A Formal Optimization Model for 5G Mobile Network Slice Resource Allocation

Andrea Fendt[1], Christian Mannweiler[2], Lars Christoph Schmelz[2], Bernhard Bauer[1]
[1] University of Augsburg, Department of Computer Science, Augsburg, Germany
[2] Nokia Bell Labs, Network Management and Automation, Munich, Germany
andrea.fendt@informatik.uni-augsburg.de, bernhard.bauer@informatik.uni-augsburg.de,
christian.mannweiler@nokia.com, christoph.schmelz@nokia.com

***Abstract*—Network slicing is one of the key features of 5G mobile networks to cope with the diverging network requirements introduced by new use cases, like the IoT, autonomous driving and the Industry of the Future. Network slices are isolated, virtualized, end-to-end networks optimized for specific use cases. But still they share a common physical network infrastructure. Due to the dynamic life cycle of network slices there is a strong demand on efficient algorithms for mobile end-to-end network slice embedding. In this paper, a mathematical model for solving the offline Network Slice Embedding Problem formalized as a standardized Mixed Integer Linear Program is presented. A latency sensitive objective function guarantees the optimal network utilization as well as minimum latency in the network slice communication.**

***Index Terms*—5G, network slice, end-to-end mobile networks, virtual network embedding, latency, linear programming**

I. Introduction

With the fifth generation of mobile networks (5G) new use cases, like the Industry 4.0, autonomous driving and massive IoT are associated. However, these new applications as well as the traditional mobile broadband and voice use cases raise very diverse requirements on the mobile networks of the future. End-to-end mobile network slicing, i.e., using virtualized isolated subnetworks sharing the same infrastructure is seen as one of the key features to meet this challenge [1] [2]. Network Slices (NSLs) are use case specific and the provided network resources as well as the network performance will be bound by Service Level Agreements (SLAs) between a network infrastructure provider or a mobile service provider and a NSL customer or tenant [2]. For each NSL the actual required amount of network resources must be reserved in order to avoid penalties that might apply for SLA violations. However, radio frequencies and the associated resources, like bandwidth, are very scarce. An overprovisioning of these resources would be inefficient and non-beneficial for most applications. Therefore, a sophisticated resource allocation and optimization mechanism is strongly needed to avoid SLA violations while still using the Radio Access Network (RAN) resources efficiently and beneficially on behalf of the network infrastructure and service providers as well as the end-users. NSLs might be subject to ongoing change, set up, termination and modification [2]. Therefore, network management and planning need to be able to flexibly configure and modify NSLs as well as allocate required resources for the demanded network functions and services. As already explained in [3], this necessitates a high degree of automation of NSL preparation and deployment, as well as fast and efficient decision making on whether or not a Network Slice Embedding (NSE) is feasible. NSLs are intended to provide services to mobile and immobile end users. Such a service can be a standard telecommunication service, like a voice call, or a mobile broadband application, like video streaming or virtual reality as well as a highly safety critical data exchange service, i.e., for autonomous driving or remote surgery. Providing such a service usually involves the connection of several network functions. This is called application or network function chaining [2]. Fig. 1 shows a simplified version of a NSL architecture

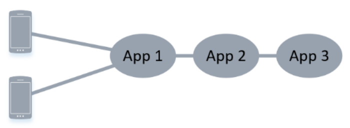

Fig. 1. Network Slice with Application Chain

with a network function chain, consisting of three applications. Two groups of User Equipments (UEs) are connected to this service. Each UE group is a group of mobile users that can use the same connections, i.e., is situated in the same cell of the physical network. The second Fig. 2 depicts a model of the physical network the NSLs are embedded in. It is assumed that each network function can run on any cloud server, if itself and its communication links fulfill the specifications and if there are enough available resources. Nevertheless, further constraints could be easily added to the model presented in this paper. Throughout this paper, the physical network as well as NSL resources and quality parameters are evaluated in a static context. For example, the variations in signal quality of a mobile network, caused by, e.g., weather and foliage as well as fluctuating resource demands of the NSLs, e.g., during the day, are not considered in this paper. However, solving the dynamic NSE is subject to ongoing research. This paper enhances the NSL resource allocation and optimization model and algorithm for end-to-end mobile networks proposed in [4]. The following research questions will be answered in this paper:

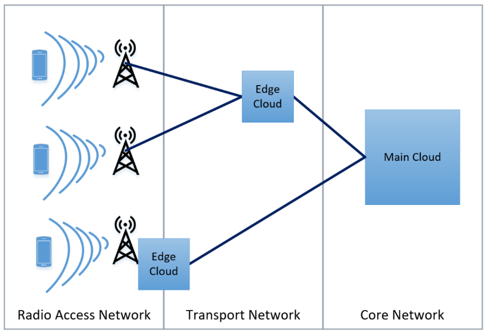

Fig. 2. Simplified End-to-End Physical Network Architecture

1) Can all NSLs be embedded into the physical network regarding their specifications and resource as well as application chaining constraints?
2) Which NSLs shall be embedded if there is a lack of resources?
3) What is the optimal embedding of the NSLs considering communication latency?

Application function chaining is introduced into the Integer Linear Program (ILP) proposed in [4]. In addition to that, path splitting is added to the model, i.e., the possibility to use several paths in the physical network to serve a single virtual link. This requires major changes in the graph, mapping and latency constraints. Furthermore, a new objective function for finding an optimal embedding of the NSLs is evaluated. The new objective function does not only maximize the sum of the embedded NSL weights, but also takes the weighted sum of the communication latencies into account. Also, node and link availability as well as reliability are added to the model as further network quality parameters. Due to path splitting, the NSL resource embedding model is now formalized as a Mixed Integer Linear Program (MILP). Solving this Program means trying the find a nearly optimal embedding respecting the qualitative as well as quantitative NSL requirements.

II. Related Work

Network slicing and its challenges are intensively discussed topics at the moment. For instance, Zhang et al. [5] present an approach for runtime optimization of throughput resources for NSLs. Wang et al. [6] are aiming at resource price balancing, considering dynamic offer and demand of NSL resources and Jiang et al. [7] use UE admission control to avoid network overloading at runtime. However, the mobile end-to-end NSE problem with fixed SLAs in the NSL preparation phase is, to the best of the author's knowledge, not covered in literature yet. This approach focuses on analyzing the provided and required network resources and network quality parameters in advance of the NSL deployment and on finding an optimal subset of NSLs to be embedded. The algorithm presented in this paper also provides a latency optimal mapping of the network applications on the server infrastructure and an embedding of the virtual communication links.
Virtual Network Embedding (VNE) is a well researched and understood mathematical problem. Although it is NP-hard [8] there is a large number of heuristics for various use cases. Also optimal solutions using ILP do exist. In the field of communication technologies, VNE is currently mainly used in fixed networks, but it is also evaluated for wireless networks, see for example [9] and [10]. Fischer et al. [11] present a survey on VNE for wired communication networks. A survey on recent related work in mobile NSE can be found in [12]. A lot of publications focus on runtime issues of NSE, like NSL isolation, resource partitioning and Physical Resource Block (PRB) assignment. NSL deployment and resource management at runtime are not covered by this paper. This work focuses on finding a (nearly) optimal utilization of the NSL resources in a mixed wired and wireless end-to-end network during the NSL preparation phase. Despotovic et al. [13] propose a scalable algorithm for solving the VNE problem optimally and efficiently. The presented approach allows to solve large problem instances with thousands of elements within only a few seconds. Unfortunately, the model has its limits. It doesn't apply for end-to-end mobile NSLs, since it doesn't consider network quality parameters, like link latency, availability and reliability. Beyond that, the VNetMapper, proposed in [13] is only capable of a one-to-one mapping of virtual to physical nodes and links. That means it is not possible to map several virtual nodes or links on one physical node or link. This is not a feasible constraint for the NSE, since the network infrastructure, e.g., antennas, base stations and transport links, must be shared among NSLs. The model proposed in this paper takes latency, availability and reliability constraints into account and optimizes the latency of the embedded NSLs. Furthermore, it is capable of many-to-one mappings and the embedding of network functions chains.

III. Network Slice Embedding Model

In this section a formal mathematical model for solving the NSE efficiently as an MILP is given. The model follows the design guidelines for efficient and scalable modeling of VNE problems, as proposed by of Despotovic et al. [13]. It is based on the ILP presented in [4], but is enhanced by a new latency sensitive objective function, as well as support for application chaining and path splitting and node as well as link availability and reliability restrictions.

