Specifying and Placing Chains of Virtual Network Functions

Sevil Mehraghdam
University of Paderborn
33098 Paderborn, Germany
Email: s.mehraghdam@uni-paderborn.de

Matthias Keller
University of Paderborn
33098 Paderborn, Germany
Email: mkeller@uni-paderborn.de

Holger Karl
University of Paderborn
33098 Paderborn, Germany
Email: holger.karl@uni-paderborn.de

***Abstract*—Network appliances perform different functions on network flows and constitute an important part of an operator's network. Normally, a set of chained network functions process network flows. Following the trend of virtualization of networks, virtualization of the network functions has also become a topic of interest. We define a model for formalizing the chaining of network functions using a context-free language. We process deployment requests and construct virtual network function graphs that can be mapped to the network. We describe the mapping as a Mixed Integer Quadratically Constrained Program (MIQCP) for finding the placement of the network functions and chaining them together considering the limited network resources and requirements of the functions. We have performed a Pareto set analysis to investigate the possible trade-offs between different optimization objectives.**

Keywords*—network function virtualization, virtual network functions, network function chaining, network service chaining, network function placement.

I. INTRODUCTION

An operator's network consists of a large number of intermediate Network Functions (NFs). Network Address Translators (NATs), load balancers, firewalls, and Intrusion Detection Systems (IDSs) are examples of such functions. Traditionally, these functions are implemented on physical middle-boxes, which are network appliances that perform functions other than standard path discovery or routing decisions for forwarding packets (RFC 3234 [1]). Middle-boxes are based on special-purpose hardware platforms that are expensive and difficult to maintain and upgrade. Following the trend of virtualization in large-scale networks, network function that were deployed as middle-boxes are also being replaced by *Virtual* Network Functions (VNFs).

Typically, network flows go through several network functions. That means a set of NFs is specified and the flows traverse these NFs in a specific order so that the required functions are applied to the flows. This notion is known as *network function chaining* or *network service chaining* [2], [3].

NFs can modify the traversing network flows in different ways. For example, a Deep Packet Inspector (DPI) can split the incoming flows over different branches according to the type of the inspected packets, each branch having a fraction of the data rate of the incoming flow. Firewalls can drop certain packets, resulting in flows with a lower data rate than incoming flows. A video optimizer can change the encoding of the video, which can result in a higher data rate. There can also be a dependency among a set of NFs that should be applied to the traffic in a network [4], which requires special attention to the *order* of traversing the functions in chaining scenarios. For instance, if the packets have to go through a WAN optimizer and an IDS, packet inspection by the IDS should typically be carried out before the WAN optimizer encrypts the contents. In case the functions in a chaining scenario do not have such a dependency, there can be multiple possibilities for chaining them together. Depending on how each function in the chain modifies the data rate of the flows, different chaining options can have different impact on the traffic in network links, on application performance, or on latency. We address two challenges in this area.

The *first challenge* is to formalize a request for chaining several NFs together, while considering the possible dependencies among them. Upon receiving such a request, the network operator has the freedom to chain the functions in the best possible way to fit the requirements of the tenant applications. After the logical chaining of functions is specified, the functions need to be placed in the operator's network. In addition to the dependencies within the chains of functions, NFs can also be shared and reused among different applications in the network. The *second challenge* is hence to find the best placement for the functions, considering the requirements of individual requests as well as the overall requirements of all network applications and the combination of requests. We will elaborate on these considerations in the following sections. For example, Figure 1 illustrates two different ways of chaining a set of functions together, one resulting in a lower average data rate requirement but higher processing requirements than the other.

Traditionally, modifying the way functions are chained together and changing the placement of functions in the network require complex modifications to the network, such as modifying the network topology or changing how physical boxes are connected to each other. Network Function Virtualization (NFV) offers more flexibility in network function chaining by simplifying chaining and placement of NFs and by making these changes more practical and achievable in real scenarios. Considering this, we focus on formalizing the chaining requests and placing the chained functions in the best locations according to optimization goals in an operator's network that supports NFV. Our solutions are not bound to a specific implementation option and can be applied to virtualized and non-virtualized NF chaining scenarios.

First, we give an overview of the related work in Section II. In Section III, we define our model for specifying the requests for chaining VNFs (given by network administrators or tenants) in a way that different chaining options can be analyzed and the one that fits the optimization goals and network requirements can be chosen. Based on the chaining possibilities given in the requests, the operator can decide the placement of the functions. The preprocessing step required before the functions can be placed in the network and a heuristic for reducing the runtime of the final decision process are described in Section IV. We define a Mixed Integer Quadratically Constrained Program (MIQCP) in Section V for placing the chained VNFs considering the following three objectives:

1) Maximizing remaining data rate
2) Minimizing number of used network nodes
3) Minimizing total latency over all paths

Finally, we describe the results of evaluating our placement and a multi-objective Pareto discussion in Section VI and conclude the paper in Section VII.

II. RELATED WORK

A number of standardization activities in the NFV area are carried out by ETSI and IETF, resulting in a white paper [5] and several drafts like problem statements in NFV [6], [3], use cases [7] and frameworks for network function chaining [8]. The model for chaining VNF presented in this paper is compatible with the models and architectures proposed in these drafts.

Our placement solution for chained NFs can be considered as an extension to the following two NP-hard problems: Location-Routing Problems (LRP) and Virtual Network Embedding (VNE). Location-routing problems [9], [10] aim at placement of components while reducing the costs in nodes, edges or paths. In these problems, each path has one start point and one end point. We need to create several paths between different pairs of NFs and connect these paths to represent the chaining. In this case, the routing problem turns into a multi-commodity flow problem, with inter-commodity dependencies.

