I. Schaefer, D. Karagiannis, A. Vogelsang, D. Méndez, C. Seidl (Hrsg.): Modellierung 2018,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2018 39

Model-driven Development of Virtual Network Embedding Algorithms with Model Transformation and Linear Optimization Techniques

Stefan Tomaszek[1], Erhan Leblebici[1], Lin Wang [2], Andy Schürr[1]

Abstract: Enhancing the scalability and utilization of data centers, virtualization is a promising technology to manage, develop and operate network functions in a flexible way. For the placement of virtual networks in the data center, many approaches and algorithms are discussed in the literature to optimize solving the so-called virtual network embedding problem with respect to various optimization goals. This paper presents a new approach for the model-driven specification, simulation-based evaluation, and implementation of possible mapping algorithms that respect a set of given constraints and using linear optimization solving techniques to select one almost optimal mapping. Rule-based model transformation techniques are used to translate a given mapping problem into a linear optimization problem by taking domain specific knowledge into account. The resulting framework thus supports the design and evaluation of (correct-by-construction) virtual network embedding algorithms on a high level of abstraction. Well-defined model transformation rule refinement strategies can be used to reduce the search space for the employed linear optimization techniques.

Keywords: Model-driven development; virtual network embedding; triple graph grammar; integer linear programming; data center

1 Introduction and Motivation

With the rapid evolvement of the Internet, online services such as social networking, e-commerce, and online gaming have become ubiquitous. These online services are constantly generating a huge amount of data that is managed and analyzed by service providers like Google, Facebook or Amazon. To this end, cloud computing has become the norm as it can provide the required availability, scalability, and cost-effectiveness and can support rapid development and operation cycles. Data centers (DCs) are major facilities for cloud computing and usually host a large number of computing or storage servers interconnected by a dedicated communication network. To operate these very large and complex environments,

[1] Technische Universität Darmstadt, Real-Time Systems Lab, Merckstr. 25, 64283 Darmstadt, Germany, {stefan.tomaszek,erhan.leblebici,andy.schuerr}@es.tu-darmstadt.de
[2] Technische Universität Darmstadt, Telecooperation Lab, Hochschulstraße 10, 64289 Darmstadt, Germany, wang@tk.tu-darmstadt.de

Acknowledgment
This work has been funded by the German Research Foundation (DFG) as part of project A1 within the Collaborative Research Center (CRC) 1053 – MAKI.

virtualization has become a key technology, decoupling the underlying infrastructure and the upper-layer application and increasing the management flexibility so that economy-of-scale can be easily achieved. As services are encapsulated in virtual machines (VMs) and are interconnected with virtual networks (VNs), cloud operators can consolidate multiple VMs on the same physical machine in the substrate network (SN), migrate the VM at runtime, and span VNs regardless of the underlying network cabling details. This flexibility makes it possible to enact a fast development process, to unify the configuration, and to reduce the energy consumption of the DC, which is a significant cost factor for the cloud operators.

However, the virtualization and thus the unification of the configuration is accompanied by a high complexity, which manifests itself especially in the virtual network embedding (VNE) problem. The VNE problem is defined as the embedding of VNs in the SN with various constraints respected and with multiple metrics optimized on both the computing nodes and the network. When considering modern frameworks like OpenStack [SAE12] for the VNE problem, administrators often perform these embeddings manually.

In recent years, research into automating VNE has been greatly intensified. A variety of algorithms and methodologies has been developed to improve the distribution of virtual servers and networks within DCs. These algorithms and methodologies depend on specific optimization objectives such as higher resource utilization or lower energy consumption and specific structures of the underlying infrastructure [Gu10]. Performing a customized embedding algorithm is generally an NP-hard optimization problem with a substantial search space [Fi13]. Therefore, many different approaches and methods have been proposed to reduce the search space with customized embedding algorithms and optimization heuristics. Unfortunately, most of these algorithms are difficult to expand and adapt to different environments and constraints because they are highly tailored for specific infrastructures, frameworks, or application scenarios.

A typical development cycle for VNE is shown in Figure 1. Taken as a rule, the development of a new dedicated VNE algorithm starts with the informal documentation or formal specification of a set of requirements and actions. According to this specification, a prototype is implemented and evaluated in a simulation framework. Only when the simulation has been successful, the algorithm is integrated and tested in a realistic testbed before it goes into production. In the classical way of developing new embedding algorithms, the specification is often manually encoded and often manually integrated into a simulation environment, which brings time-consuming, error-prone tasks.

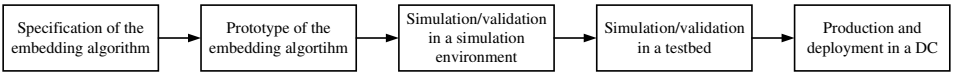

Fig. 1: Typical development cycle for VNE.