A. *Definitions*

The following graph theoretical definitions (see [14] and [4]) are used to describe the graph structures of the physical and virtual networks:
An undirected Graph G is an ordered pair $G = (\mathcal{V}, \mathcal{E})$ consisting of a set of $n \in \mathbb{N}$ vertices $\mathcal{V} = \{v_1, v_2, \dots, v_n\}$ and a set of $m \in \mathbb{N}$ edges. An edge can have two adjacent vertices $e_{ij} := \{v_i, v_j\}$ for $i, j = 1, \dots, n$, often abbreviated by $e_{ij} := v_i v_j$. v_i is called the start node and v_j is called the end node. Both are also referred to as the ends of e_{ij}. In undirected graphs $e_{ij} = e_{ji}$ applies.
An undirected graph $P = (\mathcal{V}, \mathcal{E})$ comprising of a set of pairwise different vertices $\mathcal{V} = \{v_1, v_2, \dots, v_n\}$ and a set of

successive edges $\mathcal{E} = \{v_1v_2, v_2v_3, \dots v_{n-1}v_n\}$ is called a path of length $n \in \mathbb{N}$. v_1 is referred to as the start-vertex of P and v_n is called the end-vertex of P. P can also be abbreviated as $P = v_1v_2v_3 \dots v_n$. $\mathcal{P}_{ij}$ is defined as the set of all paths in an undirected Graph $G = (\mathcal{V}, \mathcal{E})$ with start-vertex $v_i \in \mathcal{V}$ and end-vertex $v_j \in \mathcal{V}$ with $i, j = 1, \dots, n$, $n \in \mathbb{N}$ and $i \neq j$.
A network graph $\mathcal{N} = (\mathcal{U}, \mathcal{C}, \mathcal{E})$ is defined as an undirected Graph $G = (\mathcal{V}, \mathcal{E})$ with $\mathcal{V} := \mathcal{U} \cup \mathcal{C}$. $\mathcal{N}$ consists of a set of UEs $\mathcal{U} := \{u_1, \dots, u_n\}$ with $n \in \mathbb{N}$, a set of cloud nodes $\mathcal{C} := \{c_1, \dots, c_m\}$ with $m \in \mathbb{N}$ and a set of edges $\mathcal{E} \subseteq \{u_ic_j, c_kc_l\}$ for all $i = 1, \dots, n$ as well as for all $j, k, l = 1, \dots, m$ with $k \neq l$.

