestration system can simply re-order these functions and try deploying $f_2 \rightarrow f_1$ that has lower link capacity requirements.

Similar examples are possible with flexible services that contain branches in their structure. For example, we assume a load balancer is required for distributing the load over 3 instances of a certain function f and all the flows need to go through a firewall before being processed by f. If the orchestration system is allowed to arbitrarily chain the load balancer and the firewall, different results can be achieved, e.g.:

- Distributing the flows over 3 different branches by the load balancer and placing a firewall on each branch results in less traffic on each branch but 3 instances of the firewall are required.
- All flows traversing a single firewall instance before reaching the load balancer requires less firewall instances but the load the firewall need to handle and the load on each branch going out of the load balancer is larger than the previous case.

We use these facts to profit from a new degree of freedom in service composition: under-specifying the structure of a composed service and allowing the order of chaining a subset of VNFs in a service to be determined and modified dynamically, provided that the overall functionality of the service is not impaired. This paper is based on a conference paper [3] we presented at the IEEE NetSoft 2016.

One problem here is that specifying this flexibility in the service structure cannot be done using traditional graph representations in an efficient way. In previous work [4], we have proposed a context-free grammar for flexible service structure specification. In this paper, as the first part of our contributions, we present a YANG [5] model for service structures that enhances our previous model for defining, modifying, and reusing complex and flexible chaining structures for services consisting of multiple VNFs.

The YANG data modeling language is designed to provide a hierarchical model of configuration and runtime data for the Network Configuration Protocol (NETCONF) and supports data instances in different formats, e.g., XML. In a dynamic, softwarized network, where requirements of services and availability of network resources are changing over time, services and the virtual network functions that compose them can also be abstracted as reconfigurable data elements with certain attributes, e.g., required link capacity and computational resources. In this way, *structure* of a service can be modeled as a (re)configurable attribute of the service.

Such a specification can then be used by network orchestration systems for calculating the best placement for new service deployment requests, according to the current state of network and requirements of new and existing services in the network.

If the service structure is fully specified as a set of totally ordered VNFs, the only decision that needs to be taken in the placement step is mapping requested VNFs to network nodes and creating required paths among them. If necessary, previous mappings can also be modified to accommodate new services. However, when deployment requests have a flexible service structure, the placement decision also includes the decision about which chaining option to use for each service for getting the best possible mapping.

Considering that n different VNFs specified with an arbitrary order can be chained together in $n!$ ways, chaining and placement calculations for a set of service deployment requests can quickly result in combinatorial explosion. To improve the practical applicability of specifying services with a flexible structure in an environment where several services need to be managed and orchestrated, the solution space needs to be limited. This, however, is not an straightforward task; because we have to deal with different metrics for two different decisions, namely, chaining the functions together in a specific order and calculating the placement for a set of services. Measurable metrics for different

DOI: 10.1002/nem1963

chaining and combination options for different services *before* calculating the placement are limited to the information available in the service descriptors, for example, resource requirements for individual service components. These metrics are not the same as the metrics of interest *after* the services are placed and deployed in the network, for example, latency of an entire service, and resource utilization in a network that hosts several services.

As our second contribution in this paper, we propose a *selection heuristic* that uses the limited information available before placement for selecting a representative subset of possible options that can potentially result in a close-to-optimal state for the network after placement. The output of the selection heuristic is the input to the placement step, where the service graphs need to be mapped to the network. Our third contribution in this paper is a *placement heuristic* that finds close-to-optimal solutions for the service placement problem with respect to relevant metrics in large-scale distributed networks, e.g., remaining link capacity in the network after placement.

The rest of this paper is organized as follows. In Section 2, we present an overview of related work in specification and placement of services. In Section 3, we present a YANG data model for flexible service specification and in Section 4, we describe our model for handling the problem of composing, combining, and placing services with flexible specifications. We describe our multi-solution heuristic for selecting a subset of possible combinations for services in Section 5 and evaluate our solution in Section 6. Afterwards, we describe our placement heuristic and evaluation results for it in Section 7 and 8, respectively, before concluding the paper in Section 9.

2. RELATED WORK

The model used for specifying the connections and relationships among service components in most ongoing network function virtualization work is based on and similar to the VNF Forwarding Graph (VNFFG) description defined in the ETSI Network Function Virtualization (NFV) management and orchestration document [6], for example, in SONATA [7], UNIFY [8], T-NOVA [9], and OSM [10] projects.

In the cloud computing context, Sun et al. [11] have published a survey of description languages. Our model differs from these existing models and languages in the sense that we focus on a *flexible* description for expressing how the components of a service are chained and composed to set up the service. In a similar approach to flexibility in service descriptions, template-based descriptions of service structures define an application as a generic graph template that can be modified and adapted during and after deployment, e.g., the model proposed by Keller et al. [12]. However, the idea of changing the order of traversing the service components is not captured in these templates.

In the service function chaining context, Moens and Volckaert [13] have published a survey of different modeling strategies. They analyze the trade-off between flexibility and management complexity for service modeling approaches. Obviously, our flexible specification model brings along new complexities and requires orchestration mechanisms that can make us of the new degrees of freedom, which is the main focus of this paper.

Application of YANG/NETCONF in software-defined networking makes it a good candidate for adoption in network function virtualization management and orchestration. For example, the OpenDaylight Service Function Chaining project [14] uses a YANG data model [15] for describing the structure of services and requirements of their components. Our model can be used as an extension to this model to include the description of complex service function chaining structures in a flexible way. An example YANG schema has also been defined in the ETSI NFV management and orchestration document [6] that demonstrates the feasibility of integrating our model in a system compatible with ETSI NFV recommendations.

The placement problem we are investigating is a combination of two NP-hard problems: Supply Chain Network Design (SCND) and Virtual Network Embedding (VNE). SCND [16], a variant of the facility location problem, aims to open a chain of facilities of different types and assign them so that the demand is satisfied and facility or transportation costs are minimized. This corresponds to finding the optimal placement of chained components of a service in our problem. The path creation aspect of our work corresponds to the VNE problem [17], a variant of the multi-commodity

DOI: 10.1002/nem1963

flow problem. These problems allocate virtual network nodes and connect them together through links with constraints while trying to minimize costs. Such problems have been studied in different contexts, for example, Garmehi et al. [18] consider request routing and resource allocation in hybrid content delivery networks.

In our previous work [4], we have modeled the placement process for services consisting of VNFs as a Mixed Integer Quadratically Constrained Program (MIQCP). In several other contributions, the placement problem has been considered specifically in the network function virtualization context. For example, Sahhaf et al. [19] discuss the placement problem taking internal decomposition possibilities for VNFs into account. Mijumbi et al. [20] have designed and evaluated greedy algorithms and a tabu search-based heuristic for mapping and scheduling VNFs. Moens and De Turck [21] present a formal model for the VNF placement problem. Qu et al. [22] propose a delay-aware solution for VNF scheduling and resource allocation for services. Savi et al. [23] propose a solution for service placement that takes processing costs on network nodes into account. Their solution is designed for simple chains of VNFs without branching. Beck and Botero [24] present a heuristic algorithm for services with flexible structures based on our optimization model assumptions [4]. They solve the problems of finding the best composition for VNFs in the service, and finding the best placement for the service in one step. in contrast to our two-step approach, such a combined approach is limited to a specific optimization objective and cannot guarantee the optimality of the selected composition for VNFs.

Complexity of the placement problem highlights the importance of a solution for reducing the possible options for the input of placement step when services can be composed in different ways. Additionally, to enhance the practical usability of the new degree of freedom offered by flexible service specification, fast and efficient algorithms are required for calculating the placement for services.

This paper is an extended version of a conference paper [3] we have presented before. In addition to the YANG model and the selection heuristic in the conference paper, here we present a heuristic algorithm for solving the placement problem and describe the evaluation results of the heuristic in comparison to the optimal placement approach. We have also made slight modifications in the rest of the paper to accommodate the new results, e.g., in the introduction, related work, and conclusion sections.