In this paper, we target this issue and propose a new methodology for model-driven virtual network embedding (MdVNE). In the model-driven development, executable code for different platforms can be automatically generated, an integrated simulation framework can fasten the prototyping of new algorithms by using the generated code and the correct implementation of specifications can be ensured. The MdVNE combines model transformation in the form of so-called triple graph grammars (TGGs) and integer linear programming (ILP) techniques with an optimization goal with linear equalities and inequalities which represent constraints in this paper. In the first step model transformation techniques are used to reduce the search space by pattern matching methods that generate families of possible mappings of VNs to SN elements. These mappings respect a set of given constraints handling rather structural or attribute conditions described via single graph patterns. In the second step, the ILP solving techniques take further decisions among the matched mapping candidates by selecting an optimal mapping with respect to a given set of constraints and optimization goals. On the contrary to the TGG constraints, the ILP constraints with a global scope go beyond single graph patterns and describe rather mathematical constraints over available resources.
Compared to existing solutions, the benefits of the proposed MdVNE approach include

- Developers specify embedding algorithms on a rather abstract level using a combination of first order logic constraints, inequalities, and model transformation rules.
- Prototypes of implementations are then generated from the high-level specification, leveraging state-of-the-art incremental pattern matching, model transformation, and ILP solving technologies.
- The generated low-level implementation respects the high-level specifications of embedding constraints and optimization goals by construction.
- The selected implementation techniques simplify the development of incremental reconfiguration of embeddings even including scenarios where embedding constraints and optimization goals are modified at runtime.
- The offered algorithm development and simulation framework supports the design of rather different categories of embedding and optimizing algorithms by combination and weighting of purely ILP-based and model transformation based approaches.

The remainder of this paper is organized as follows. After introducing the related work in Section 2 a running example is introduced in Section 3. The new mapping approach MdVNE is presented in detail in Section 4, followed by the evaluation in Section 5. Finally, the paper is concluded in Section 6.

2 Related Work

Virtualization of DC networks has been widely explored and a survey can be found in [Ba13]. As a result, many different algorithms [Gu10], [Ze15], [Xi12], [Zh13] for VNE have been proposed to maximize the resource utilization or minimize the cost for DCs. As the embedding problem is actually a case of the multi-way separator problem, it is NP-hard

and, therefore, not scalable without reducing the space size by heuristics or meta-heuristics [Fi13]. Guo et al. propose SecondNet [Gu10], which introduces a heuristic approach to map a subset of virtualized data centers (VDCs) to a tree-based DC. However, the authors only consider constraints on bandwidth and the number of virtual machines per physical server for reduced complexity. Zeng et al. [Ze15] consider the DC architecture and the traffic between virtual machines to minimize the overall communication costs between virtual servers and employs the commercial ILP solver Gurobi [Gu16] to solve the optimization problem. In addition, Xie et al. [Xi12] incorporate the time dimension into the VNE process and Zhani et al. [Zh13] include dynamic migration to adapt embedding decisions over time. The major advantage of MdVNE over the above-mentioned algorithms is that different architectures, constraints for resources, demands, and various optimization goals can be integrated and embedding decisions can be smoothly adapted in accordance to constraint changes on the fly.

In other network areas such as software-defined networks (SDNs) or wireless networks, the model-driven development is already used with promising results in order to increase the abstraction level, to create applications and algorithms independently of existing technologies and to verify them during development. In the SDN area, which makes the control and forwarding level independent of the physical network and can be a part of virtualization in DCs, Lopes et al. [Lo16] describe a model-driven approach to develop, verify and generate application-, controller- and network-independent code for SDN applications. In the area of wireless networks, Kluge et al. [Kl17] describe a model-driven approach to develop topology control algorithms with graph transformations while ensuring compliance with user-defined constraints and consistencies. However, none of the proposals can support both server- and network-end constraints simultaneously and thus are not directly suitable for VNE in complex DC environments.

3 Running Example

A typical scenario for DC operators consists of requests from customers for a customized VN infrastructure with switches, servers or network functions like firewalls. One typical VN request is a virtual cluster in which servers are connected to one central switch for creating a network environment [Ba11]. Such a topology is common in many enterprise scenarios e.g. to build a web application with clustered web servers. These network topologies may have different properties, resources, and constraints to be integrated and mapped by the DC that entails a high degree of configuration diversity.

In the following sections of the paper, a simplified example for a virtual cluster and DC is used to introduce the new mapping approach called MdVNE. The used scenario (Figure 2) and the metamodel for modeling the environment and generating the network instances (Figure 3) are now described in detail. Starting with Figure 2 (a), a snapshot of a DC with a queue of VNs, which should be mapped to the DC, is shown. Let us assume that some VNs are already mapped and all available positions (*slots*) to embed are occupied except

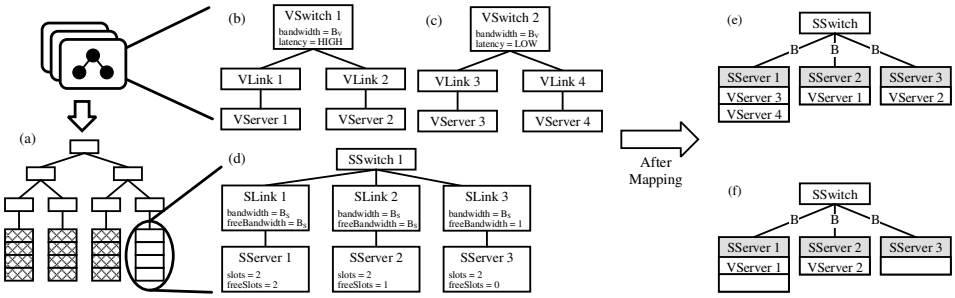

Fig. 2: Running example: (a) DC with multiple VNs in the mapping queue; (b) and (c) are examples for a virtual cluster; (d) is a subtree of the DC; (e) and (f) are two exemplary mapping solutions.