Virtual network embedding (VNE) problems [11] are similar to our problem in the chaining aspect. Chained NFs can be seen as graphs to be embedded into a substrate network. These problems treat the nodes of the virtual graphs (NFs in our case) independently. In our problem, however, flows from different tenants can share and reuse NFs for better NF utilization. Similar to the approach of Fuerst et al. [12], we place several NFs on a single node to reduce inter-location traffic.

Joseph and Stoica [13] present a model for describing the behavior and functionality of a limited number of NFs but it is not a complete model for VNFs since it does not contain any information about requirements like computational resources and their influence on the network traffic. Gember et al. [14] introduce a network-aware orchestration layer for NFs. Our model for chaining VNFs is similar in some aspects, but their model is defined in a data center network, while we focus on an operator's network with multiple data center sites. Moreover, they do not capture any resource requirements of the functions apart from processed traffic.

NFV is still a concept under investigation and standardization of the problem definition and use cases are not yet finalized. Therefore, we found only a very limited amount of ongoing work related to our research focus. We define our models in a flexible way that can be extended and reinterpreted easily when more technical details about the chaining scenarios and implementation requirements are available.

III. NETWORK FUNCTION CHAINING SPECIFICATION

We model the substrate network, where VNF chains are defined and placed, as a connected directed graph, $G{=}(V,E)$. Some of network nodes are *switch nodes*, with typical routing and switching capabilities, along with (small) computational capacity that can be used for running VNFs, e.g., inside an FPGA in the switch. The remaining nodes are distributed sites with (much larger) computational capacity. We consider each of these sites as a large computational unit, called a *data center node*, without looking into their internal topology. We define two types of computational capacities for the nodes: $c_d(v)$ and $c_s(v)$ $(\forall v{\in}V)$. For today's switch nodes, c_d is zero and for current data center nodes, c_s is zero. We define both types of capacities for all nodes to keep our model open to future extensions. For example, in future, switches might be equipped with general-purpose processing capabilities, leading to $c_d > 0$ for switch nodes. The network links are directed edges in the graph, with data rate $d(v,v')$ and latency $l(v,v')$ for every edge $(v,v'){\in}E$.

The following information about offered network functions is available and maintained by the network operator:

- Set F of available network functions.
- Computational resource requirements $p(f)$ $(\forall f \in F)$ of an instance of the VNF f per each request, whether it is placed on a switch node, $p_s(f)$, or on a data center node, $p_d(f)$. Some functions can be placed either on a switch or on a data center node, e.g., a load balancer ($p_d{>}0$ and $p_s{>}0$), and some can be placed only on a data center node, e.g., a Virtual Machine (VM) implementing a video optimizer ($p_d{>}0$ and $p_s{=}0$).
- Maximum number of instances of the VNF that can be deployed, $n_{\text{inst}}(f)$, e.g., determined by the number of licenses that the operator owns for the VNF.
- Number of chaining requests an instance of a VNF can handle, $n_{\text{req}}(f)$. For example, an anti-virus function can be configured once and used for every chain that needs this function, (i.e., number of requests it can handle is only limited by hardware specifications), but a firewall might need specific configurations for each chaining request and one instance of this function cannot be shared between two chains (i.e., $n_{\text{req}}{=}1$).

A network operator receives *deployment requests* for different partially ordered sets of VNFs. In these requests, a network administrator or the tenant specifies which of the offered functions should be applied in which order to given flows between fixed start and end points. A deployment request contains the following information:

- Set U of individual requests for instances of available network functions.
- *Chaining request*, denoted as c, for specifying the desired order of functions.
- For each branch leaving a requested function, the percentage of incoming data rate it produces given as an ordered set $r(u)$ $(\forall u{\in}U)$ for each function. For example, for a

DPI that is expected to send 20% of the incoming packets towards a video optimizer and 80% towards a firewall this set is given as $\{0.2, 0.8\}$.

- Set A of fixed start and end points for the flows, e.g., an application VM deployed in a data center node or a router that connects the operator's network to external networks.
- Set $A_{\text{pairs}} \subseteq A \times A$ of *pairs* of start and end points belonging to different flows.
- Location of start and end points of flows in the network, $\text{loc}(a) \in V$ $(\forall a \in A)$.
- Initial data rate entering the chained functions, d_{in}.
- Maximum tolerable latency between the start and end points of flows, $l_{\text{req}}(a, a')$ $(\forall (a, a') \in A_{\text{pairs}})$.

We define a context-free language for formalizing the chaining requests. Using this language, complex requests can be composed that contain the order of functions to be applied to network flows. Every chaining request is formalized using different types of *modules*. The elements of this language are the following modules:

- An individual function or a start/end point for the chain
- A set of functions that should be applied to network flows in an optional order (*optional order* module)
- A function that splits the incoming flows into multiple branches that consist of different modules (*split* module)
- A function that splits the incoming flows into multiple branches that all consist of the same module (*parallel* module)

Several (possibly nested) modules can be placed sequentially in the chaining request to reflect a simple and fixed *order* among desired functions.