3. A YANG MODEL FOR SERVICE SPECIFICATION

In Figure 1, we present the tree representation of our YANG module (effectively, the grammar) for flexible specification of complex services, created using pyang [†], a YANG validation tool. To keep the descriptions compact, we have only included structural attributes in the YANG model shown here. Different requirements and specifications of services and their components can be included in the model, e.g., as described in the IETF draft of service function chaining YANG data model [15]. We use `service-function` in our model for referring to the *service function* type defined in this IETF draft, which includes detailed specification of individual VNFs in a service (e.g., compute, storage, and memory requirements).

Within this tree, each leaf node that represents a data element is specified by a name and a type, e.g., `identifier` of type `string`. A list of leaf nodes is specified by `<name>*` and type of the leaves, e.g., the list `best-binding-functions` defines a set of `service-functions`. Lists of non-leaf nodes are specified by `<name>*` and a key written as `[<name>]` that is unique among all items of the list. E.g., in the list of `service-components` each item has a unique `component-identifier`. Description of the list elements follows as children of the node. To express a choice among different options, the node name is written as `(<name>)?` and the possible choices are represented as its child nodes with the format `:(<name>)`. Optional data items are

[†] `https://github.com/mbj4668/pyang`, date accessed: 2016-07-01

DOI: 10.1002/nem1963

```
module: flexible-service-specification
+--rw specification
  +--rw name?  string
  +--rw version?  string
  +--rw included-network-functions*  [network-function-id]
  | +--rw network-function-id  string
  | +--rw network-function-type?  service-function
  | +--rw latency-bound?  uint32
  +--rw input-data-rate?  uint32
  +--rw service-structure
     +--rw service-components*  [component-identifier]
       +--rw component-identifier  string
       +--rw compositions*  [composition-identifier]
         +--rw composition-identifier  string
         +--rw (composition-type)?
           +--:(best-binding)
           | +--rw best-binding-functions*  service-function
           +--:(all-bindings)
           | +--rw all-bindings-functions*  service-function
           +--:(split)
           | +--rw splitter-function  service-function
           | +--rw optional-best-binding*  service-function
           | +--rw outgoing-branches* [branch-id]
           |   +--rw branch-id  uint8
           |   +--rw (branch-type)?
           |     +--:(normal-branch)
           |     | +--rw composition  composition-ref
           |     | +--rw replications?  uint8
           |     +--:(pass)
           |       +--rw string  string
           +--:(function)
           | +--rw single-function  service-function
           +--:(link-to-composition)
             +--rw existing-composition  composition-ref
```

Figure 1. YANG data model

represented as <name>?, e.g., number of `replications` can be optionally specified for a branch if outgoing branches from a splitter function are identical.

In our flexible service specification module, the service specification consists of a list of totally ordered `service-components`. Each `service-components` is a list of at least one `composition` of service functions.

Different types of compositions can be used for defining complex structures:

- `best-binding` composition defines a set of *partially* ordered VNFs.
- `all-bindings` composition defines a set of VNFs can be specified to be chained together in a way that all possible permutations of them are traversable, i.e., a full mesh of paths has to be built among the VNFs.
- `split` composition defines a branching structure for splitting the flows over different branches. This composition consists of:
  - a VNF that can classify and split the flows over different branches,
  - an optional best-binding composition to be traversed before the flows reach the branches,
  - branches that can consist of a single VNF (or service endpoint), a composition of multiple VNFs, or can be an empty branch (`pass`) that can be used for skipping a part of the service structure. In case the branches are identical, they need to be specified only once. The number of required replications can be specified if it is known, if not it can be left open to be specified at deployment time.
- `function` composition defines a single function to be included as a part of the service.
- `link-to-composition` can be used to include an existing composition into a new service description. For this purpose, we define a reference to `composition-identifier` as a new type called `composition-ref` and use it for referring to compositions defined elsewhere. Recursive structures are not allowed in YANG data modeling language and can only be expressed using path references.

With this model, services can be defined from scratch using the available types of functions in the network. Additionally, the network orchestration system can maintain a catalog of pre-composed services that a tenant can request as an standalone service or for using it as part of a more complex

DOI: 10.1002/nem1963

service structure. Examples of YANG schemas for service and function descriptors are shown in the ETSI NFV Management and Orchestration document [6]. For example, the following lines are part of extensions that should be made to an existing schema (based on Figure 1) to allow a list of functions to be treated as a `best-binding` composition.

```
leaf-list best-binding-functions {
  description
    "List of VNFs/endpoints without a specific order of traversing them.";
  type service-function;
  min-elements 1;
}
```

4. PROBLEM DESCRIPTION

In this section, we first describe our assumptions regarding flexible service deployment requests and afterwards, we describe the decision problem for chaining and placing services that arrive at the network with such flexible specifications.

4.1. Service Deployment Requests

Information included in a deployment request in our model corresponds to the service descriptor format defined by ETSI for Network Function Virtualization (NFV) management and orchestration [6], e.g, VNFs included in the service and their resource requirements. Connections and dependencies among them, however, go beyond ETSI descriptors and are specified in a flexible way (i.e., based on our YANG model). Moreover, we assume the input data rate to a service is variable and can change during the lifetime of the service. To support this, we define resource requirements of the included VNFs as a function of the data rate they need to handle. We assume the requirements of a VNF increases linearly with the incoming data rate to this VNF.

To model the influence of each VNF on the service structure, we categorize the VNFs based on their expected impact on network flows into two different types as follows:

1. VNFs that forward incoming flows with a data rate that can be equal to, more than, or less than their data rate when entering the VNF (e.g., as a result of changing data encoding by the VNF),
2. VNFs that split incoming flows over different branches with equal or different data rates (e.g., for load balancing or as a result of traffic classification).

We realize this classification by assigning the expected ratio of outgoing data rate to incoming data rate to each branch leaving a VNF.

Based on this categorization, changing the order of traversing two VNFs in a service can result in the following cases:

- If both VNFs are of type 1 and have a similar ratio of outgoing data rate to incoming data rate, then chaining them together in either order results in the same data rate on the path between them and in the same number of instances for the VNFs and therefore, the same resource requirement in total.
- If both VNFs are of type 1 but have different expected ratios of outgoing data rate to incoming data rate, then chaining them in different orders results in different data rates on the path between them and therefore, different total resource requirements. The number of instances required for each VNF is the same in both cases.
- If at least one of the VNFs is of type 2, then based on the ratio of outgoing to incoming data rates for each VNF, chaining them in different orders can result in different data rates on the paths between them, different number of required VNF instances, different number of paths to connect the instances, and different total resource requirements. For example, if the type 2 VNF is a load balancer and the other VNF is a firewall (of type 1), placing the firewall before the load balancer means one instance of each VNF is needed. But placing the load balancer first could mean having one firewall instance on each of the outgoing branches from the load balancer.

DOI: 10.1002/nem1963

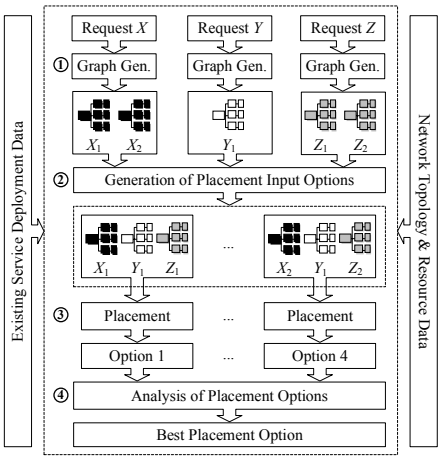


Figure 2. Possible steps for finding the best placement option for a set of example deployment requests

4.2. Service Placement

We assume multiple flexible service requests need to be mapped to the network. Figure 2 shows a possible set of actions that should be performed for finding the best placement option for three requests. In this figure, services X and Z each include two VNFs that can be traversed in an arbitrary order. Therefore, in step ①, two different service graphs can be built for each of them (denoted as X_1, X_2 and Z_1, Z_2, respectively). The request for service Y specifies a total order resulting in one single service graph.