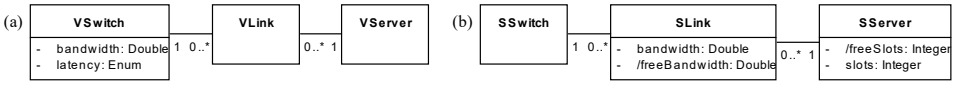

Fig. 3: Metamodels for (a) the VN and (b) the DC/SN.

the marked subtree. In the following, we only focus on the marked subtree from Figure 2 (d), which is called SN in the following. The mapping queue contains the VNs as shown in Figure 2 (b) and (c). Every VN has a central switch (*VSwitch*) with two links (*VLink*) each connected to a server (*VServer*). The bandwidth for these links is denoted as B_V (*VSwitch.bandwidth*) and the global VN has the service level agreement (SLA) that the latency must be *HIGH* or *LOW* (*VSwitch.latency*). A *LOW* latency means that the whole VN must be mapped to one substrate server in order to minimize the traffic delay whereas a *HIGH* latency has no restrictions. The SN is similar except that the bandwidth is defined for every *SLink* (*SLink.bandwidth*), every *SServer* owns a number of *slots*, each slot being able to host a virtual server and the bandwidth for the server internal traffic is assumed as unlimited. We further assume that every link in the SN has the same bandwidth B_S.

After defining the networks, the mappings of the VN to the SN are specified. These mapping constraints must be strictly adhered at all times because they represent e.g. technical conditions or SLAs with the customers. In this paper, the following constraints are defined:

(1) Every virtual switch must be mapped to one substrate switch or server.
(2) Every virtual server must be mapped to one substrate server.
(3) Every virtual link must be mapped to a substrate server or to one substrate link.
(4) Virtual networks with latency *LOW* must completely be mapped to one substrate server.
(5) The sum of all bandwidths of virtual links mapped to a substrate link must not exceed the available bandwidth of the substrate link.
(6) The sum of all virtual servers mapped to a substrate server must not exceed the available number of slots of the substrate server.

In Figure 2, two exemplary results (e) and (f) after the mapping process of both VNs to the SN are shown. In (e), the VN (b) with *HIGH latency* is mapped to *SServer 2* and *3*, and the VN (c) to *SServer 1*. Result (f) presents that VN (c) with *latency LOW* must be rejected because it cannot be mapped to one server after VN (b) is mapped to *SServer 1* and *2*.

While our set of constraints reflects a subset of requirements from real-world scenarios, further types of resources include CPU or storage capacity. Further constraint types include quality of service regarding response times and security levels reducing allowed mappings to certain subtrees in DC.

4 Mapping Approach

A typical workflow for a mapping process is presented in Figure 4, which consists of three phases: a preparation, mapping and deployment phase. In the preparation phase customers define the VN requests with their network functions, demands, SLAs, or change already existing virtual networks e.g. bandwidth. Furthermore, changes of the DC can be executed (add, remove, change server, switches,...) or the migration and shutting down of virtual networks. After that, the mapping phase is started in which the new mappings for the VN request are planned and activated/deployed in the deployment phase. This paper only focuses on the mapping phase while other phases rather concern technical details of communication networks.

Having defined and exemplified VNs and SN separately, our next goal is an explicit modeling of their mapping relationships. TGGs [Sc95] meet this requirement to specify the mappings between two graph-like structures via graph transformation rules. The combination of TGGs and ILP [LAS17] can be used to generate families of possible mapping candidates between two graphs that respect a set of given structural constraints and transfer this search space to an ILP solver for solving the optimization problem. The ILP solving techniques have been used to solve the resulting optimization problem (e.g., minimizing energy consumption) expressed as linear inequalities. The advantages over a classical VNE algorithm like [Ba11] or [Ze15] is that different constraints for resources, demands or optimization goals can be combined and easily added or extended so that a very wide range of applications are supported. Because TGG offer support for incremental updates of models, it is possible to adapt the incremental methods for this embedding approach in order to be able to efficiently deal with the highly dynamic system.

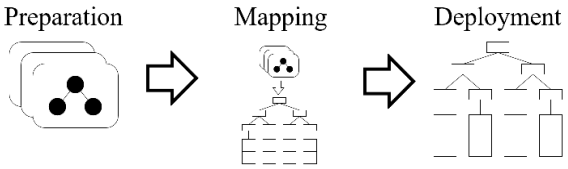

Fig. 4: Workflow for embedding of VN request.

4.1 Triple Graph Grammar

TGG is an approach to specify graph grammars, whose languages are sets of graph triples. Each graph triple consists of a so-called source and target graph plus a correspondence graph with traceability information between source and target. Given such a specification, the source graph contains the VNs, the target graph the SN and the correspondence graph the mapping relationships. For generating the mapping candidates between the VNs and the SN two steps are needed: (i) Creation of the SN and (ii) creation of the VNs with their (mapping) correspondences between the VNs and the SN. In the following, we assume that the SN is already created by the TGGs so we describe part (ii) now in detail.

In Figure 5, the TGG rules for generating the virtual networks and the mapping candidates are shown. Black elements are required context elements for executing the rule, whereas green elements marked by ++ are created by the rule. The naming convention for the elements and the correspondences are as follows: The name indicates the type e.g. *Sw* is an element of type *Switch* and *SwSw* is the correspondence between a virtual and a substrate *Switch*. The subscript letter *V* or *S* indicates if the element is part of the virtual or the substrate network, the letters e.g. *a* represent an index of the specific type. Two types of tuples are defined, one for the links and one for the correspondences. The first type of the tuples are for links and the letters *a* and *b* represent the source and target node e.g. for $L_{V(a,b)}$ Sw_{Va} is the source and Sr_{Vb} is the target element (Figure 5 d). The second type is for the correspondences and the first part represents virtual and the second the substrate element e.g. for $LSr_{(a,b),c}$ $L_{(a,b)}$ is in the VN and Sr_c in the SN.