$\mathcal{G} = (\mathcal{V}, \mathcal{T}, \mathcal{P}, \mathcal{S})$ is the grammar for this language. $\mathcal{V}$ is the set of non-terminals consisting of the following variables: $\langle\text{modules}\rangle$, $\langle\text{mod}\rangle$, $\langle\text{order}\rangle$, $\langle\text{optorder}\rangle$, $\langle\text{split}\rangle$, $\langle\text{parallel}\rangle$, $\langle\text{term}\rangle$, $\langle\text{moremod}\rangle$, $\langle\text{moreterm}\rangle$, $\langle\text{num}\rangle$.

$\mathcal{T} = U \cup A \cup D \cup \{1, 2, \ldots, n\} \cup \{\epsilon\}$ is the set of symbols of this language where ϵ is the empty string. A subset of natural numbers from 1 to n is required for displaying the number of branches that leave a VNF. n can be defined as a number larger than the maximum number of outgoing branches in all requests. This upper bound is necessary because the set of symbols has to be a finite set. D is the set of delimiters consisting of the following symbols: . , ; { } ( ) [ ]. $\mathcal{S} = \langle\text{start}\rangle$ is the start symbol of the grammar and $\mathcal{P}$ is the set of production rules:

$$\langle\text{start}\rangle ::= \langle\text{modules}\rangle \tag{1}$$

$$\langle\text{modules}\rangle ::= \langle\text{order}\rangle\ \langle\text{modules}\rangle \mid \langle\text{mod}\rangle \tag{2}$$

$$\langle\text{order}\rangle ::= \langle\text{mod}\rangle\ \cdot \tag{3}$$

$$\langle\text{mod}\rangle ::= \langle\text{optorder}\rangle \mid \langle\text{split}\rangle \mid \langle\text{parallel}\rangle \mid \langle\text{term}\rangle \tag{4}$$

$$\langle\text{optorder}\rangle ::= (\ \langle\text{term}\rangle\ \langle\text{moreterm}\rangle\ ) \tag{5}$$

$$\langle\text{split}\rangle ::= \langle\text{term}\rangle\ [\ \langle\text{modules}\rangle\ \langle\text{moremod}\rangle\ ] \tag{6}$$

$$\langle\text{parallel}\rangle ::= \langle\text{term}\rangle\ \{\ \langle\text{term}\rangle\ \langle\text{moreterm}\rangle\ ;\ \langle\text{modules}\rangle\ ;\ \langle\text{num}\rangle\ \} \tag{7}$$

$$\langle\text{moreterm}\rangle ::= \ ,\ \langle\text{term}\rangle\ \langle\text{moreterm}\rangle \mid \epsilon \tag{8}$$

$$\langle\text{moremod}\rangle ::= \ ,\ \langle\text{modules}\rangle\ \langle\text{moremod}\rangle \mid \epsilon \tag{9}$$

$$\langle\text{term}\rangle ::= u_1 \mid u_2 \mid \ldots \mid u_{|U|} \mid a_1 \mid a_2 \mid \ldots \mid a_{|A|} \tag{10}$$

$$\langle\text{num}\rangle ::= 1 \mid 2 \mid 3 \mid \ldots \mid n \tag{11}$$

Rule 4 expresses the 4 different types of modules. We refer to the requests for using an instance of a VNF and the start/end points of chains as *terms* in this grammar. Rule 3 is used for defining a fixed and simple order among modules. Optional order modules are produced by Rule 5. Such a module consists of a set of functions that should be applied to the flows and the order of traversing these functions can be chosen by the operator. A request for a split module can be expressed by Rule 6, where the splitting function is a *term* and the modules on different branches can be any of the defined types of modules. Finally, a request for a parallel module can be produced using Rule 7. Parallel modules have 4 parts: 1) The function that splits the flows into different branches; 2) a set of functions, including the splitting function, that can be placed in an arbitrary order before the flows reach the modules on different branches (optional); 3) the module that should be replicated for the given number of times on multiple branches; 4) number of outgoing branches from the splitting function (number of times the module mentioned in part 3 should be replicated).

A VNF chaining request formalized this way is a representation for a connected directed graph that we refer to as a *VNF graph*. The required functions and also the start and end points of the flows are nodes of the VNF graph ($U \cup A$). The start/end points are mapped to fixed locations in the substrate network and the location for the VNFs has be to determined. Each one of the directed links in the set of edges in this graph (U_{pairs}) represents the order of traversing the functions. Every link in the VNF graph has to be mapped to a path (consisting of at least one edge) in the substrate network graph. We define a two-step process for deploying the chained functions based on the deployment requests: processing the requests and building VNF graphs, and finding the optimal placement for the VNF graphs based on optimization goals in the operator's network. We describe this process in the following sections.

IV. PROCESSING DEPLOYMENT REQUESTS

The network operator receives deployment requests for placing chained VNFs in the network, where chaining requests are formalized using the language described in Section III. The operator needs to find the best placement for several chained functions based on multiple deployment requests. The first step is to build VNF graphs for every deployment request.

For each deployment request, the chaining request (c) is first parsed using the grammar of the language (Section III). The parser matches and stores different modules of the request. Modules consisting of a single function or start/end point of flows are stored as a node of the VNF graph. For the modules where a number of functions can be ordered arbitrarily (*optional order* and *split* modules) every possible permutation of the set of functions is computed and stored separately as a candidate for being a part of the final VNF graph. Moreover, for every match of a *parallel* module, the module on the branches is replicated for the requested number of times and stored as a part of the graph. Using the specified orders and depending on the modules that build the chaining request, at the end of the parsing process different modules are stored as parts of the VNF graph with explicit orders among all functions. Parts of the graphs are then connected using directed links that represent the sequence of modules in the request. Considering the different permutations for different modules, at least one VNF graph is built out of each chaining request. Using the rest of the information in the deployment request and the information available about the network functions, computational resource requirements are assigned to the nodes of the VNF graphs. The links of the graphs are also annotated by data rate and latency requirements.