To consider *all* possible ordering options in the placement, the created service graphs go through step ② that creates all possible combinations of different service graphs and passes them to the placement decision algorithm. For the small example in Figure 2, the placement algorithm in step ③ needs to run four times, once for each possible combination of the service graphs. The placement that provides the best values for metrics of interest in the network can be chosen in step ④ for actual deployment.

For a large set of flexible service specifications, number of possible combinations of service graphs can become very large very quickly. In this case, for solving the chaining and placement problem, one end of the spectrum is to explore the whole set of possible combinations. However, in spite of the benefits resulting from flexible specification of service structures, running a (computationally expensive) placement process for every possible combination is not a practical solution. Alternatively, the whole set of steps shown in Figure 2 can be modeled as one large optimization problem that needs to be solved only once. Considering the complexity of the placement optimization problem for one single service, solving such a complicated optimization problem will not be possible in an acceptable time scale either.

The other end of the spectrum is to define a "default" way of chaining VNFs together, irrespective of different functionalities and requirements of services. For example, a rule can be defined in step ① to sort the VNFs in ascending order of the ratio of outgoing data rate to incoming data rate [4]. In this way, we have exactly one service graph for each request; in Figure 2, this corresponds to filtering out X_1 or X_2 (and Z_1 or Z_2). Using the example rule, the resulting graph has the smallest value for average required link capacity. The placement process needs to run only once for the single combination of all these pre-filtered service graphs.

However, choosing one single combination in this way eliminates the variety of optimization options offered by different ways of chaining VNFs together. For example, there might be chaining options that optimize metrics like number of required VNFs, required computational resources for satisfying all deployment requests, number of utilized network nodes, etc. after placement. But they are simply discarded using a single-solution heuristic. Such a solution would only be acceptable if the best possible service graph could be detected already in step ② in Figure 2. But the information

DOI: 10.1002/nem1963

available about the possible combinations in this step does not directly correspond to the metrics of interest after placement that can only be measured and analyzed in step ④.

What we can ideally achieve using the information available before placement is to relate them to the structure and requirements of services by concentrating on multiple metrics and based on that, using a multi-solution heuristic, select a representative subset of possible combinations for services. Then we can use the reduced set as the set of placement input options. This reduces the decision time, by reducing the number of times the placement process needs to be performed, compared to exploring the whole set of options. Moreover, regardless of the optimization objectives used for placement calculation, there is a good chance of finding close-to-optimal solutions; because in the placement input set we have representatives for all possible options.

We describe such a selection heuristic in Section 5, which provides a generic method for selecting a subset of possible combinations of the requests as input for placement. The metrics used for filtering the combinations should be selected intuitively based on the optimization objectives. We have evaluated the heuristic using metrics and objectives that are targeted towards congestion control over nodes and links in the network, as one of possible optimization objectives. However, the solution can easily be generalized to other metrics and objectives.

Once the filtering process is completed, the actual placement can be calculated, e.g., based on our MIQCP formulation [4] for an *optimal* placement, for the selected combinations of the requests with the intended optimization objective. Our MIQCP formulation of the placement decision problem consists of a rather complex model to ensure a realistic decision. Therefore, even if the output of the selection heuristic is one single combination, finding the optimal placement on large-scale networks can take several hours for large request sets with complex structures. That means, in addition to the selection heuristic and to further enhance the decision time, we need a heuristic algorithm for the placement calculation step as well. We present our placement heuristic in Section 7 and describe the evaluation results of the heuristic algorithm in comparison to the optimal placement calculation.

5. SELECTION HEURISTIC

For this heuristic, first of all, we need to compute every possible combination of different service graphs that need to be placed in the network. Although the number of combinations can be large, computing them *without* performing the actual placement is not an expensive task.

For choosing a representative subset of the combinations, we need to extract meaningful metrics out of the information available about the combinations before placement. We pick the following metrics for evaluating different combinations of services.

- Sum of data rates over all virtual links in all service graphs in a combination
- Sum of resource requirements (e.g., an abstract value representing the amount of required compute, memory, and storage resources) over all VNFs in all service graphs in a combination
- Total number of VNFs specified over all service graphs in the combination

This can of course be generalized to any other metric aggregating the information available in service deployment requests that can represent the different options for service structure and requirements.

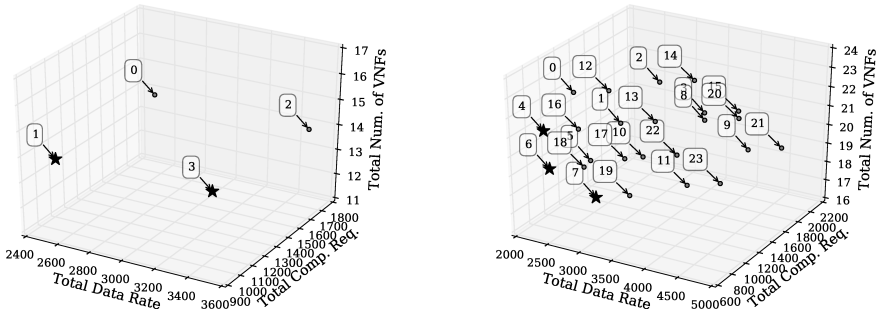
The combinations with the best possible trade-offs among the chosen metrics can then be identified by looking at the Pareto-optimal combinations regarding these metrics. In a scenario where different metrics need to be optimized, Pareto-optimal solutions are the solutions that cannot improve one metric without worsening at least one other metric. The *selected combinations* by our multi-solution heuristic are the combinations in this Pareto set.

To demonstrate how the Pareto sets could look like, we have used two different sets of example service deployment requests consisting of five services (which we refer to as S_0–S_4). Each service has two or three VNFs in it that can be chained together in an arbitrary order. These VNFs need to be traversed between given service endpoints. In each service, one VNF can split incoming flows over three different branches. Input data rates for services and ratio of outgoing data rate to incoming

DOI: 10.1002/nem1963

(a) Chaining option 1 (b) Chaining option 2

Figure 3. Two different options for the structure of an example service, consisting of two VNF types f_1 and f_2 that can be traversed in an arbitrary order between the endpoints



(a) Combinations of S_0 and S_1 (b) Combinations of S_2, S_3, and S_4

Figure 4. Selected combinations for example services

data rate for VNFs are chosen randomly. Resource requirements of each VNF increase linearly with data rate that enters the VNF and the resource requirement for a unit of data rate is also chosen randomly.

Services S_0 and S_1 have a similar structure specification, with different input data rates and different ratios of outgoing data rate to incoming data rate for each included VNF. Their structure is shown in Figure 3. Each of them results in two different service graphs and therefore, they can be combined into four different inputs for the placement process (in step ② of Figure 2). These combinations (numbered from 0 to 3) result in different requirements and different values for our metrics of interest, as shown in Figure 4a. For simplicity and visibility, we assume the computational requirements of VNFs in this example represent the overall resource requirements. The Pareto-optimal combinations (1 and 3) are marked with a star. Combination 1 consists of the chaining option 2 for S_0 and the option 1 for S_1, while combination 3 consists of the option 2 for both S_0 and S_1.

Similarly, as shown in Figure 4b, there are 24 possible combinations for service graphs that can be built for S_2 to S_4, out of which 3 combinations belong to the Pareto set and are selected by the heuristic. Structure of S_3 and S_4 also corresponds to the structure shown in Figure 3 but S_2 consists of 3 VNFs that can be traversed in an arbitrary order, resulting in 6 different chaining options that we do not show here.