B. Parameters of the Model

The parameters of this enhanced NSE model are based on the parameters used in the preceding model presented in [4]. The physical network is defined as a network graph $\mathcal{N} = (\mathcal{U}, \mathcal{C}, \mathcal{E})$ with $u_v \in \mathcal{U}$, the UEs, $c_w \in \mathcal{C}$, the cloud nodes, and $e_j \in \mathcal{E}$, the communication links (edges) between the cloud nodes connecting the applications deployed on them. The physical network is also referred to as the substrate in the following. Adjacent edges can be combined to a path $P_r \in \mathcal{P}_{vw}$ starting in an UE or cloud node $d_v \in \mathcal{U} \cup \mathcal{C}$ and ending in a cloud node $c_w \in \mathcal{C}$.
In this physical network $n \in \mathbb{N}$ NSLs, or virtual networks shall be embedded. Each of those n NSLs is represented as an undirected graph $\mathcal{N}_k = (\mathcal{U}_k, \mathcal{A}_k, \mathcal{L}_k)$ for $k = 1, .., n$ with $\mathcal{U}_k \subseteq \mathcal{U}$ a subset of all UEs in the substrate network, i.e., the UEs of the virtual networks are already embedded in the substrate by definition. $a_m^k \in \mathcal{A}_k$ describes the m-th application node of the NSL $\mathcal{N}_k$. NSLs are isolated from each other. Therefore, they do not share application node instance. Although several NSLs might share the same application types, distinct instances of those applications have to be defined for each NSL. Ultimately, the i-th virtual communication link of the k-th NSL is defined as $l_i^k \in \mathcal{L}_k$.
Finding a NSE means to determine a mapping of the application nodes a_m^k on the cloud nodes c_w and the corresponding virtual links l_i^k on suitable paths of edges e_j in the physical network. This mapping must respect NSL resource and property requirements. In this work, three types of resources and three network quality parameters are considered. However, this can easily be extended to further resources and parameters of interest. For the communication links the throughput $\overline{T}$ is the most important resources. In addition to that, the network quality parameters latency $\overline{L}$, availability $\overline{A}$ and reliability $\overline{R}$ are considered. For the cloud servers and application nodes the most important resources are the computational power P, the memory M provided by the server, or required by the application. In addition to that, the quality parameters availability A and reliability R are also considered for the nodes. To be able to model a NSE, the available and requested resources as well as the provided and required network quality have to be defined as distinct model parameters. For the physical network the throughput $\overline{T_j^s}$ is defined as the maximum throughput that can be transmitted on a mobile or fixed network communication link $e_j \in \mathcal{E}$. In this paper uplink and downlink are not distinguished. For communication via the RAN the available throughput is not only dependent on the bandwidth but also on the actual signal quality experienced by the users. For this approach an expected Channel Quality Index (CQI) has to be specified in advance. For each cloud node $c_w \in \mathcal{C}$ of the substrate the computational power P_w^s as well as the memory storage capacity M_w^s is specified. The resource requirements of the k-th NSL $\mathcal{N}_k$ are denoted as $\overline{T_i^k}$ for the required throughput on the i-th communication link $l_i^k \in \mathcal{L}_k$, P_m^k for the required computational power of an application node $a_m^k \in \mathcal{A}_k$ and M_m^k for its required memory storage capacity. Furthermore, the requirements of the NSL elements have to be fulfilled by the physical network elements they are mapped on. For instance, a physical link in the substrate network used by a virtual link of a NSL must not exceed a predefined latency and has to fulfill certain availability and reliability requirements. On the one hand $\overline{L_j^s}$ specifies the latency for a physical communication link $e_j \in \mathcal{E}$. Although in practice the latency is dependent on the actual throughput, it can be assumed that the latency remains constant if a certain throughput is not exceeded. The latency values used in this model can be seen as an upper estimation for the communication latency on a link which is under full load, but not overloaded. On the other hand $\overline{L_i^k}$ defines the maximum allowed latency for a virtual link $l_i^k \in \mathcal{L}_k$. Similarly, the availability A_w^s and reliability R_w^s of a substrate node c_w and the availability $\overline{A_j^s}$ and reliability $\overline{R_j^s}$ of the substrate edges e_j are defined. The availability and reliability requirements of the NSLs are stated as A_m^k and R_m^k for the required availability the m-th application node and $\overline{A_i^k}$ and $\overline{R_i^k}$ for the required availability and reliability of the virtual link l_i^k.

C. Variables of the Model

The following variables provide a formal description of the NSE for the MILP used to solve the optimization problem:

$$y_k := \begin{cases} 1 & \text{if } \mathcal{N}_k \text{ is embedded into } \mathcal{N}_s \\ 0 & \text{otherwise} \end{cases}$$

$$a2c_{mw}^k := \begin{cases} 1 & \text{if } a_m^k \text{ is mapped on } c_w \\ 0 & \text{otherwise} \end{cases}$$

$$l2p_{ir}^k \in (0,1) \text{ percentage of data transfer of } l_i^k \text{ mapped on } P_r \in \mathcal{P}_{vw}$$

Most of the variables are binary. This tremendously increases runtime efficiency of the embedding algorithm. However, to allow path splitting one continuous variable-type $l2p_{ir}^k$ is required. For convenience, it is defined:

$$p2e_{rj} := \begin{cases} 1 & \text{if } e_j \text{ is used in } P_r \\ 0 & \text{otherwise} \end{cases}$$

$$l2e_{ij}^k := \sum_r (l2p_{ir}^k \cdot p2e_{rj})$$

$l2e_{ij}^k$ combines the variables $l2p_{ir}$, mapping virtual links to paths, with the set of parameters $p2e_{rj}$, mapping paths to edges, to receive a virtual link to physical edge mapping $l2e_{ij}^k$. Note that $p2e_{rj}$ are no additional variables, but only model parameters since the paths are fixed for a certain substrate network. In addition to that, the $l2e_{ij}^k$ is not a new type of variables, but only another notation for the link mappings.