Each of the graphs that can be created from a request can have different characteristics in terms of average data rate required for its links and number of VNF instances. For example, the Load Balancer (LB) in Figure 1 splits the incoming flows into three different branches to balance traffic over three instances of f_3. The ratio of outgoing to incoming data rate in all VNFs except the LB is 1. Placing this load balancer earlier in the directed graph, as in Figure 1b, reduces the data rate of the links on each branch after it. But it also means that up to three instances of all subsequent VNFs will be required on the paths towards f_3. Each of these instances has a lower processing requirement than the instances in Figure 1a that should handle higher data rates.

For every module that requests $n \in \mathbb{N}_{\geq 0}$ VNFs with optional order among them (i.e., *optional order* or *parallel* module), $n!$ permutations are computed and stored. For a chain that contains multiple optional order or parallel modules, the number of different ways for combining these modules is the product of the number of permutations for different modules. For example, if a chaining request contains one optional order module with 3 different VNFs and one parallel module in which 4 VNFs can be placed with an arbitrary order, a total of $3! \cdot 4! = 144$ combinations of these modules is possible. That means, for finding the best combination, the placement step in the deployment process has to be done 144 times so that the results can be compared and the option that fits the network requirements can be chosen. Our deployment process creates the VNF graphs for each deployment request separately, and then all possible combinations of VNF graphs from different requests would have to be computed and sent to the placement step. As the number of combinations increases very quickly in the number of deployment requests and the number of functions in each request, trying every possible combination becomes impractical.

We propose a heuristic for choosing one of the possible combinations, respecting the optimization goals in an operator's network. In this method, instead of computing all possible permutations of the sets of functions with arbitrary orders, we sort the functions in ascending order according to their ratio of outgoing to incoming data rate. The function that reduces the data rate of the flows the most is placed before all other functions in the module. Therefore, each deployment request results in one single VNF graph and the final input to the placement step is a disconnected graph consisting of the VNF graphs from different requests. We define the placement optimization problem in Section V for finding the best placement in the substrate network for the combined VNF graphs. We optimize the placement for a new set of deployment requests by mapping all new requests together into a network that may or may not contain a previous deployment of chained functions. This can be extended to adapt the existing deployments to the new state of the network.

For each deployment request, our heuristic chooses a VNF graph that has the minimum overall data rate requirement among all possible VNF graphs for that request. This method might discard some graphs that are optimal in terms of total number of required VNF instances or the total latency. However, the gain in execution time can compensate for this deviation from optimality. We have performed a Pareto analysis of the placement optimization problem to show the trade-offs between different optimization objectives, which we present in Section VI.

TABLE I. REQUIRED INPUT FOR PLACEMENT

Domain	Parameter	Description
$\forall v \in V$	$c_d(v)$	Data center computational resources in v
	$c_s(v)$	Switch computational resources in v
$\forall (v, v') \in E$	$d(v, v')$	Data rate capacity on (v, v')
	$l(v, v')$	Latency of (v, v')
$\forall f \in F$	$n_{\text{inst}}(f)$	Number of allowed instances for f
	$n_{\text{req}}(f)$	Number of requests f can handle
	$p_d(f)$	Data center resource demand of f
	$p_s(f)$	Switch resource demand of f
$\forall u \in U$	$t(u)$	Requested function
$\forall (u, u') \in U_{\text{pairs}}$	$d_{\text{req}}(u, u')$	Data rate demand of (u, u')
$\forall a \in A$	$\text{loc}(a)$	Network node where a is placed
$\forall (a, a') \in A_{\text{pairs}}$	$\text{paths}(a, a')$	All possible paths between a and a'
	$l_{\text{req}}(a, a')$	Maximum latency between a and a'

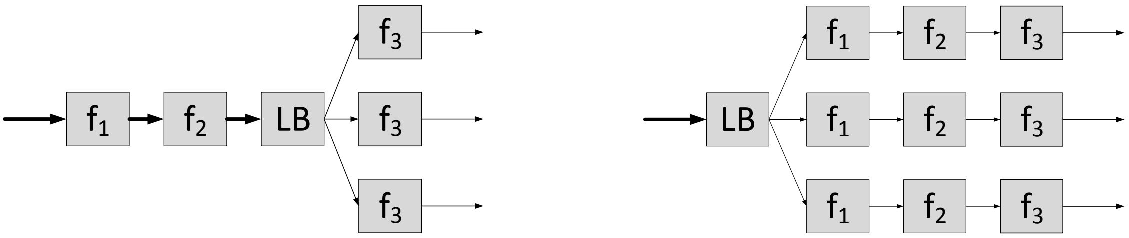
TABLE II. DECISION VARIABLES

Domain	Variable	Description
$\forall u \in U$, $\forall v \in V$	$m_{u,v}$	u mapped to v
	$\text{ms}_{u,v}$	u mapped to a switch function on v
	$\text{md}_{u,v}$	u mapped to a data center function on v
$\forall f \in F$, $\forall v \in V$	$i_{f,v}$	An instance of f mapped to v
$\forall (v, v') \in E$, $\forall x, y \in V$, $\forall (u, u') \in U_{\text{pairs}}$	$e_{v,v',x,y,u,u'}$	(v, v') belongs to path between x and y, where u and u' are mapped to
$\forall v \in V$	used_v	At least one request mapped to v
$\forall (v, v') \in E$	$\text{remdr}_{v,v'}$	Remaining data rate on (v, v')
$\forall (u, u') \in U_{\text{pairs}}$	$\text{lat}_{u,u'}$	Latency of the path between u and u'

V. PLACEMENT OF CHAINED NETWORK FUNCTIONS

There can be several metrics that the operator might need to optimize. We formulate the placement optimization problem as an MIQCP with respect to data rate, number of used network nodes, and latency. Input to the placement step is the capacity of network nodes and links, requirements of different network functions, and the combined VNF graph from the request processing step. Table I shows an overview of the input parameters to the placement optimization problem.