Taking the selected combinations instead of the complete set of combinations gives us placement inputs with the best possible trade-offs among the metrics available before the actual placement. In the following section, we evaluate the quality of the selected combinations, by comparing the placement results for these combinations to the set of combinations that optimize the placement objectives (in our case, controlling congestion in network nodes and links).

6. EVALUATION OF SELECTION HEURISTIC

To evaluate how good the selected combinations can represent the set of all possible combinations, we have calculated the placement for different sets of example requests in a total of 450 runs. We follow the steps shown in Figure 2 for the placement process. Each placement run consists

DOI: 10.1002/nem1963

of computing the optimal placement for a set of deployment requests with flexible specifications. During each run, the placement is computed for *all* possible combinations of the service graphs generated for the deployment requests, including the combinations selected by our heuristic. In this way, we can analyze the quality of the solutions achieved using the heuristic among all possible solutions.

The deployment requests follow the model described in Section 4.1 and have similar characteristics as described in Section 5 regarding the services shown in Figure 4a and 4b. We have chosen the structure of the example services based on an IETF Service Function Chaining (SFC) draft on general use cases for SFC [25]. We have used 9 different sets of deployment requests, each including one, two, or three services. In order to have interesting distinctions among different combinations of a service, each service includes at least one VNF of type 2 (as described in Section 4.1).

Sum of input data rates for the requests in a set is the same for all placement runs and is distributed randomly over the requests in the set. Endpoints of the requested service graphs are pinned to random nodes in the network in each run. The underlying network has 12 nodes and 42 directed edges (including self-loops) and is based on the *abilene* network from SNDlib [26]. Among available network instances in this library, we have selected a small one to be able to apply the optimization approach in a reasonable time. Similarly, we have chosen the size and complexity of the deployment requests in a way that running optimization approach for all possible combinations of them is feasible in a reasonable time.

We have used a modified version of the placement optimization problem description from our previous work [4] for calculating the placement of the request sets.

Assuming the network graph $G{=}(V, E)$, we define the following constraints in the placement problem:

$$\forall v{\in}V : \text{NodeUtil}_v \leq \text{MaxNodeUtil}$$

$$\forall (v, v'){\in}E, v{\neq}v' : \text{LinkUtil}_{(v,v')} \leq \text{MaxLinkUtil}$$

where NodeUtil_v and $\text{LinkUtil}_{(v,v')}$ are continuous variables with values between 0 and 1 that show the utilization of node v and link (v, v'), respectively. With similar definitions, MaxNodeUtil and MaxLinkUtil serve as upper bounds for node and link utilization, which we minimize using the objective function, as an attempt to minimize node and link congestion in the network. We use the equally-weighted sum of MaxNodeUtil and MaxLinkUtil as one possible option for such an objective function:

$$\text{minimize} \quad (0.5 \cdot \text{MaxNodeUtil} + 0.5 \cdot \text{MaxLinkUtil})$$

Out of the results of each placement run, for each combination of a set of requests, we calculate the maximum link utilization and maximum node utilization for network nodes. For each set of requests, the *preferred combinations* are the combinations corresponding to the results that belong to the Pareto set regarding these two metrics *after* the actual placement (not to be confused with selected combinations that we had *before* performing the placement).

To evaluate the quality of selected combinations, we need to quantify the differences and relations between the set of selected combinations and the set of preferred combinations. For this, in the following sections, we consider two views:

- Evaluation of preferred combinations that were not selected by our heuristic in each placement run (Figure 5a)
- Evaluation of selected combinations by our heuristic that do not belong to the set of preferred combinations in a placement run (Figure 5b)

Afterwards, we describe the gain in decision time that the heuristic can provide.

DOI: 10.1002/nem1963

Figure 5. Sets of combinations selected by our heuristic and combinations that give the best placement results (preferred combinations) among all possible combinations of services with flexible structures

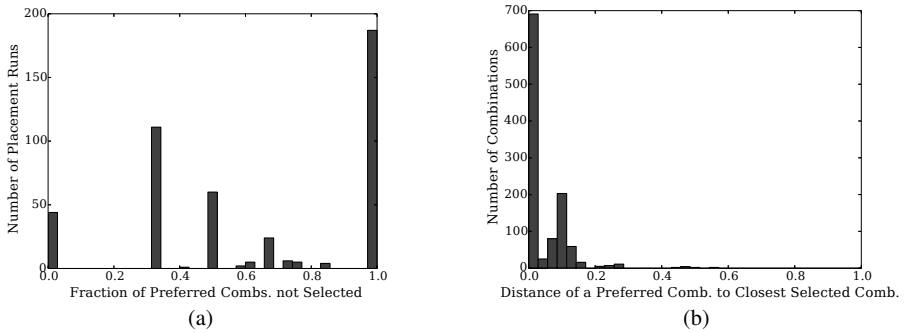


Figure 6. Evaluation of preferred combinations that are not selected by our heuristic

6.1. Preferred Combinations not Selected by Heuristic

Figure 6a shows the histogram of fraction of preferred combinations in placement results that were not selected by the heuristic. For each placement run one value has been calculated as

$$\frac{|P \setminus S|}{|P|},$$

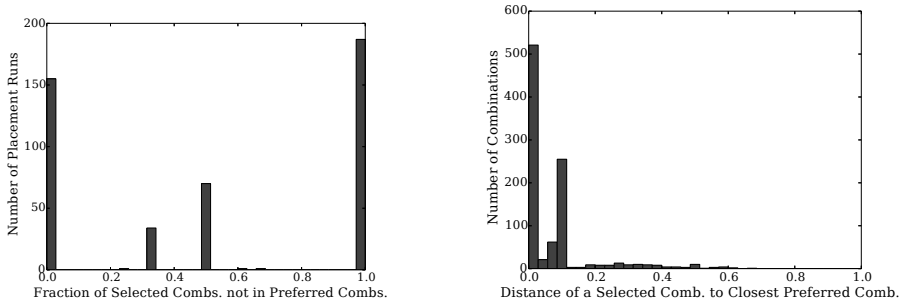
where P is the set of preferred combinations in that run, S is the set of selected combinations by the heuristic, and $|X|$ denotes the cardinality of set X and $\setminus$ denotes the set difference [‡].

The values show that in many simulation runs, none of the preferred combinations are included in the combination set chosen by the heuristic. To evaluate the importance of the combinations that were missed by the heuristic, we calculate the max-norm distance of each preferred combination to the closest selected combination. In a vector space, the max-norm distance of two vectors, also known as the chessboard distance, is the greatest difference of them along any of the dimensions, which can give a meaningful comparison of the quality of two combinations in our case. The max-norm distance of two combinations reflects the difference in the resulting maximum link utilizations or difference in maximum node utilizations (depending on which one is more significant) when these combinations are placed in the network. The largest possible distance between two combinations according to these metrics is 1.

Figure 6b shows the resulting histogram. For the majority of preferred combinations there is at least one selected combination with a distance close to 0. For the remaining combinations, the distance is negligible. The largest recorded distance has a value of around 0.54 and has been recorded for 2 combinations out of around 1100 preferred combinations over all placement runs. That means, in spite of the large number of preferred combinations that are not included in the

[‡] In terms of classification theory or information retrieval, this is the *false negative or miss rate* if we think of our heuristic as a classifier to detect preferred combinations.

DOI: 10.1002/nem1963



(a) (b)

Figure 7. Evaluation of selected combinations that are not among preferred combinations after placement

combination set selected by the heuristic, the variety of possible structures and requirements in the set of all possible combinations is actually captured by the heuristic results.

6.2. Selected Combinations outside Preferred Combinations Set

For the second part of the evaluation, in Figure 7a we show the histogram of fraction of selected combinations that do not belong to the preferred combination set over different placement runs. For each placement run one value[§] has been calculated as

$$\frac{|S \setminus P|}{|S|}.$$

In more than 150 placement runs, all of the combinations selected by our heuristic belong to the preferred combination set, while in close to 200 runs none of the selected combinations are in the preferred combination set. However, Figure 7b shows that for the majority of selected combinations, the max-norm distance to the closest preferred combination is close to 0 and negligible. The largest distance has a value of around 0.66, which is recorded for one combination out of around 970 selected combinations over all placement runs. That means, the combinations selected by our heuristic can closely represent the preferred results in these placement runs.