D. Mixed Integer Linear Program Formulation

The MILP formulation of the NSE is specified as follows:
maximize

$$\rho \frac{\sum_k (\omega_k \cdot y_k)}{\sum_k \omega_k} + (-1)(1-\rho)\frac{\sum_{k,i,j} \overline{L_j^s} \cdot l2e_{ij}^k}{\sum_{j,P_r \in \mathcal{P}} \overline{L_j^s} \cdot p2e_{rj} \cdot q}$$

subject to
Map-once constraints:

$$\sum_w a2c_{mw}^k = y_k, \forall k, m \tag{1}$$

Graph constraints:

$$\sum_{P_r \in \mathcal{P}_{vw}} l2p_{ir}^k = y_k, \forall k, i \text{ with } l_i^k = \{u_v, a_m^k\} \tag{2}$$

$$\sum_{P_r \in \mathcal{P}_{vw}} l2p_{ir}^k = a2c_{bv}^k, \forall k, i \text{ with } l_i^k = \{a_b^k, a_m^k\} \tag{3}$$

$$\sum_{P_r \in \mathcal{P}_{vw}} l2p_{ir}^k = a2c_{mw}^k, \forall k, i \text{ with } l_i^k = \{f_v^k, a_m^k\} \text{ and } f_v^k \in \mathcal{U}_k \cup \mathcal{A}_k \tag{4}$$

Resource constraints:

$$\sum_k \sum_i l2e_{ij}^k \cdot \overline{T}_i^k \leq \overline{T}_j^s, \forall j \tag{5}$$

$$\sum_k \sum_m a2c_{mw}^k \cdot P_m^k \leq P_w^s, \forall w \tag{6}$$

$$\sum_k \sum_m a2c_{mw}^k \cdot M_m^k \leq M_w^s, \forall w \tag{7}$$

Network quality constraints:

$$\sum_j l2e_{ij}^k \cdot \overline{L_j^s} \leq \overline{L_i^k} \cdot l2p_{ir}^k, \forall k, i, P_r \in \mathcal{P} \tag{8}$$

$$a2c_{mw}^k \cdot A_m^k \leq A_w^s, \quad \forall k, m, w \tag{9}$$

$$a2c_{mw}^k \cdot R_m^k \leq R_w^s, \quad \forall k, m, w \tag{10}$$

$$l2e_{ij}^k \cdot \overline{A_i^k} \leq l2e_{ij}^k \cdot \overline{A_j^s}, \quad \forall k, i, j \tag{11}$$

$$l2e_{ij}^k \cdot \overline{R_i^k} \leq l2e_{ij}^k \cdot \overline{R_j^s}, \quad \forall k, i, j \tag{12}$$

The objective function of maximizing the sum of embedded NSL weights, as proposed in [13] and [4], is augmented by a latency component. The NSL weights ω_k represent the relative utility of the NSLs, e.g., the revenue of the slice for the infrastructure provider. The weight $\rho \in (0, 1)$ should be chosen very high, for instance $\rho = 0.99$, to make sure the objective of embedding as many NSLs as possible as well as selecting the most beneficial ones is always the first priority. The second priority is to provide a mapping which minimizes the latency of the used communication links. The latency component of the objective function is multiplied with -1 since the weighted latencies are positive and their sum shall be minimized. The two components of the objective function are normalized before weighting them against each other. For the latency component this means to sum up the edge latencies for all paths and edges multiplied with the number of links q in the denominator. In the numerator all latencies of the edges in the physical network are weighted with the percentage of usage of this edge by all links in all NSLs. This is a simple means of achieving that the optimization algorithm minimizes the mapping on links with high latency. Also, other objectives, like reducing energy consumption by being able to share cloud node servers among several application and shut down unused servers or reducing cost by choosing cheaper communication links and cloud servers, are imaginable and could be taken into account by the objective function as well. However, this has to be considered carefully since the complexity of the objective functions has a high impact on the runtime efficiency of the overall NSE algorithm [13].
When optimizing the above objective function, twelve different constraint types apply. The first eq. 1 states that each application node must be mapped on exactly on cloud server node [13] [4], if the k-the NSL has been embedded into the substrate. In this case, $a2c_{mw}^k$ sums up to 1 over all cloud sever nodes and $y_k = 1$. Otherwise, y_k will be 0, therefore $a2c_{mw}^k$ must be 0 for all cloud server nodes. There is one map-once constraint for each combination of k and m, i.e., for each application node in every NSL. Eq. 2 defines the graph constraints for the virtual link to physical path mapping, for those virtual links starting in an UE node. If the k-th NSL is embedded into the substrate network the sum of all link to path mappings for the virtual links starting in an UE node has to be equal to 1. Note that the $l2p_{ir}^k$ variables can take values from the interval $(0, 1)$, i.e., the link mapping can be split on several paths. The same goes for eq. 3 and 4. However, eq. 3 states that if an application node $a_b^k \in \mathcal{A}_k$ has been mapped on a cloud node c_w also the adjacent links of a_b^k need to be mapped one or split among several paths that start in c_w. Eq. 4 requires the same for the end nodes of the virtual links. The three graph constraint types are defined for all virtual links. Compared to the graph constraints used in [4], in this model it is now allowed to share the same cloud node among chained applications. This is achieved by allowing paths to start in an arbitrary cloud server node (not only in an UE node) and by adding paths that only consist of a so called free-self link, which leads from a cloud node to itself with zero latency, infinite throughput resources and a 100% availability and reliability. The resource constraints in ineq. 5 specify that the throughput for each edge e_j of the substrate must not be exceeded by the sum of throughputs used by the virtual links mapped to it. For link splitting the respective proportion of throughput resources is taken. Ineq. 6 and 7 define similar