Decision variables are described in Table II. "remdr" and "lat" are continuous variables and all other ones are binary indicator variables. We show the constraints of the optimization problem in Section V-A and the objective functions in Section V-B.



(a) (b)

Fig. 1. Two of the chaining options for a set of functions

A. Constraints

In this section, we describe the constraints of the placement optimization problem in 3 parts: 1) Placing functions in network nodes and mapping requests for using instances of network functions to these nodes; 2) creating paths between functions; 3) collecting metric values. Placement and path creation constraints have a clear separation that facilitates extending the model in either part without causing problems in the other part. Besides, necessary ties between these parts are also carefully defined, for example, using the decision variable e, to build a consistent and uniform model for placing functions and chaining them together optimally.

1) Network Function Placement Constraints:

$$\forall u{\in}U : \sum_{v\in V} m_{u,v}{=}1 \tag{12}$$

$$\forall a{\in}A : m_{a,\text{loc}(a)}{=}1 \tag{13}$$

$$\forall f{\in}F, \forall v{\in}V :$$

$$\sum_{u\in Ut(u)=f} m_{u,v}{\leq}\mathcal{M}{\cdot}i_{f,v} \tag{14}$$

$$i_{f,v}{\leq}\sum_{u\in U,t(u)=f} m_{u,v} \tag{15}$$

$$\forall u{\in}U, \forall v{\in}V : \text{ms}_{u,v}{+}\text{md}_{u,v}{=}1 \tag{16}$$

$$\forall u{\in}U, \forall v{\in}V, p_s(t(u)){=}0, p_d(t(u)){\neq}0 :$$

$$\text{ms}_{u,v}{=}0 \tag{17}$$

$$\text{md}_{u,v}{=}1 \tag{18}$$

$$\forall v{\in}V :$$

$$\sum_{u\in U} m_{u,v}{\cdot}\text{md}_{u,v}{\cdot}p_d(t(u)){\leq}c_d(v) \tag{19}$$

$$\sum_{u\in U} m_{u,v}{\cdot}\text{ms}_{u,v}{\cdot}p_s(t(u)){\leq}c_s(v) \tag{20}$$

$$\forall f{\in}F : \sum_{v\in V} i_{f,v}{\leq}n_{\text{inst}}(f) \tag{21}$$

$$\forall v{\in}V, \forall f{\in}F : \sum_{u\in U,t(u)=f} m_{u,v}{\leq}n_{\text{req}}(f) \tag{22}$$

Every request for using a VNF should be mapped to exactly one node (Constr. 12). Start/end points of the flows are fixed in the network, so $\forall a{\in}A$, $m_{a,\text{loc}(a)}$ is not a decision variable. However, values of m are also used in path creation constraints and there are paths to be created between the start/end points and the rest of the chained functions. Therefore, for these points m is defined similar to the functions (Constr. 13). Constraints 14 and 15 are complementary to each other and avoid starting an instance of a function on network nodes without mapping any requests to them. They also make sure that a request is mapped to a node only if an instance of the required function is placed on that node. $\mathcal{M}{\in}\mathbb{N}$ is a number larger than the sum on the left side of the inequality in Constr. 14 (a so-called "big M" constraint). Constr. 16 ensures that a request is mapped to a node either as a switch function or a data center function but not both. For every u that is a request for a VNF that can only be placed on a data center, $\text{ms}_{u,v}$ is set to zero and $\text{md}_{u,v}$ is set to one. This constraint is necessary for correctness of Constr. 19 and 20 when the function has a non-zero value for both types of computational resource requirements (i.e., can be placed either on a switch node or on a data center node). Computational resource requirements of all requests mapped to a node should be less than or equal to available resources in that node. In data center nodes only the data center resource requirements of the functions mapped to them are considered and in switch nodes only the switch resource requirements (Constr. 19 and 20). Variables m, md, and ms are binary decision variables and the product of two binary variables can easily be linearized. Therefore, Constr. 19–20 can be considered as quadratic constraints instead of cubic. For every function f, up to $n_{\text{inst}}(f)$ instances can be placed in the network (Constr. 21). Finally, Constr. 22 ensures that every instance of f handles no more than $n_{\text{req}}(f)$ requests.