6.3. Gain in Decision Time

Figure 8 shows the ratio of combinations selected by our heuristic to the total number of combinations over different placement runs. As illustrated in Figure 2, time to reach a final placement decision depends on the number of times the placement needs to be calculated, which is in turn determined by the number of selected combinations. Using this heuristic, the placement needs to be calculated less than half as often as the case where no heuristic is applied. However, most of our service deployment requests were small requests, similar to the case shown in Figure 4a. Comparing this case to Figure 4b shows that the larger service sets are, the larger possible combinations are, and the ratio of selected combinations to all tends to decrease. That means, for large sets of flexible services, our multi-solution can select combinations that can result in optimal or close-to-optimal solutions after placement in a significantly less time compared to the option of exploring all possible combinations.

[§]The *false discovery rate* in terms of classification theory.

DOI: 10.1002/nem1963

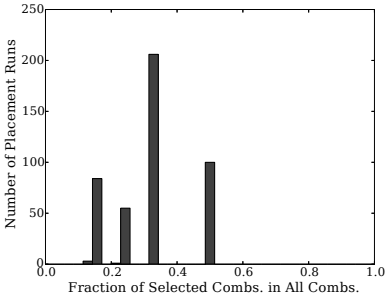


Figure 8. Fraction of selected combinations among all combinations over all placement runs

7. PLACEMENT HEURISTIC

In this section, we describe a heuristic algorithm that can very quickly find a close-to-optimal solution for the network service placement problem. As described in Section 2, this is a complex problem and finding an optimal solutions for a large number of services on a large-scale network is practically not feasible.

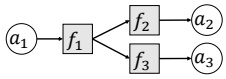
While the objective function in an optimization approach can be replaced to optimize the values for different metrics as necessary, for the heuristic algorithm we need to select the metrics to be optimized beforehand as this drives the decision process of the algorithm. Our placement model is based on the assumption that the substrate network is a geographically distributed network, e.g., a large-scale telco operator's network, with multiple data and compute centers connected to each other. In these networks, applications like video streaming and file sharing are increasingly taking up link capacities and need low-latency paths. Moreover, in such scenarios it is important to place the services in a way that enough capacity is left on the network links to satisfy larger number of subsequent requests as well as requests with large data rates.

Considering there requirements, we have designed a heuristic algorithm that calculates the placement for a set of service graphs in a way that the traffic between the endpoints of a service is routed through the shortest path whose bottleneck link has just enough capacity for the requirements of the service (smallest fit first). The bottleneck link on a path is the link with the smallest capacity along this path. Among different paths with equal lengths that can carry the required amount of traffic, the path with the smallest bottleneck is selected to leave paths with more capacity for serving other, possibly larger requests. Similarly, among different paths with equal bottleneck capacity, the path with the least number of hops is preferred. A path is accepted for placement only if the nodes along this path have enough capacity to host all VNFs of the service and the latency of the path matches the end-to-end latency requirements of the corresponding service.

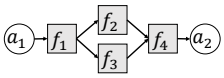
For calculating the paths, we use one of the variations of the bottleneck shortest paths problems, an algorithm to solve the *single-source shortest paths for all flows problem* (SSSP-AF) by Shinn and Takaoka [27]. In this problem, it is assumed that a set of flows with different data rates are given as set W and the network network links have limited capacities. For every flow with a data rate $w \in W$, the goal is to find the shortest paths (with respect to number of hops) from a source node to all other nodes in the network such that the paths can carry flows with data rates of up to w. The output of the algorithm is a set of (d, w) tuples for each destination node, where d is the number of hops in the shortest path that can carry flows with data rates up to w. This algorithm has a complexity of $\mathcal{O}(mn)$, where m is the number of links and n is the number of nodes in the substrate network graph [27].

The input and basic assumptions of the heuristic algorithm are identical to the optimization model [4] and the assumptions described in Section 4. For example, for the heuristic design we assume that the service deployment requests include an (exact or estimated) upper bound for the input data rates at the starting points of the services, and the input data rates to all other VNFs in the services can be calculated based on given expected ratios for the outgoing to incoming data rate for each VNF. Moreover, we assume the start and end point locations of service flows are given. Such a

DOI: 10.1002/nem1963



(a) Two different end points: a_2 and a_3



(b) Two simple chains between its start and end points: $a_1 \to f_1 \to f_2 \to f_4 \to a_2$ and $a_1 \to f_1 \to f_3 \to f_4 \to a_2$

Figure 9. Example service graphs

Algorithm 1 Heuristic algorithm for placement of a combination of multiple requests

```
C ← a combination of multiple annotated service graphs
G ← annotated substrate network graph
function PLACECOMBINATION(C, G)
    A ← sorted list of pairs of start and end points of services
    for (a_s, a_e) ∈ A do
        chains(a_s, a_e) ← sorted list of simple chains between start and end points a_s and a_e
    for (a_s, a_e) ∈ A do
        for c ∈ chains(a_s, a_e) do                          ▷ Every chain c is a subgraph of C
            PLACECHAIN(c, G)
            if placement successful then
                if latency of the created path ≤ latency bound between a_s and a_e then
                    placement for c is valid

function PLACECHAIN(c, G)
    R ← c                                                    ▷ Remaining pairs of VNFs to be placed in the chain
    D ← data rate between each VNF pair                      ▷ Extracted from the annotated chain c
    for (f_1, f_2) ∈ c do
        p_s ← location of f_1                                ▷ Current location
        if f_2 is not already placed then
            if requirements of f_2 can be satisfied by available resources on p_s then
                place f_2 on p_s and update G
            else
                d ← max_{∀(x,y)∈R}(D(x, y))                  ▷ Largest data rate over pairs of VNFs to be placed
                p_e ← location of first VNF in c after f_2 that has already been placed
                P ← GETPATH(p_s, p_e, d, G)                  ▷ Ordered set of nodes in the path
                while there are unexplored nodes on P do
                    v ← next node on P
                    if requirements of f_2 ≤ available resources on v then
                        place f_2 on v and update G
                        break                                ▷ Placement of f_2 done, stop iterating over P
                if there are no more nodes to explore on P then
                    return c cannot be placed                ▷ No node along P can host f_2, placement of c failed
            R ← R \ (f_1, f_2)                               ▷ Remove (f_1, f_2) from remaining pairs of VNFs to be placed
    return placement results and updated G
```

start point can be, for example, one of the network nodes where the requests for a group of end users in a specific geographical location enter the network. An end point can be, for example, the back-end server of the application that is already placed and used by other instances of the service, or a physical network function with a fixed location that needs to be used along the virtualized network functions.

An overview of the steps of this heuristic solution can be found in Algorithm 1. The input to this algorithm is a set of service graphs that need to be mapped to the network (e.g., one of the selected combinations given by the selection heuristic in Section 5). The combination is a (disconnected) graph annotated with data rates of the logical links between pairs of VNFs, requirements of the VNFs (e.g., given as tuples of computation, memory, and storage requirements), the end-to-end latency that can be tolerated for each service, and the location of the start and end points of services in the network. The substrate network graph is also given, annotated with capacity and latency of network links, and capacity of nodes (e.g., in terms of available computation, memory, and storage resources).

Services might have different end points (e.g., Figure 9a) and different branches between one pair of start and end points within them (e.g., Figure 9b). For such services, we find and store

DOI: 10.1002/nem1963

every simple *chain* of VNFs between every pair of start and end points in the service. A chain is a subgraph of the service graph with a linear structure, which starts at the start point of the service and ends at one of the end points of the service. Each of the chains in a service might have different requirements, e.g., different amount of data rate over their edges, so the placement of them needs to be prioritized and regulated.