restrictions for the computational power and the memory storage of the cloud server nodes in the substrate network. These resources must not be exceeded by the application nodes mapped to them. The resource constraints are based on the capacity constraints as proposed in [13] and [4]. The most complex constraints are the latency constraints, see ineq. 8. A simple version of the latency constraints description not allowing path splitting can be found in [4]. The latency constraints state that no virtual link can be mapped on a path in the substrate network which has an overall latency that is higher than the allowed latency of the virtual link. Ineq. 8 defines one constraint for each virtual link and physical path combination. Two cases need to be considered when evaluating ineq. 8. In the first case, it is assumed that a certain l_i^k is mapped on a path P_r with a proportion of $\alpha \in]0,1)$. The latency $\overline{L_j}$ of this edge will be multiplied by α and added to the sum. However, not the sum of latencies weighted by the proportion of link usage shall be considered, but rather the sum of latencies of the used edges shall be compared with the allowed latency for this link. All weights used in the sum of one latency constraint are equal to α, since the whole path is used with the same proportion $l2p_{ir}^k = \alpha$. Therefore, we also scale the right-hand side of the ineq. by $l2p_{ir}^k = \alpha$. Also note that for the edges P_r is not mapped on, the left side of ineq. 8, is multiplied by $l2e_{ij}^k = 0$. In the second case, when l_i^k is not mapped on the considered path P_r, the proportion $\alpha = l2p_{ir}^k$ will be zero, therefore the left-hand side of the ineq. 8 is zero, which is always smaller or equal to the right-hand side of the ineq., since it cannot become negative. Finally, the remaining network quality constraints refer to the availability as well as reliability of the links and nodes in the virtual and physical networks. Ineq. 9 specifies the requirement that applications can only be mapped on could servers which comply to their minimum availability. There is one constraint per application to cloud node mapping. The same goes for ineq. 10, referring to the application's reliability. For the required availability and reliability of the virtual links, the actual availability and reliability of all physical links e_j used in a path mapped to a virtual link l_i^k must be good enough to fulfill the link's requirements. As already seen for the latency constraints, the scaling introduced by path splitting on the left-hand side of the ineq. must be neutralized by applying the same factor to the right-hand side of the ineq.. The availability and reliability restrictions for link mappings do only consider the availability/reliability of the edges on the path, but not of the cloud server nodes visited on that path.

IV. Example, Implementation and Evaluation

In this section a simple example for an efficient NSE is presented. In addition to that, the algorithm and its implementation will be evaluated.
Fig. 3 shows the physical network with two UEs and three cloud server nodes connected by six edges. The first of two NSLs $\mathcal{N}_0 = (\mathcal{U}_0, \mathcal{A}_0, \mathcal{L}_0)$ to be embedded into the above described substrate network is shown in Fig. 4. It consists of one group of UEs connect to two application chains that

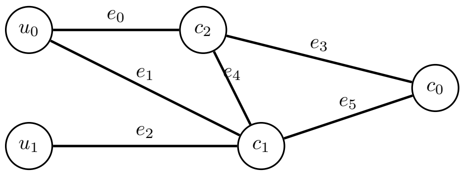

Fig. 3. Simple Example: Substrate Network

share the second and last element of the application chain a_2^0. The other NSL $\mathcal{N}_1 = (\mathcal{U}_1, \mathcal{A}_1, \mathcal{L}_1)$, shown in Fig. 5, only has one application chain but two user groups connected to it. The slice weights $\omega_0 = \omega_1 = 0.5$ are equal. The tables Tab. I and II show the characteristics and available resource of the physical network as well as the resource and network quality requirements of the NSLs.