2) Path Creation Constraints:

$$\forall (v,v'){\in}E, \forall x,y{\in}V, \forall (u,u'){\in}U_{\text{pairs}} :$$

$$e_{v,v',x,y,u,u'}{\leq}m_{u,x}{\cdot}m_{u',y} \tag{23}$$

$$\forall (u,u'){\in}U_{\text{pairs}} :$$

$$\sum_{(x,v)\in E, y\in V} e_{x,v,x,y,u,u'}{\cdot}m_{u,x}{\cdot}m_{u',y}{=}1 \tag{24}$$

$$\sum_{(x,v)\in E, y\in V} e_{x,v,x,y,u,u'}{\cdot}(1{-}m_{u,x}{\cdot}m_{u',y}){=}0 \tag{25}$$

$$\sum_{(v,y)\in E, x\in V} e_{v,y,x,y,u,u'}{\cdot}m_{u,x}{\cdot}m_{u',y}{=}1 \tag{26}$$

$$\sum_{(v,y)\in E, x\in V} e_{v,y,x,y,u,u'}{\cdot}(1{-}m_{u,x}{\cdot}m_{u',y}){=}0 \tag{27}$$

$$\forall (u,u'){\in}U_{\text{pairs}}, \forall w,x,y{\in}V :$$

$$\sum_{\substack{v\in V, v\neq y,\\ (v,w)\in E}} e_{v,w,x,y,u,u'}{=}\sum_{\substack{v'\in V, w\neq x,\\ (w,v')\in E}} e_{w,v',x,y,u,u'} \tag{28}$$

$$\forall (u,u')\in U_{\text{pairs}}, \forall v,x,y\in V, x\neq y:\ e_{v,v,x,y,u,u'}=0 \tag{29}$$

$$\forall (u,u')\in U_{\text{pairs}}, \forall x,y\in V, \forall (v,v'),(v',v)\in E, v\neq v':$$
$$e_{v,v',x,y,u,u'}+e_{v',v,x,y,u,u'}\leq 1 \tag{30}$$

$$\forall (v,v')\in E:$$
$$\sum_{(u,u')\in U_{\text{pairs}},\forall x,y\in V} e_{v,v',x,y,u,u'}\cdot d_{\text{req}}(u,u')\leq d(v,v') \tag{31}$$

$$\forall (a,a')\in A_{\text{pairs}}:$$
$$\sum_{\substack{(v,v')\in E,x,y\in V,\\ (u,u')\in \text{paths}(a,a')}} e_{v,v',x,y,u,u'}\cdot l(v,v')\leq l_{\text{req}}(a,a') \tag{32}$$

An edge in the network graph belongs to a path between nodes v and v' if there are requests mapped to these nodes and a path needs to be created between them (Constr. 23). Constr. 24 ensures the path in network graph created for edge (u,u') in the VNF graph starts at exactly one edge going out of node x in the network where request u is mapped to. Without Constr. 25 any random edge might be marked as the first edge of this path. Analogously, Constr. 26–27 ensure the correctness and uniqueness of the creation of the last edge in the path. In Constr. 24–27, the product of the binary variables can be linearized to avoid having cubic constraints in the problem. Constr. 28 ensures that for every node w in the network graph, if one of its incoming edges belongs to a path between the nodes where requests u and u' are mapped to, then one of its outgoing edges also belongs to this path. Excluded from this rule are the cases where the incoming edge to a node is the last edge in the path and where the outgoing edge from a node is the first edge in the path. Constraints 29–30 prevent the creation of infinite loops and unnecessary extensions of the created paths. For every edge in the network, the sum of the required data rates of all paths going through that edge should be less than or equal to the data rate capacity of this edge (Constr. 31). Moreover, the sum of latencies of all edges that belong to a path between the start and end points of a flow should not exceed the maximum tolerable latency given for that flow (Constr. 32).

3) *Metrics Calculation Constraints:*

$$\forall v\in V: \sum_{f\in F} i_{f,v}\leq \mathcal{M}'\cdot \text{used}_v \tag{33}$$

$$\forall v\in V: \text{used}_v\leq \sum_{f\in F} i_{f,v} \tag{34}$$

$$\forall (v,v')\in E:$$
$$\text{remdr}_{v,v'}=d(v,v')-\sum_{\substack{(u,u')\in U_{\text{pairs}},\\ \forall x,y\in V}} e_{v,v',x,y,u,u'}\cdot d_{\text{req}}(u,u') \tag{35}$$

$$\forall (u,u')\in U_{\text{pairs}}: \text{lat}_{u,u'}=\sum_{x,y\in V,(v,v')\in E} e_{v,v',x,y,u,u'}\cdot l(v,v') \tag{36}$$

Using Constr. 33 and 34 we mark a network node as *used* if there is an instance of at least one function mapped to it. $\mathcal{M}'\in\mathbb{N}$ is a number larger than the sum on the left side of the inequality in Constr. 33. For each edge (v,v') in the network, the remaining data rate after the placement of chained functions is calculated by subtracting the sum of required data rates of all paths that go through this edge from the initial data rate of it (Constr. 35). For every edge (u,u') in the VNF graph, the latency of the paths created between the nodes where requests u and u' are mapped to is equal to the sum of latencies of all network edges that belong to this path (Constr. 36).

B. Objectives

Different objectives can be targeted for placement optimization, and each of them can result in a different mapping of the VNF graphs into the network graph. We define three objective functions and describe the behavior of the placement process using each objective.

1) *Maximizing the remaining data rate on network links:*

$$\text{maximize} \sum_{(v,v')\in E,v\neq v'} \text{remdr}_{v,v'} \tag{37}$$

As highly utilized links can result in congestion in the network, solutions that leave more capacity on the links are desirable. This objective aims at leaving more data rate on the links. By maximizing the sum of remaining data rate over all edges except self-loops, it forces the placement algorithm to use self-loops (i.e., links between two functions that are placed on one network node) more than other links.

2) *Minimizing the number of used nodes in the network:*

$$\text{minimize} \sum_{v\in V} \text{used}_v \tag{38}$$

This objective can result in an energy-efficient solution by allowing more unused nodes to be switched off. However, it might concentrate the placement of functions on a small subset of nodes causing congestion in the network.