In line 4 of Algorithm 1, pairs of start and end points (a_s, a_e) of different services are sorted in decreasing order according to the sum of data rates over all virtual links between them and stored in A. Similarly, if there are multiple simple chains between one start and end point, in lines 5 and 6, the simple chains are sorted in decreasing order according to the sum of data rates over all virtual links among them and stored as a list called chains(a_s, a_e) for every (a_s, a_e) in A. This ordering ensures that the chains with higher data rates are placed first and have a higher chance of getting shorter paths.

Every chain is an ordered list of pairs of VNFs (f_1, f_2) (including the start and end point of the chain), such that f_1 and f_2 are nodes of the combined graph C, (f_1, f_2) is a virtual link in graph C, and by following the pairs in a chain in the given order we can get from the starting point of that chain to the end point of the chain after traversing all VNFs in that chain.

We start the placement with the *heaviest* pair of start and end points, the pair that has the largest sum of data rates over its virtual links, and if there are multiple simple chains between them, we start with the *heaviest* chain. In line 9, the PLACECHAIN function is called, which calculates the placement for the input chain c on the substrate network graph G.

If the placement is successful, the results are returned together with the updated network graph G (e.g., with less capacity on its nodes and links after accommodating the placed chain). If the latency requirements of the placed chain are met, the placement is accepted as a valid placement. For simplicity, this algorithm does not include any backtracking steps in case the placement is unsuccessful or invalid. Implementing the backtracking using different path options provided by the SSSP-AF algorithm [27] is straightforward.

The PLACECHAIN is described starting from line 13. Within this function, we iterate over the pairs of VNFs (f_1, f_2) (virtual links in the chain) and place them one by one. As the start point of the chain is also a part of the chain, while iterating over these pairs, in the simplest cases, f_1 is already mapped to a location in the network, and we try to find the location for f_2. For this, in line 19 and 20, similar to the behavior of the placement optimizer, we first check the feasibility of placing f_2 on the same node where f_1 was placed. If that is not possible, in line 24, we calculate the shortest path towards the end point that has enough capacity for the largest data rate over the remaining parts of the chain to be placed, based on the SSSP-AF algorithm.

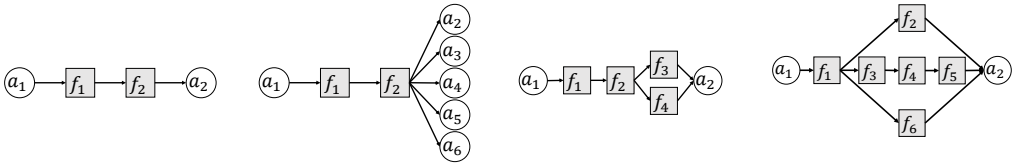
As an input for path calculation using the SSSP-AF algorithm, in line 22 we find the largest data rate over the pairs of VNFs that still need to be placed.

SSSP-AF calculates the paths from a given node towards all other nodes in the network, out of which we only need the paths towards one specific end point. Function GETPATH processes the output from SSSP-AF to extract the path P towards this end point as an ordered list of the network links belonging to this path.

The end point p_e used by GETPATH for calculating P is equal to the location of a_e (end point of the chain) if the current chain has no overlaps with another chain that has already been placed in the network. Figure 9b shows an example of such an overlap. We assume the simple chain $a_1 \rightarrow f_1 \rightarrow f_2 \rightarrow f_4 \rightarrow a_2$ is the heavier chain and needs to be placed before the other chain between a_1 and a_2. For placing the first chain, for all VNF pairs in it, p_e is set to the location of a_2 and GETPATH is called to find the shortest path between the current node and the location of the end point of the chain. However, while placing the second chain $a_1 \rightarrow f_1 \rightarrow f_3 \rightarrow f_4 \rightarrow a_2$, function f_4 has already been placed, so in line 23, p_e is set to the location of f_4 and GETPATH will be called to find the shortest path towards that node. Moreover, to avoid calculating the placement more than once for functions like f_1 and f_4 that appear on multiple chains of a service, in line 18 we check if the function has already been mapped to a node.

Once a VNF is placed in the network, for placing the next VNF in the chain, we do not rely on the previously calculated shortest path towards the end; instead, we recalculate the path for every

DOI: 10.1002/nem1963



(a) (b) (c) (d)

Figure 10. Structure of service graphs used for evaluations

new (f_1, f_2) to be placed to make use of the opportunity that once the virtual link with the largest data rate on the chain has been mapped to the network, the subsequent VNFs can be placed along a path that might be even shorter than the one initially calculated.

In the following section, we evaluate the performance and running time of this heuristic algorithm.

8. EVALUATION OF PLACEMENT HEURISTIC

As described in Section 7, we have designed the placement heuristic in a way that services are placed on network nodes along short paths with acceptable latencies and just enough link capacity. We evaluate this heuristic algorithm against optimal placement results obtained using an optimization approach described in our previous work [4].

For services with flexible structures, the placement heuristic algorithm is used for placing the selected combinations: among different options for the arbitrary chaining possibilities in the service structure, some are selected and passed one by one to the placement step as a fixed and explicitly defined graph. Therefore, the services we have used for evaluating the placement heuristic do not include any arbitrary chaining options.

We have used 5 different sets of example service requests that include simple chains as well as more complex branched structures. As in the evaluation of the selection heuristic, the structure of the example services is based on the IETF Service Function Chaining (SFC) draft on general use cases for SFC [25]. This document is one of the rare resources where examples of possible service structures are illustrated. Sets of services are selected as follows, in increasing complexity, based on the time required by the placement optimizer to find a solution for them:

- *Set 1* includes 4 simple chains each having a structure as shown in Figure 10a.
- *Set 2* includes 3 services, one having a structure as shown in Figure 10b and two having a structure as shown in Figure 10c.
- *Set 3* includes 8 simple chains each with a structure as shown in Figure 10a, making this set similar to *Set 1*.
- *Set 4* includes 7 services in total, four simple chains like *Set 1* and three services like in *Set 2*.
- *Set 5* includes 3 services with a structure as shown in Figure 10d.

These sets include services with different structures, i.e., simple chains of functions, services with converging branches, and services with independent, diverging branches. The VNFs used in the services can split incoming flows over different outgoing branches, increase or decrease data rates of incoming flows, or forward them without modifying the data rate. Combination of these cases represents what a placement algorithm would need to deal with, highlighting various aspects of our solution, e.g., making sure the flows converge towards the required functions, the right flow ends up in the right endpoint, etc.

For each of these sets, we have performed 300 placement runs. Each run uses a new random seed for setting up the input. In each run, we have calculated placement once using the heuristic algorithm and once using the optimization approach.

Similar to the evaluation setup in Section 6, the *abilene* network from SNDlib [26] is used as the substrate network. Start and end points of the service graphs are mapped to random nodes in the

DOI: 10.1002/nem1963

network in each run. For one set, sum of input data rates for the services is always the same for all runs, with each service getting a new, randomly assigned share of the total data rate as input in each run. In different runs different amounts of data rate need to be routed among different network nodes. We have used this approach to create enough variation in the amount and sources of the load in the network, while keeping the total input load in a fixed level to reduce cases where the mapping is infeasible and no insight is provided for comparing the optimization approach to the heuristic approach.

We have chosen the following objective function for the optimization approach:

$$\text{maximize} \sum_{(v,v')\in E, v\neq v'} \text{remcap}_{v,v'}$$

where E is the set of links in the substrate network and $\text{remcap}_{v,v'}(\forall(v,v') \in E)$ shows the remaining data rate on every link (v, v') after the placement is calculated. The value of $\text{remcap}_{v,v'}$ is calculated by subtracting the data rate of every flow that passes (v, v') from the capacity of the link. We exclude the internal links of network nodes (self-loops in the network graph) from the objective function and maximize $\text{remcap}_{v,v'}$ only for those links $(v, v') \in E$ where $v \neq v'$. In this way, we force the the placement to use these internal links more than other links. This results in consecutive VNFs in a service being mapped to the same network node as long as there is enough capacity on the current node to host the next VNF (also taking other constraints of the optimization model [4] into account).