Rule (a) creates a new virtual switch (Sw_{Va}) and mapping candidate ($SwSw_{(a,b)}$) to an existing substrate switch (Sw_{Sb}). Rule (b) and (c) are similar to rule (a) except that in rule (c) an attribute constraint Sr_{Sb}.*freeSlots*≥ 1 is added which means that this rule only matches if the attribute constraint is fulfilled. *FreeSlots* represent, similar to *freeBandwidth*, the number of slots that are not occupied by an active mapping before the mapping phase has

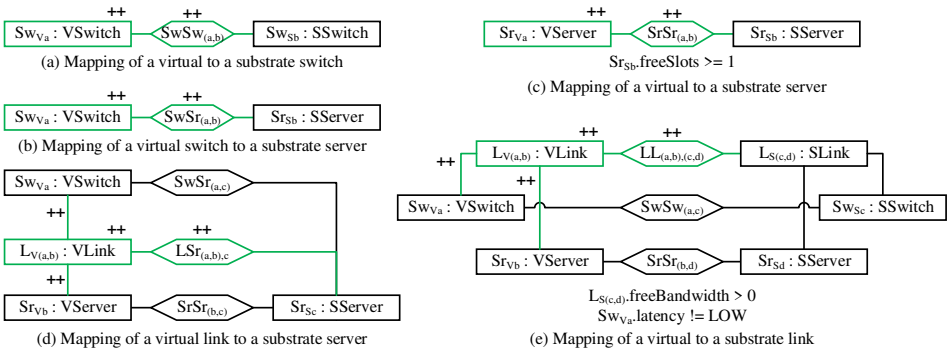

Fig. 5: TGG rules for creating the mapping of the VN to the SN.

started. Rule (d) creates a virtual link ($L_{V(a,b)}$) with a mapping candidate ($LSr_{(a,b),c}$) to an existing substrate server (Sr_{Sc}), if the mappings $SwSr_{(a,c)}$ and $SrSr_{(b,c)}$ already exist. The last rule (e) creates a mapping of a virtual link ($L_{V(a,b)}$) to a substrate link ($L_{S(c,d)}$) if the mappings $SwSw_{(a,c)}$ and $SrSr_{(b,d)}$ exist, *freeBandwidth* of $L_{S(c,d)}$ is greater than 0 and the latency of the switch Sw_{Va} is not *LOW*. This implicates that if Sw_{Va} has *latency LOW* only rule (d) can produce a link mapping. After the TGG rules are explained in detail, the connection of the constraints from section 3 to the TGG rules are summarized in Table 1.

Constraint	TGG rule	Annotation
(1)	(a) and (b)	A virtual switch can be mapped to a substrate server or switch.
(2)	(c)	A substrate server must have a free slot to map a virtual server.
(3)	(d) and (e)	A virtual link can be mapped to a substrate server or link.
(4)	(d) and (e)	Rule (d) must be executed to map a virtual link because rule (e) cannot be executed if *latency* = *LOW*.
(5), (6)	-	Is not represented by TGG rules

Tab. 1: Representation of the constraints from section 3 by the TGG rules from Figure 5.

To generate all mapping candidates, the TGG rules are executed on the example instance from Figure 2. An example of these mapping candidates of VN (c) from Figure 2 is shown in Figure 6, which shows a subset of all created elements and neglects *VLink 2* and *VServer 2* for brevity. During the creation of all correspondences e.g. $SwSr_{(1,1)}$ further constraints are internally created as integer (in-)equalities. These constraints are described and listed in the next subsection.

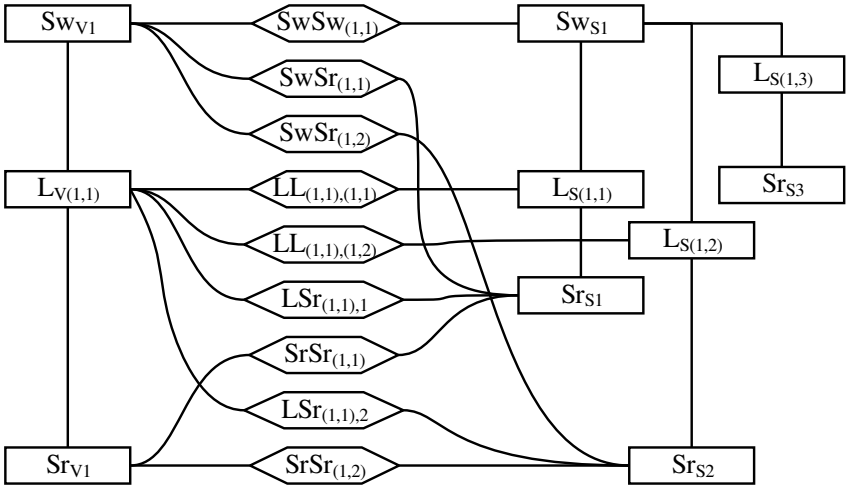

Fig. 6: All possible mappings after executing the TGG rules for the VN (c) from Figure 2. The *VLink 2* and *VServer 2* are neglected to avoid diagram clutter.

4.2 Linear inequalities

The execution of the TGG rules generates all potential candidates of mappings and integer variables that match the required graph structures and attribute constraints as specified in the rules. Additionally, ILP constraints are generated to ensure that each virtual network element is mapped to one and only one substrate element. In the next step, the linear optimization problem is solved by establishing linear inequalities for all constraints and passing them to the ILP solver. These linear inequalities specify which mappings will be activated and which are discarded e.g. if $SwSw_{(1,1)}$ is activated, $SwSr_{(1,1)}$ is discarded as they map the same virtual element and are thus mutually exclusive. To show the relationship between the mapping candidates and their integer variables in the inequalities, the name of the mapping candidate variables are retained with lowercase letters e.g. $swsw_{(1,1)}$ for their integer variables with the value 0 or 1. Other necessary parameters for establishing the linear inequalities are given in Table 2.