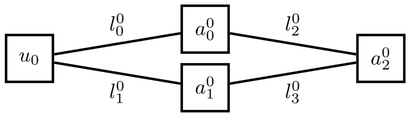

Fig. 4. Simple Example: Network Slice 0

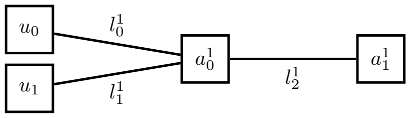

Fig. 5. Simple Example: Network Slice 1

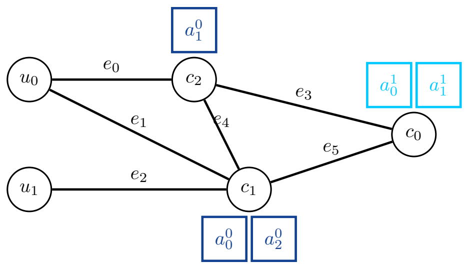

Fig. 6. Simple Example: Network Slice Embedding

In Fig. 6 an optimal embedding of the applications is shown. It is possible to embed both NSLs. The best cloud node to embed a_2^0 would be c_2, because it can be linked with the lowest latency. However, the availability and reliability requirements of a_2^0 exclude c_2 as a candidate. a_2^0 is mapped on c_1. c_0 would also have been feasible, but c_1 can be connected to u_0 with a lower latency. Due to their high CPU and memory capacity requirements the application nodes a_0^1 and a_1^1 can only be deployed on the main cloud c_0. Consequently l_2^1 has to be mapped on the path P_8, consisting of the free-self link of c_0. For the link l_0^1 path splitting is required between the path $P_3 = e_0 e_3$ and the path $P_5 = e_1 e_5$, since neither of the links e_0 or e_1 would have enough throughput resources.
The evaluation of medium sized problem instances without path splitting and latency optimization in [4] shows that this

TABLE I
NETWORK LINK PARAMETERS

Link	$\overline{T}$	$\overline{L}$	$\overline{A}$	$\overline{R}$
e_0	30	1	0.8	0.9
e_1	20	2	0.8	0.9
e_2	10	5	0.8	0.9
e_3	90	2	0.99	0.99
e_4	50	2	0.8	0.9
e_5	80	1	0.99	0.99
l_0^0	4	10	0.8	0.9
l_1^0	3	12	0.8	0.9
l_2^0	6	4	0.99	0.99
l_3^0	5	3	0.8	0.9
l_0^1	40	3	0.8	0.9
l_1^1	10	6	0.8	0.9
l_2^1	50	1	0.8	0.9

TABLE II
NETWORK NODE PARAMETERS

Node	P	M	A	R
c_0	200	200	0.99	0.99
c_1	40	40	0.99	0.99
c_2	50	50	0.8	0.9
a_0^0	30	30	0.8	0.9
a_1^0	20	20	0.8	0.9
a_2^0	10	10	0.99	0.99
a_0^1	60	60	0.8	0.9
a_1^1	60	60	0.8	0.9

kind of problem can be solved pretty fast. The runtime for a representative instance with 371 nodes and 559 in the substrate as well as 5 NSLs with 6 nodes and 18 links is only 43 sec. on a standard MacBook Pro 2015. However, evaluation with a dedicated simulator, written in Java programming language, shows that the standard MILP provided by the GNU Linear Programming Kit (GLPK) library of the GNU Project [15] is not capable of solving the NSE with path splitting, i.e., when using this specific combination of very restrictive constraints combined with using binary and continuous variables. Small problem instances, like the above example, can be solved optimally with the Solving Constraint Integer Programs (SCIP) Optimization Suite [16] within only approximately 0.1 sec.. The SCIP uses the branch-cut-and-price method to solve the MILP optimally, instead of the standard LP relaxation and cutting planes used by Linear Programming. The time needed for solving the NSE problem with and without path splitting for medium sized, randomly generated problem instances, with 61 nodes and 60 links in the substrate and 30 NSLs with 14 to 18 nodes and 9 to 14 links per NSL, highly varies depending on the specific network topology and the resource constraints. For some instances, solving the NSE optimally takes less than one minute, for others it takes nearly one hour to find a close to optimal solution. However, 88% of the tested instances are solved in less than 7 minutes, with allowing as well as without allowing path splitting. Path splitting does often decrease solving times significantly, but also cases where it significantly increases the runtime of the solver can be observed. For processing very large problem instances with thousands of nodes and links more efficiently, scalable VNE heuristics have to be evaluated. However, these implementations can't guarantee optimal or very close to optimal solutions, like the MILP approach proposed in this paper.