The placement solution that is computed using this objective function has the maximum possible value for mean remaining capacity over all network links, excluding the self-loops. Therefore, as a first step to compare the results of the heuristic algorithm to the optimal results, in Figure 11a we show the comparison between the mean remaining capacity over all network links for each of the service sets based on the results of all placement runs. Results of the heuristic algorithm, with respect to the metric that is optimized by the optimization approach, are very similar and in some cases almost identical to the optimal results. The largest difference between the results of the heuristic algorithm and the optimization approach is around 5% and belongs to the results of *Set 1*. The plots (Figure 11a–11e) include confidence intervals at 95% of confidence level.

To highlight the differences better, we ignore the unused network links and in Figure 11b we show the comparison between the mean remaining capacity only over those links that were used for mapping the services in each run.

The largest difference between the results is around 22% and can be seen in the results of *Set 1*. The optimization approach can handle the placement of this set much better than the heuristic algorithm, partly due to the type of the VNFs used in the services in this set. These services consist of simple chains of VNFs, and one of the VNFs in each chain has a ratio of outgoing to incoming data rate larger than 1, so the input data rate to the services increases somewhere in the middle of the chains. The optimization approach obviously tries to map the two VNFs with such a large data rate between them into one node. By mapping the link between them to an internal link in the node (a self-loop), this part of the flow does not consume capacity on inter-node links. The heuristic algorithm, in contrast, cannot foresee this increase in the data rate and simply places the VNFs into nodes with enough capacity along the shortest path that can push through the maximum data rate over the chain.

The same effect can be observed in other sets as well. However, the different between the results is influenced by other parameters as well. For example, the services in *Set 3* have the same structure as *Set 1* (simple chain) and include some VNFs increase the data rate. The difference of results there is smaller than the difference for *Set 1*, because number of services in *Set 3* is twice the number of services in *Set 1*; that means, input data rate to each service is smaller in *Set 3* and hence, data rate of the flow after leaving the VNF that increases the data rate is smaller and has a smaller effect than it has in the case of *Set 1*.

The difference of heuristic results to optimal results is also affected by the fact that services with complex structures are broken into simple chains and the chains are mapped to the network one by one using the heuristic. This is reflected, for instance, in the placement results for *Set 2* and

DOI: 10.1002/nem1963

Set 5, which consist of branching services. In such services, location of the VNFs that appear in more than one simple chain in the service structure (e.g., f_1 in Figure 10d, which is a part of all 3 simple chains between a_1 and a_2) is determined only based on the first chain that is placed. The optimization approach, in contrast, considers the whole service at the same time and can find better a better mapping.

We also evaluate the *minimum* link capacity that remains in the network after placement, over the links that are used for the services. The heuristic algorithm always selects the path with the smallest bottleneck value among all the paths that have enough capacity for the data rate of the service. As shown in Figure 11c, compared to the results of the optimization algorithm this behavior does not cause much higher chances of congesting network links. The largest difference is around 52% and again belongs to *Set 1* as described before. However, for complex services like in *Set 2* and *Set 5*, the heuristic can get as close as 90% to the behavior of the optimization approach regarding this metric.

Although none of our placement approaches explicitly attempts to optimize usage of resources on network *nodes*, in Figure 11d we show the remaining capacity on the most congested network node after the placement. There is no significant difference between the behavior of the heuristic algorithm and the optimization approach in this regard.

Difference of running times between the two algorithms is shown in Figure 11e, on a logarithmic scale. *Set 5* is the most complex input for the optimization approach among our test sets. Finding a solution for this set requires around 31 minutes on average in our test environment, using the Gurobi[¶] solver on a machine with Intel X6560 CPUs running at 2.67 GHz. On the same machine, the heuristic can find a solution in around 69 milliseconds.

As described in Section 7, no backtracking step is included in the heuristic algorithm that would, for example, try another path in case a chain cannot be placed along the first suitable path found by the SSSP-AF algorithm. Therefore, we show a comparison of the success ratio between the algorithms in Figure 11f. We consider a run as successful if a placement can be calculated for the complete set of input services in that run.

A failure ratio of around 35% can be observed for *Set 2*. Resource requirements of VNFs increase with the incoming data rate. As the input data rates and start and end locations are assigned randomly in each run, requirements of VNFs can differ greatly over different runs. Therefore, in some cases the placement is simply not feasible as the network does not have enough resources to host the service. Because of the branched structure and outgoing to incoming ratios of individual VNFs in *Set 2*, this effect is stronger for this input set. Even the optimization algorithm has not been able to find a solution for all instances of this set. The results can be improved, for example, by trying the second-best path when the placement of a chain fails. For other sets the success ratio is acceptable. As every network node in our model is a large data center, in reality it will likely have enough capacity for hosting VNFs. Therefore, as long as there is a path in the network that has enough capacity to route the required traffic between the start and end point of a chain, the heuristic algorithm will find this path and place the VNFs on the nodes along this path.

9. CONCLUSION

The YANG model presented in this paper provides a powerful tool for flexible specification of complex service structures. It extends the structural aspects of existing YANG definitions for service function chaining that are under development, e.g., as part of the OpenDaylight service function chaining project.

Our *selection* heuristic produces a number of combinations for the possible service graphs that can be built for a set of requests with flexible specifications. The output is the Pareto-optimal points regarding different metrics that represent possible variations in the structure and requirements of a flexibly-specified service. These variations cover link capacity requirement, virtual resource requirement, and number of required instances for VNFs in the service.

[¶] `http://www.gurobi.com`, date accessed: 2016-07-01

DOI: 10.1002/nem1963

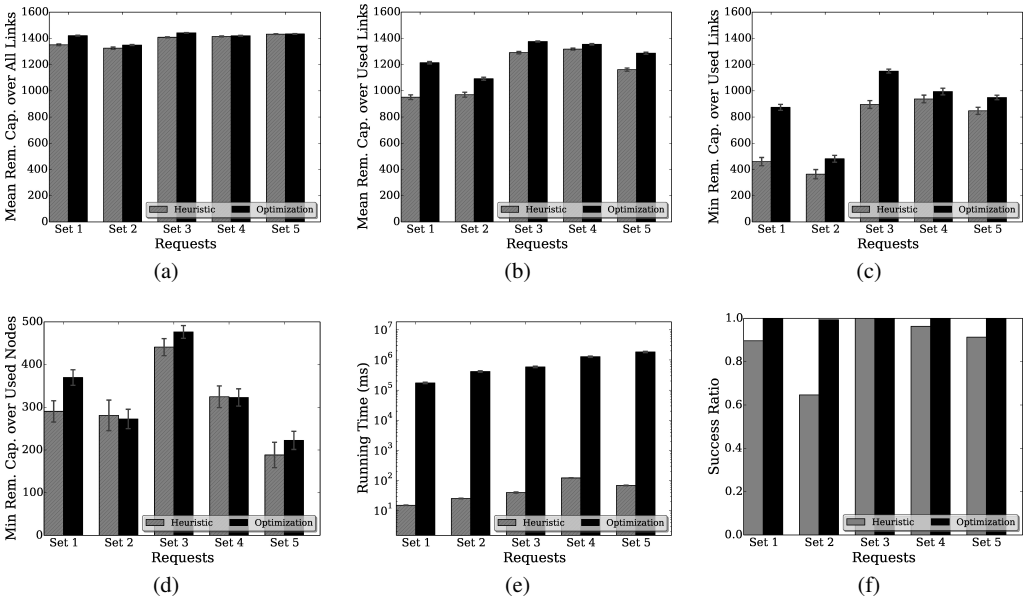


Figure 11. Evaluation results for placement heuristic

We have evaluated the set of selected combinations using a placement optimization approach that is aimed towards congestion control in network nodes and links. However, the selection criteria is generic and flexible and can be used with any other metric and optimization objective. Our evaluations show that the set of selected combinations can closely represent the Pareto set of the combinations with respect to the values of node and link utilization after performing the placement. For service deployment request sets with multiple possible combinations, our selection heuristic can reduce the decision time by eliminating at least half of the options. The resulting gain in decision time tends to increase for large request sets with a large number of combinations.