Variables for virtual network		Variables for substrate network	
M_V	Number of all virtual switches	M_S	Number of all substrate switches
N_V	Number of all virtual servers	N_S	Number of all substrate servers
K_V	Number of all virtual links	K_S	Number of all substrate links

Tab. 2: Different parameters for the linear inequalities.

Some of the advantages of this approach are the reduction of the search space by using graph grammars, and the inequalities automatically derived from the TGG rules if executed, which prevent a virtual element from being mapped several times, or that the dependencies between different mappings, e.g. $lsr_{(1,1),2}$ and $swsr_{(1,2)}$ are taken into account.

In the following, the constraints and their linear inequalities are described in detail and shown in a compact form in Table 3.

Constraint (1) is represented by TGG rule (a) and (b) (Table 1) and after their execution, the generated inequalities are presented in Table 3. The first line is generated by rule (a) considering that after the generation of all possible mappings between one virtual switch and all substrate switches, a maximum of one mapping can be selected. Therefore, the sum of all integer variables $swsw_{(i,j)}$ must be smaller or equal to 1 e.g. $swsw_{(1,1)} \leq 1$. The inequalities generated by rule (b) are very similar to rule (a) with the only difference that instead of a substrate switch a substrate server is used which leads to the following exemplary inequality e.g. $swsr_{(1,1)} + swsr_{(1,2)} \leq 1$.

Constraint (2) is represented by TGG rule (c) which leads to similar inequalities like rule (a) and (b) e.g. $srsr_{(1,1)} + srsr_{(1,2)} \leq 1$. The additional attribute condition $Sr_{Sb}.freeSlots \geq 1$ is not included in the linear inequalities because the pattern matching checks this condition before executing the rule and creating a potential mapping. Assuming that this constraint is not encoded in the TGG rules, this attribute condition would be manually encoded and added to the inequalities.

Constraint	Inequalities	TGG rules
(1)	$\forall i, 1 \leq i \leq M_V \mid \sum_{j=1}^{M_S} swsw_{(i,j)} \leq 1$	(a)
	$\forall i, 1 \leq i \leq M_V \mid \sum_{j=1}^{N_S} swsr_{(i,j)} \leq 1$	(b)
(2)	$\forall i, 1 \leq i \leq N_V \mid \sum_{j=1}^{N_S} srsr_{(i,j)} \leq 1$	
(3)	$\forall i, j, 1 \leq i \leq M_V, 1 \leq j \leq N_V \mid \sum_{p=1}^{N_S} lsr_{(i,j),p} \leq 1;$ $\forall i, j, p, 1 \leq i \leq M_V, 1 \leq j \leq N_V, 1 \leq p \leq N_S \mid$ $lsr_{(i,j),p} \leq swsr_{(i,p)}, lsr_{(i,j),p} \leq srsr_{(j,p)}$	(d)
	$\forall i, j, 1 \leq i \leq M_V, 1 \leq j \leq N_V \mid \sum_{p=1}^{M_S} \sum_{q=1}^{N_S} ll_{(i,j),(p,q)} \leq 1,$ $ll_{(i,j),(p,q)} \leq swsw_{(i,p)}, ll_{(i,j),(p,q)} \leq srsr_{(j,q)}$	(e)
(4)	No additional inequalities are needed.	-
(5)	$\forall i, j, 1 \leq i \leq M_S, 1 \leq j \leq N_S \mid L_{S(i,j)}.bandwidth -$ $\sum_{p=1}^{M_V} \sum_{q=1}^{N_V} ll_{(i,j),(p,q)} * Sw_{VP}.bandwidth \geq 0$	-
(6)	$\forall i, 1 \leq i \leq K_S \mid Sr_{Si}.slots - \sum_{N_V}^{j=1} srsr_{(i,j)} \geq 0$	-

Tab. 3: Representing linear inequalities for the constraints (section 3) and the TGG rules (Figure 5).

Constraint (3) is represented by TGG rule (d) to map a virtual link to a substrate server and rule (e) to map it to a substrate link. Compared to the previous constraints these rules have implications to express that the context elements and mappings are already selected. In rule (d) the implication is that a virtual switch and virtual server are already mapped to the same substrate server. The result are two additional inequalities e.g. $lsr_{(1,1),2} \leq swsr_{(1,2)}$, $lsr_{(1,1),2} \leq srsr_{(1,2)}$ meaning that if $lsr_{(1,1),2}$ is selected then $swsr_{(1,2)}$ must also be chosen because the link $L_{V(1,1)}$ can only be mapped to server Sr_{S2} if the switch Sw_{V1} is already mapped to Sr_{S2}. The inequalities for rule (e) are similar except that a virtual link is mapped to a substrate link and and a virtual switch to a substrate switch e.g. $ll_{(1,1),(1,1)} + ll_{(1,1),(1,2)} \leq 1$, $ll_{(1,1),(1,1)} \leq swsw_{(1,1)}$, $ll_{(1,1),(1,1)} \leq srsr_{(1,1)}$, and $ll_{(1,1),(1,2)} \leq srsr_{(1,2)}$.

Constraint (4) is represented by the combination of TGG rule (d) and (e) because if *latency* $=$ *LOW* then rule (e) is not executed ($Sw_{Va}.latency \neq LOW$) and, therefore, the virtual network must be mapped to one server (rule (d)), if possible. Consequently, no additional inequalities are needed to realize this constraint.