3) *Minimizing the latency of the created paths:*

$$\text{minimize} \sum_{(a,a')\in l_{\text{req}}} \Big(\sum_{P\in \text{paths}(a,a')} \Big(\sum_{(u,u')\in P} \text{lat}_{u,u'} \Big) \Big) \tag{39}$$

In complex chaining scenarios with branches in the structure, there are multiple simple paths between the start and end points. As each path consists of different sets of edges, they can have different latencies. This objective function minimizes the mean latency of all paths created for all deployment requests.

These objective functions cause the placement to focus on a specific goal. For example, using the third objective, we get solutions with minimum latency but the remaining data rate of the links or the number of used nodes in the network are not predictable. There can be conflicts in the solutions that are considered as *optimal* using each of these objectives. Results of our Pareto analysis (Section VI) show that the three metrics can have trade-offs but are not necessarily conflicting.

VI. EVALUATION

We have performed two types of evaluation of our model and placement optimization process: 1) Observing the behavior of the placement process and our heuristic for reducing the runtime of the process when there are several ordering possibilities in deployment requests; 2) Pareto analysis for showing the possible trade-offs between our metrics of interest. Currently, there is no commonly accepted evaluation model for either the actual chained network functions or the user requests for tenant applications with chains of virtual or physical network functions in their structure. Therefore, we have designed small evaluation scenarios with manually created deployment requests on a substrate network with 12 nodes and 42 directed edges (including self-loops), based on the *abilene* network from SNDlib [15]. We have built these requests to test the capabilities of the request processing and placement steps while being compatible with known use cases of chaining VNFs [7], [16]. We have used the Gurobi Optimizer to solve the MIQCP on machines with Intel Xeon X5650 CPUs running at 2.67 GHz.

A. Evaluation of Optional Orders in Chaining Requests

For this part, we have used a set of chaining requests which allow an arbitrary order among a set of functions. Our placement results show that for small requests that have small requirements compared to the available resources in the network sorting the functions according to their ratios of outgoing to incoming data rate gives the best (or one of the best) solutions for the requests. When placing several requests the combination of requests have higher requirements and there can be dependencies between different requests (e.g., shared VNFs). In that case, combining the *sorted* chains may result in a sub-optimal solution. The runtime, however, should also be considered. For example, in our evaluation settings, placing a combination of 3 requests with optional orders (resulting in 6, 6, and 4 VNF graphs, respectively) by sorting them takes an average of 13 minutes using different objective functions to give a close to optimal solution. But computing all combinations ($6 \cdot 6 \cdot 4 = 144$) to find the optimal one needs 31 hours. For this example, placement of the sorted chains was one of the optimal placements regarding remaining data rate and the latency but it used more network nodes compared to some other solutions. Our heuristic offers the opportunity to place the chained functions in an acceptable time by allowing a slight deviation from the optimal solution. Considering the fact that even a simple set with 3 requests can result in 144 combinations, we use this heuristic for the second part of our evaluations and choose the sorted chain for each of the requests in our request sets.

B. Pareto Set Analysis

For the Pareto analysis, we have performed the placement for different sets of deployment requests using the objectives defined in Section V-B. Our request sets resemble three different VNF chaining scenarios (broadband, mobile core, and data center networks) from the IETF draft on service function chaining use cases [7]. We have defined two different sets of requests with different complexities for each scenario. First we did a range estimation run for the request sets by performing the placement using each of the objectives and recording the highest and lowest values for the metrics. After identifying the interesting ranges, we performed the Pareto optimization. Some of our results show trade-offs between optimizing the remaining data rate, number of used nodes, and latency; there are also results that show it is possible to find a placement that optimizes all three metrics. Figure 2 shows the results for two of our request sets. For better visibility, we use different colors for different number of used nodes in this figure. As illustrated in Figure 2a, the objectives are not always conflicting. If there are enough free resources in the network, it might be possible to find a solution that is optimal in terms of all three metrics (the solution marked by a star).

Results of the Pareto optimization for another one of our request sets is shown in Figure 2b. Allowing the placement to use more nodes gives results with lower latency and higher remaining data rate. But after a certain point increasing the number of used nodes does not improve the results anymore. The figure also shows a trade-off between latency and remaining data rate. Using a fixed number of nodes, the latency of the solution increases to get a higher value for the remaining data rate, which also means that improving the latency can result in a lower remaining data rate.

Network operators can have different objectives which require different placement solutions. Results of our Pareto analysis can help the operators prioritize their optimization goals and choose the right objective functions.

VII. CONCLUSION

In this paper, we have presented a formal model for specifying VNF chaining requests, including a context-free language for denoting complex composition of VNF. Using this model, we have formulated an optimization problem for placing the chained VNF in an operator's network with multiple sites, based on requirements of the tenants and the operator. Our evaluations have shown that placement of chained VNF is not a trivial problem and that placement decisions have to be taken differently according to the desired placement objective (i.e., remaining data rate, latency, number of used network nodes). Our results warrant further investigation to allow fast and efficient placement of VNFs and to facilitate the deployment of NFV in different types of networks.

ACKNOWLEDGMENT

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] B. Carpenter and S. Brim, "Middleboxes: Taxonomy and Issues. RFC 3234," 2002.

[2] F. Schneider, T. Egawa, S. Schaller, S.-i. Hayano, M. Schöller, and F. Zdarsky, "Standardizations of SDN and Its Practical Implementation," *NEC Technical Journal, Special Issue on SDN and Its Impact on Advanced ICT Systems*, vol. 8, no. 2, Apr. 2014.