Results of the selection heuristic can be used in different ways. For example, one can try the placement for all selected combinations and select the best option according to the results. Another interesting possibility would be to change the structure of services for adapting the deployments to the network state. That means, it is possible to categorize the selected combinations according to specific metrics of interest and use different combinations over time. For example, one can do the initial placement using a combination with the lowest resource requirements and switch to the one with the lowest link capacity requirement if a high link utilization is detected.

Our *placement* heuristic provides quick and close-to-optimal solutions for mapping services to the network. This algorithm can place services with simple or complex structures. It maps the VNFs of a service to nodes with enough capacity along the shortest path with the smallest bottleneck value, just enough for carrying the data rate of the service. We have evaluated this algorithm in comparison to an optimization approach that maximizes the mean remaining data rate on network links. Heuristic results show a maximum of 5% deviation from optimal results, with respect to this objective.

Our selection and placement heuristics can be applied jointly to place services with flexible structures. A large portion of possible combinations can be eliminated, leaving a representative set of combinations that can produce optimal or close-to-optimal results. Afterwards, by applying the placement heuristic instead of optimally placing the selected combinations, the decision time is further improved. Using these heuristics and with a negligible deviation from optimal solution, our evaluations show that flexible structures are feasible for complex services.

DOI: 10.1002/nem1963

ACKNOWLEDGMENT

This work has been performed in the framework of the SONATA project, funded by the European Commission under Grant number 671517 through the Horizon 2020 and 5G-PPP programs. This work is partially supported by the German Research Foundation (DFG) within the Collaborative Research Center On-The-Fly Computing (SFB 901) and the International Graduate School "Dynamic Intelligent Systems".

REFERENCES

1. Naudts B, Kind M, Verbrugge S, Colle D, Pickavet M. How can a mobile service provider reduce costs with software-defined networking? *International Journal of Network Management* 2016; **26**(1):56–72, doi:10.1002/nem.1919.
2. ETSI NFV ISG. GS NFV 003 V1.1.1 Network Function Virtualisation (NFV); Terminology for Main Concepts in NFV Oct 2013.
3. Mehraghdam S, Karl H. Placement of Services with Flexible Structures Specified by a YANG Data Model. *IEEE 2nd Conference on Network Softwarization (NetSoft)*, 2016.
4. Mehraghdam S, Keller M, Karl H. Specifying and Placing Chains of Virtual Network Functions. *IEEE 3rd International Conference on Cloud Networking (CloudNet)*, 2014, doi:10.1109/CloudNet.2014.6968961.
5. Bjorklund M. YANG – A Data Modeling Language for the Network Configuration Protocol (NETCONF). RFC 6020 (Standards Track) Oct 2010.
6. ETSI NFV ISG. GS NFV-MAN 001 V1.1.1 Network Function Virtualisation (NFV); Management and Orchestration Dec 2014.
7. SONATA. `http://sonata-nfv.eu`. Date accessed: 2016-07-01.
8. UNIFY. `https://www.fp7-unify.eu`. Date accessed: 2016-07-01.
9. T-NOVA. `http://www.t-nova.eu`. Date accessed: 2016-07-01.
10. OSM. `https://osm.etsi.org`. Date accessed: 2016-07-01.
11. Sun L, Dong H, Ashraf J. Survey of Service Description Languages and Their Issues in Cloud Computing. *IEEE 8th International Conference on Semantics, Knowledge and Grids (SKG)*, 2012, doi:10.1109/SKG.2012.49.
12. Keller M, Robbert C, Karl H. Template Embedding: Using Application Architecture to Allocate Resources in Distributed Clouds. *IEEE/ACM 7th International Conference on Utility and Cloud Computing (UCC)*, 2014, doi:10.1109/UCC.2014.49.
13. Moens H, Volckaert B. Comparing topology and stream based strategies for modeling service function chains. *IEEE 2nd Conference on Network Softwarization (NetSoft)*, 2016.
14. OpenDaylight. `https://www.opendaylight.org`. Date accessed: 2016-07-01.
15. Penno R, Quinn P, Zhou D, Li J. Yang Data Model for Service Function Chaining. *Internet-Draft draft-penno-sfc-yang-13*, IETF Secretariat Mar 2015.
16. Melo M, Nickel S, Saldanha-da Gama F. Facility Location and Supply Chain Management – A Review. *European Journal of Operational Research* 7 2009; **196**:401–412, doi:10.1016/j.ejor.2008.05.007.
17. Fischer A, Botero JF, Beck MT, de Meer H, Hesselbach X. Virtual Network Embedding: A Survey. *IEEE Communications Surveys & Tutorials* 2013; **15**(4):1888–1906, doi:10.1109/SURV.2013.013013.00155.
18. Garmehi M, Analoui M, Pathan M, Buyya R. An Economic Mechanism for Request Routing and Resource Allocation in Hybrid CDN-P2P Networks. *International Journal of Network Management* 2015; **25**(6):375–393, doi:10.1002/nem.1891.
19. Sahhaf S, Tavernier W, Colle D, Pickavet M. Network Service Chaining with Efficient Network Function Mapping Based on Service Decompositions. *IEEE 1st Conference on Network Softwarization (NetSoft)*, 2015, doi:10.1109/NETSOFT.2015.7116126.
20. Mijumbi R, Serrat J, Gorricho JL, Bouten N, De Turck F, Davy S. Design and Evaluation of Algorithms for Mapping and Scheduling of Virtual Network Functions. *IEEE 1st Conference on Network Softwarization (NetSoft)*, 2015, doi:10.1109/NETSOFT.2015.7116120.
21. Moens H, De Turck F. VNF-P: A Model for Efficient Placement of Virtualized Network Functions. *IEEE 10th Conference on Network and Service Management (CNSM)*, 2014, doi:10.1109/CNSM.2014.7014205.
22. Qu L, Assi C, Shaban K. Delay-Aware Scheduling and Resource Optimization with Network Function Virtualization. *IEEE Transactions on Communications* 2016; **PP**(99), doi:10.1109/TCOMM.2016.2580150.
23. Savi M, Tornatore M, Verticale G. Impact of Processing Costs on Service Chain Placement in Network Functions Virtualization. *IEEE 1st Conference on Network Function Virtualization and Software Defined Network (NFV-SDN)*, 2015, doi:10.1109/NFV-SDN.2015.7387426.
24. Beck MT, Botero JF. Coordinated Allocation of Service Function Chains. *IEEE Conference Global Communications (GLOBECOM)*, 2015, doi:10.1109/GLOCOM.2015.7417401.
25. Liu W, Li H, Huang O, Boucadair M, Leymann N, Fu Q, Sun Q, Pham C, Huang C, Zhu J, *et al.*. Service Function Chaining (SFC) General Use Cases. *Internet-Draft draft-liu-sfc-use-cases-08*, IETF Secretariat Sep 2014.
26. Orlowski S, Pióro M, Tomaszewski A, Wessäly R. SNDlib 1.0–Survivable Network Design Library. *3rd International Network Optimization Conference (INOC)*, 2007, doi:10.1002/net.20371.
27. Shinn TW, Takaoka T. Variations on the Bottleneck Paths Problem. *Theoretical Computer Science* 2015; **575**:10 – 16, doi:10.1016/j.tcs.2014.10.049. Special Issue on Algorithms and Computation.