Constraint (5) cannot not be expressed by TGG rules because adding all mapping candidates is (actually) not expressible by eMoflon, the used to tool to specify TGGs and generate executable code. Therefore, these must be manually added to the generated inequalities. The constraint requires that the bandwidth of all mapped virtual links to a substrate link must no exceed the available bandwidth of this substrate link e.g. $L_{S(1,1)}.bandwidth - ll_{(1,1),(1,1)} * Sw_{V1}.bandwidth \geq 0$.

Constraint (6) is realized in a similar way as constraint (5) except that the sum of all virtual servers mapped to a substrate server must not exceed the available slots of the substrate server e.g. $Sr_{S1}.slots - srsr_{(1,1)} \geq 0$.

After the generation of all linear inequalities, these inequalities are handed over to Gurobi, the selected ILP solver, with the optimization goal to maximize the number of mapped virtual elements. Depending on the application or scenario, the optimization goal can also be modified and, e.g. be designed to minimize energy consumption.

Comparing the search space for this running example for a brute-force method and the TGG approach to generate all inequalities shows that the number of integer variables and inequalities could be reduced significantly. Generating all mapping candidates without checking additional attribute constraints result in more integer variables, e.g. $SrSr_{(1,3)}$ cannot exist because Sr_{S3} has no free slots, which implicates that the link mapping candidate $LSr_{(1,1),3}$ and $LL_{(1,1),(1,3)}$ can also not exist. The same holds for $SrSr_{(2,3)}$, $LSr_{(1,2),3}$ and $LL_{(1,2),(1,3)}$. In a brute-force approach, all attribute constraints *freeSlots* and *freeBandwidth* must additionally be encoded as inequalities while the pattern matcher did already check these constraints during the executing of the TGG rules. At the end, checking the *latency* during the generation of the mapping candidates reduces the number of integer variables and inequalities significantly. Looking at VN (c) from Figure 2 the TGG rule (e) is never executed and, therefore, no mapping candidates for $LL_{(a,b),(c,d)}$, no inequalities to express the implications for the source switch and target server and no attribute constraints have to be encoded as inequalities.

5 Evaluation

In this section, the presented MdVNE approach is evaluated and compared with a brute force and the Oktopus algorithm [Ba11], an established VNE algorithm for DCs, in relation to the runtime and the number of ILP variables and constraints. After introducing the simulation setup, the following three research questions are discussed:

RQ (1): How does the runtime, ILP constraints and variables change if the number of servers in the SN increases?

RQ (2): How does the runtime behave in comparison to a brute force mapping approach and the Oktopus algorithm in a specific scenario?

RQ (3): Does the reduction of the search space for the mapping candidates by MdVNE offer advantages with respect to the total runtime compared to a brute force mapping approach?

5.1 Simulation setup

The structure and underlying scenario for the evaluation is based on the presented running example. As shown in Figure 2, a DC with a two-tier network infrastructure is used in which virtual networks are stored in a queue and mapped one after the other to the DC. The two-tier DC infrastructure consists of two aggregation switches each connected to a varying

number of racks (marked area in Figure 2 (a)), each with a top of the rack switch (ToR switch) and 10 servers with four slots. Each server is connected to the ToR switch with a bandwidth of 10, and the ToR switches with a bandwidth of 100 to each aggregation switch. The virtual networks are implemented as virtual clusters [Ba11], which have a central switch connected to all virtual servers with a bandwidth of two. The following evaluation will vary the number of racks in the DC and the number of virtual servers. All other parameters remain constant. In order to obtain a wide range of virtual server distribution configurations, a random sequence of virtual networks with virtual servers between 2 and 10 is generated and used for all further evaluations to map one virtual network after the other to the DC. The evaluation is done on an Intel Core i7-7700HQ with 2.80 GHz with Windows 10 (version 1703) and the Java SE Development Kit 8u141.

In the following, three approaches are presented and compared with each other. The first and second approach are the presented MdVNE and a brute force ILP mapping approach. They are quite similar to each other except that in the brute force approach no attribute constraints are used further restricting the execution of the rules, e.g. $Sr_{Sb} \geq freeSlots$ (Figure 5 c). These attribute constraints are encoded into the ILP problem by additional inequalities after variables and formulas have been generated for all possible mappings of virtual to substrate elements. As a last comparison, the established Oktopus algorithm [Ba11] is executed and evaluated. Because this algorithm is not based on graph transformations or the Eclipse Modeling Framework (EMF), other data structures in the background are used which makes the comparison more difficult. In addition, Oktopus uses heuristics to map the virtual networks in contrast to the MdVNE approach. A qualitative comparison of the three algorithms is nevertheless out of scope of this paper.

5.2 Results

In the following, the results in combination with the research questions are presented and discussed.

RQ (1): To answer the first research question, the MdVNE approach is used to map 40 virtual networks and increases the number of racks from 2 to 50. This corresponds to a total server count of 20 to 500. After all 40 virtual networks are mapped, an average value for the total runtime of the mapping process, the runtime of Gurobi, the ILP solver, the number of ILP variables and constraints are being calculated. The mapping process of the first network is ignored in this calculation, as many Java and EMF initializations take place and the system is not in a steady state. In order to obtain reliable results all simulations were performed three times with a maximum percentage deviation from the average value of 9 %. The evaluation of the runtime for the MdVNE approach over the number of racks can be found in Figure 7. The complete mapping process seems to have a polynomial growth that can be explained by the fact that the distribution of the virtual servers to different substrate servers generates mapping candidates in a combinatorial manner. The growth of the ILP solver runtime values depends on the internal heuristics of the solver but it seems to be

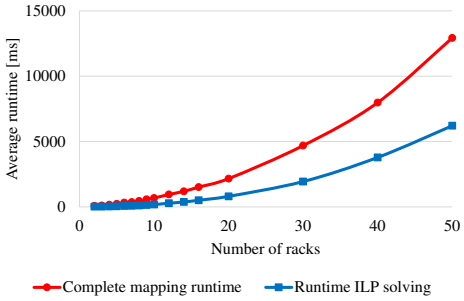

Fig. 7: MdVNE: Runtime of the complete mapping process and the ILP solver in ms over the number of racks in the SN.