[3] P. Quinn and T. Nadeau, "Service Function Chaining Problem Statement," Active Internet-Draft, IETF Secretariat, Internet-Draft draft-ietf-sfc-problem-statement-05, Apr. 2014.

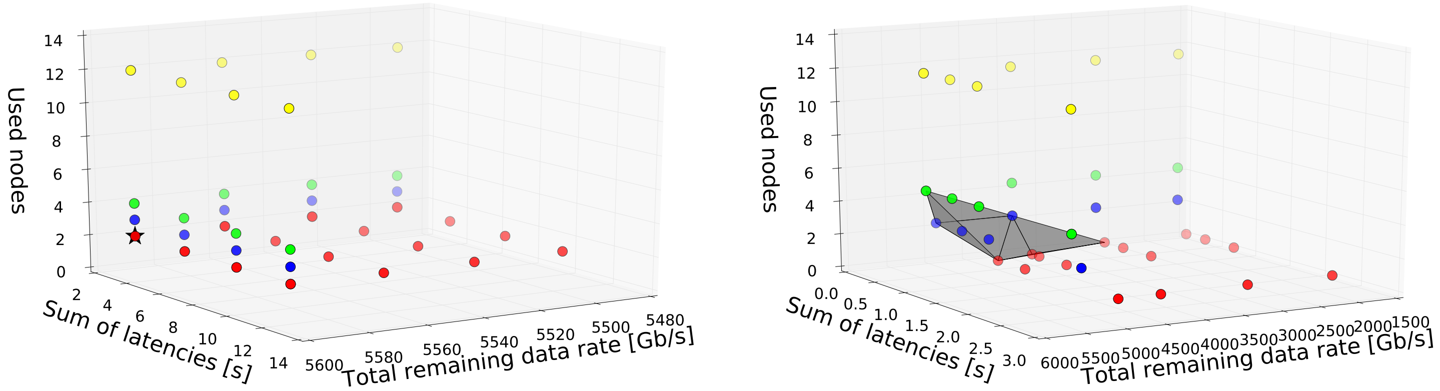
[4] V. Sekar, S. Ratnasamy, M. K. Reiter, N. Egi, and G. Shi, "The middlebox manifesto: enabling innovation in middlebox deployment," *Proceedings of the 10th ACM Workshop on Hot Topics in Networks*, p. 21, 2011.

[5] R. Guerzoni, "Network Functions Virtualisation: An Introduction, Benefits, Enablers, Challenges and Call for Action, Introductory white paper," *SDN and OpenFlow World Congress*, 2012.

[6] W. Xu, Y. Jiang, and C. Zhou, "Problem Statement of Network Functions Virtualization Model," Internet-Draft, IETF Secretariat, Internet-Draft draft-xjz-nfv-model-problem-statement-00, Sep. 2013.

[7] W. Liu, H. Li, O. Huang, M. Boucadair, N. Leymann, Z. Cao, Q. Sun, and C. Pham, "Service Function Chaining (SFC) Use Cases," Active Internet-Draft, IETF Secretariat, Internet-Draft draft-liu-sfc-use-cases-05, Apr. 2014.

[8] M. Boucadair, C. Jacquenet, R. Parker, D. Lopez, J. Guichard, and C. Pignataro, "Service Function Chaining: Framework and Architecture," Active Internet-Draft, IETF Secretariat, Internet-Draft draft-boucadair-sfc-framework-02, Feb. 2014.



(a) Results for a sample request set with optimal solution

(b) Results for a sample request set including a Pareto set

Fig. 2. Sample results from the multi-objective Pareto analysis

[9] G. Nagy and S. Salhi, “Location-routing: Issues, models and methods,” *European Journal of Operational Research*, vol. 177, no. 2, pp. 649–672, Mar. 2007.

[10] C. Prodhon and C. Prins, “A Survey of Recent Research on Location-Routing Problems,” *European Journal of Operational Research*, Jan. 2014.

[11] A. Fischer, J. F. Botero, M. T. Beck, H. de Meer, and X. Hesselbach, “Virtual Network Embedding: A Survey,” *IEEE Communications Surveys & Tutorials*, vol. 15, no. 4, pp. 1888–1906, 2013.

[12] C. Fuerst, S. Schmid, and A. Feldmann, “Benefits and Limitations of Pre-Clustering,” in *IEEE 2nd International Conference on Cloud Networking (CloudNet 2013)*, San Francisco, CA, 2013, pp. 91–98.

[13] D. Joseph and I. Stoica, “Modeling middleboxes,” *Network, IEEE*, vol. 22, no. 5, pp. 20–25, 2008.

[14] A. Gember, A. Krishnamurthy, S. S. John, R. Grandl, X. Gao, A. Anand, T. Benson, A. Akella, and V. Sekar, “Stratos: A network-aware orchestration layer for middleboxes in the cloud,” *arXiv preprint arXiv:1305.0209*, 2013.

[15] S. Orlowski, M. Pióro, A. Tomaszewski, and R. Wessäly, “SNDlib 1.0–Survivable Network Design Library,” in *Proceedings of the 3rd International Network Optimization Conference (INOC 2007), Spa, Belgium*, April 2007.

[16] N. Leymann, “Flexible Service Chaining. Requirements and Architectures.” http://www.ewsdn.eu/previous/presentations/Presentations_2013/EWSDN-2013-v10a.pdf, 2013.