V. CONCLUSION AND FUTURE WORK

In this paper the NSE with network function chaining and path splitting has been formalized in a standardized, simple and easy to process MILP. It shows that a representation of the NSE for end-to-end mixed (mobile and core) NSLs as an MILP is feasible and beneficial. In addition, a latency sensitive objective function has been presented optimizing the overall utility of embedded NSLs as the first priority, but also proposing an embedding with minimum latency for the virtual links of the end-to-end NSLs. Although path splitting makes the problem pretty complex, out-of-the-box MILP solvers can find good, that means close to optimal, solutions within reasonable time. In future work other heuristics and algorithms for making the NSE scalable for very large problem instances and solving smaller ones even more efficiently will be evaluated.
Beyond that, network slices as well as their underlying substrate networks are subject to persistent change. In this paper resource demand and supply are assumed to be constant over time. Future work will include considering known and seasonal changes in resource availability and network quality parameters of the physical network and changes in the NSL requirements. Furthermore, uncertainties in these parameter need to be considered, when deploying several NSLs with fixed SLAs on a common end-to-end mobile network infrastructure.

REFERENCES

[1] Nokia, "Dynamic end-to-end network slicing for 5G", Espoo, Finland, 2016.
[2] 3GPP, "Study on management and orchestration of network slicing for next generation network", TR 28.801 V15.0.0T, 2017.
[3] A. Fendt et al, "A Network Slice Resource Allocation Process in 5G Mobile Networks", in Proceedings of the 12th International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS-2018), Matsue, 2018, pp. 695-704.
[4] A. Fendt et al., "A Network Slice Resource Allocation and Optimization Model for End-to-End Mobile Networks", in IEEE 1st 5G World Forum (5GWF18) Conference Proceedings, Santa Clara, 2018.
[5] H. Zhang et al., "Network Slicing Based 5G and Future Mobile Networks: Mobility, Resource Management, and Challenges" in IEEE Communications Magazine, 2017.
[6] G. Wang et al., "Resource Allocation for Network Slices in 5G with Network Resource Pricing" in IEEE Global Communications Conference, 2017.
[7] M. Jiang, M. Condoluci and T. Mahmoodi, "Network slicing management & prioritization in 5G mobile systems" in European Wireless 2016, Oulu, 2016, pp. 1-6.
[8] D. Andersen, "Theoretical approaches to node assignment", Computer Science Department, 2002, Unpublished Manuscript.
[9] R. Riggio et al., "Scheduling Wireless Virtual Networks Functions" in IEEE Transactions on network and service management, Vol. 13, No. 2, 2016.
[10] I. Tsompanidis, A. Zahran and C. Sreenan, "A Utility-based Resource and Network Assignment Framework for Heterogeneous Mobile Networks" in 2015 IEEE Global Communications Conference, San Diego, 2015.
[11] A. Fischer et al., "Virtual Network Embedding: A Survey" in IEEE Communication Surveys & Tutorials", Vol. 15, No. 4, 2013.
[12] M. Richart et al., "Resource Slicing in Virtual Wireless Networks: A Survey" in IEEE Transactions on Network and Service Management, VOL. 13, NO. 3, 2016.
[13] Zoran Despotovic et al., "VNetMapper: A Fast and Scalable Approach to Virtual Networks Embedding", 2014 23rd International Conference on Computer Communication and Networks (ICCCN), Shanghai, 2014, pp. 1-6.
[14] Reinhard Diestel: "Graphentheorie", 3rd edition, Springer, Heidelberg, 2006.
[15] Free Software Foundation, "GLPK (GNU Linear Programming Kit)", https://www.gnu.org/software/glpk/, accessed Feb. 21th, 2018.
[16] A. Gleixner et al., "The SCIP Optimization Suite 6.0", Optimization Online, 2018.