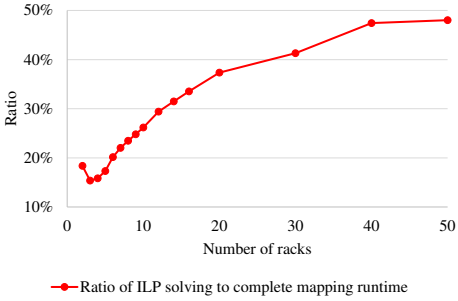

Fig. 8: MdVNE: Percentage of the ILP solver runtime of the complete mapping runtime over the number of racks in the SN.

a polynomial growth in this scenario. The influence of the ILP solver for the complete runtime of the MdVNE approach grows continuously from 15 % for three racks to 48 % at 50 racks (Figure 8) because the number of the generated ILP constraints and variables is also growing proportionally to the number of racks (Figure 9). The linear growth of both parameters can be explained by the steady increase of the elements in the model that are proportional to the number of combinatorial pairs of virtual servers as mapping candidates.

RQ (2): To answer the next two research questions the evaluation was modified to a scenario of 6 racks, which makes it possible to map the first 40 virtual networks from the queue into the DC. The result of the average runtime measurement is shown in Figure 10 with a logarithmic scaled runtime in ms over the number of mapped virtual networks e.g. 30 mapped virtual networks mean that 29 networks are already mapped.
As expected, the optimized and for this application scenario tailored Oktopus algorithm has the lowest constant runtime between 2 ms to 5 ms in this comparison. The efficient hand-tailored background data structure (in contrast to the usage of EMF models as data structures for the MdVNE and brute force algorithm) minimizes the internal Java overhead

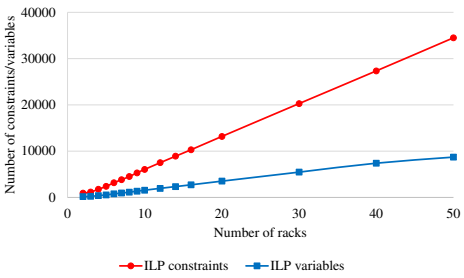

Fig. 9: MdVNE: Number of ILP integer variables and constraints over the number of racks in the SN.

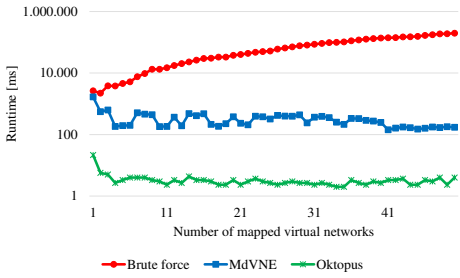

Fig. 10: Runtime of the MdVNE, brute force and Oktopus approach over the number of mapped virtual networks.

and, therefore, reduces the runtime, too. The MdVNE approach has also an almost constant runtime in a range of 140 ms to 500 ms, which is two magnitudes higher than the Oktopus algorithm e.g. 184 ms for MdVNE and 2 ms for Oktopus. This overhead is mainly caused by the usage of EMF as model framework. In both approaches, the higher values indicate that a virtual network had to be distributed on several substrate servers because all positions inside the tree have to be taken into account in a combinatorial manner. For the brute force mapping approach, a polynomial growth can be approximately assumed because for every combinatorial pair of elements in the increasing model an ILP variable is generated. This can be seen in more detail for the MdVNE and the brute force approach in Figure 11.

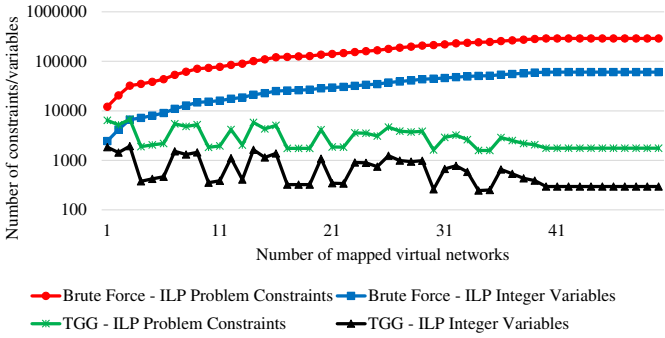

Fig. 11: ILP constraints and variables for the MdVNE and brute force approach over the number of mapped virtual networks.

RQ(3): We can see in Figure 11 that the ILP variables and constraints can be significantly reduced by using a more sophisticated ILP variable and constraint generation approach, which uses attributed graph transformations (TGGs) to filter/eliminate unfeasible mapping configurations early on in the MdVNE optimization algorithm. The result is a reduction in the runtime of the mapping process by two orders of magnitude (Figure 10). Generally, as many constraints as possible should be integrated into the TGG rules to reduce the complete mapping process.

5.3 Conclusion

In this section, we see that using the MdVNE approach it is possible to specify algorithms for the generation of ILP formulas to solve an optimization problem on a high level of abstraction (TGGs). The VNE problem to map a varying number of virtual clusters into a two-tier DC network by respecting constraints, attribute conditions and structural patterns could be realized and evaluated. The algorithm scales because of the linear runtime complexity and is thus in the same complexity class as the optimal tailored Oktopus algorithm. The overhead of the TGG execution phase is acceptable against the ILP solver phase especially when the number of servers increases in the DC. Furthermore, we see that with very little implementation effort (adapting the TGG rules) the search space of the ILP problem and,

therefore, the complete runtime of the mapping process can be significantly reduced. In the evaluation and running example only conservative structural and attribute constraints were specified in the TGG rules, but it is also possible to define more domain specific complex constraints to define sort of heuristic algorithm to reduce the search space even more and to improve the scalability.

6 Conclusion and Future Work

This paper presents a new methodology called model-driven virtual network embedding (MdVNE) combining model transformation and integer linear programming techniques to solve virtual network embedding problems. The model transformation and pattern matching techniques are used to generate families of possible mappings and reduce the search space by respecting a set of given constraints. Afterwards ILP solving techniques are used to select optimal mapping candidates. The advantage of this methodology is that the embedding algorithm can be specified on a rather abstract level and a prototypical implementation is automatically generated from this high-level specification. The development of new algorithms with this method can be fastened and, therefore, easily adjusted to other environments, applications and scenarios.

The evaluation has shown that it is possible to specify an algorithm for the VNE problem by using TGGs and an ILP solver. This algorithm scales in the range of 20 to 500 servers in a two-tier data center network with a linear runtime complexity. The reduction of the search space by the usage of pattern matching techniques reduces the runtime significantly.

To develop and evaluate different algorithms, the simulation framework and the metamodel of the DC and the VNs will be extended to support the definition of new types of constraints that take resources like e.g. CPU or demands e.g. latency into account. In addition, metrics to measure the qualitative properties of different algorithms will be added. Because of the high dynamics in this system, which requires a re-embedding or migration of existing mappings, e.g. changes in the DC or the virtual networks, incremental mapping scenarios are studied right now and will be supported in future versions of MdVNE.

References

[Ba11] Ballani, H.; Costa, P.; Karagiannis, T.; Rowstron, A.: Towards Predictable Datacenter Networks. ACM SIGCOMM Computer Communication Review 41/4, pp. 242–253, 2011.

[Ba13] Bari, M. F.; Boutaba, R.; Esteves, R.; Granville, L. Z.; Podlesny, M.; Rabbani, M. G.; Zhang, Q.; Zhani, M. F.: Data Center Network Virtualization: A Survey. IEEE Communications Surveys & Tutorials 15/2, pp. 909–928, 2013.

[Fi13] Fischer, A.; Botero, J. F.; Beck, M. T.; de Meer, H.; Hesselbach, X.: Virtual Network Embedding: A Survey. IEEE Communications Surveys & Tutorials 15/4, pp. 1888–1906, 2013.

[Gu10] Guo, C.; Lu, G.; Wang, H. J.; Yang, S.; Kong, C.; Sun, P.; Wu, W.; Zhang, Y.: SecondNet: A Data Center Network Virtualization Architecture with Bandwidth. In: Proceedings of the 6th International Conference. Co-NEXT, pp. 1–15, 2010.

[Gu16] Gurobi Optimization, I.: Gurobi Optimizer Reference Manual; 2015. URL http://www. gurobi. com/, 2016.

[Kl17] Kluge, R.; Stein, M.; Varró, G.; Schürr, A.; Hollick, M.; Mühlhäuser, M.: A Systematic Approach to Constructing Families of Incremental Topology Control Algorithms using Graph Transformation. Software & Systems Modeling/, pp. 1–41, 2017.

[LAS17] Leblebici, E.; Anjorin, A.; Schürr, A.: Inter-model Consistency Checking Using Triple Graph Grammars and Linear Optimization Techniques. In: Fundamental Approaches to Software Engineering. FASE, pp. 191–207, 2017.

[Lo16] Lopes, F. A.; Lima, L.; Santos, M.; Fidalgo, R.; Fernandes, S.: High-Level Modeling and Application Validation for SDN. In: Network Operations and Management Symposium. NOMS, pp. 197–205, 2016.

[SAE12] Sefraoui, O.; Aissaoui, M.; Eleuldj, M.: OpenStack: Toward an Open-Source Solution for Cloud Computing. International Journal of Computer Applications 55/3, pp. 38–42, 2012.

[Sc95] Schürr, A.: Specification of Graph Translators with Triple Graph Grammars. In: Graph-Theoretic Concepts in Computer Science. Vol. 903. Lecture Notes in Computer Science, pp. 151–163, 1995.

[Xi12] Xie, D.; Ding, N.; Hu, Y. C.; Kompella, R.: The Only Constant is Change: Incorporating Time-Varying Network Reservations in Data Centers. In: Applications, Technologies, Architectures, and Protocols for Computer Communication. SIGCOMM, pp. 199–210, 2012.

[Ze15] Zeng, D.; Guo, S.; Huang, H.; Yu, S.; Leung, V. C.: Optimal VM Placement in Data Centers with Architectural and Resource Constraints. International Journal of Autonomous and Adaptive Communications Systems 8/4, pp. 392–406, 2015.

[Zh13] Zhani, M. F.; Zhang, Q.; Simona, G.; Boutaba, R.: VDC Planner: Dynamic Migration-Aware Virtual Data Center Embedding for Clouds. In: Integrated Network Management. IM, pp. 18–25, 2013.