Generalized and Resource-Efficient VNet Embeddings with Migrations

Gregor Schaffrath  
T-Labs / TU Berlin  
Berlin, Germany  
grsch@net.t-labs.tu-berlin.de

Stefan Schmid  
T-Labs / TU Berlin  
Berlin, Germany  
stefan@net.t-labs.tu-berlin.de

Anja Feldmann  
T-Labs / TU Berlin  
Berlin, Germany  
anja@net.t-labs.tu-berlin.de

ABSTRACT
Network virtualization technology is believed to be a key enabler for the “Future Internet” as it allows to overcome existing ossifications and facilitates fast innovations. A crucial feature of this technology is the decoupling of virtual networks from the underlying substrate: Virtual networks (VNets) can be mapped to those locations (subject to certain specification constraints) in the substrate network such that they consume the least amount of resources. Therefore, resource-efficient embeddings are one of the main concerns in network virtualization.

This paper presents an algorithm for a very general class of VNet embedding problems. In contrast to related literature, our algorithm supports (cost-aware) migrations and reconfigurations, which is crucial especially if VNet requests are not known in advance. It avoids the inefficiencies of heuristics where virtual nodes and links are mapped separately, and it can handle a large class of links. Moreover, it features a high mapping flexibility, allowing, e.g., to respect placement policies, resource prioritization, and realization of VNet requests using multiple substrate resources.

In addition, the paper introduces a simple formal language that can be used to describe and communicate VNet requests and substrate resources in an automated and unambiguous manner.

1. INTRODUCTION
Network virtualization is a novel paradigm that decouples services from the underlying substrate network and enables the network provider to offer services with Quality-of-Service/Quality-of-Experience guarantees by using appropriate resource provisioning.

At the heart of network virtualization lies the idea of dynamic embeddings: Given a substrate network of network elements with different capabilities and connections, virtual network (VNet) requests are satisfied by mapping the virtual network components onto suitable resources. For instance, a multinational company may request a virtual network that connects their branches in different countries by low-latency/low-jitter communication links that can be used, e.g., for teleconferencing. In another example, a scientific institution such as CERN may request access to a virtual network consisting of a large number of CPUs in order to distribute the computational load from experiments in the particle collider. Finally, a university lab may want to evaluate their networking protocols in a controlled but realistic environment. Thus it may request access to a set of small virtual networks each with different bandwidth and latency characteristics.

While the idea of being able to dynamically embed services or networks where (and, in terms of energy conservation also: when) they are most useful sounds appealing, the computation of good embeddings is challenging. In contrast to traditional embedding theory or classic embedding algorithms used, e.g., for circuit design, virtual network requests arrive over time and need to be handled online—without changing existing embeddings if possible, and minimizing migration costs otherwise. This has several algorithmic implications. On the positive side, a good and resource-efficient embedding is usually also a robust embedding in the sense that many future requests can be satisfied without the need to substantially change the embeddings of already embedded VNets. However, this is not always possible, especially once other VNets expire; in this case minimizing the resource cost is not the only objective, rather migration costs have to be taken into account as well.

1.1 Related Work
Virtualization and in particular network virtualization is a promising technology [17] [18] [23] to overcome the “ossification” of the Internet [7] and facilitates the co-existence of innovation and reliability [20]. For a recent review on network virtualization, the reader is referred to the surveys [6] and [11].

The VNet embedding problem has already been studied in various settings. Unfortunately, many variants of the VNet embedding problem are computationally hard: even if all VNet requests are given in advance, the offline optimization problem with constraints on virtual nodes and virtual links can be reduced to the NP-hard multi-way separator problem (e.g., see [2] for a survey). Thus, there is a large body of literature on heuristic solutions. For example, Fan and Ammar [10] study dynamic topologies to accommo-
date communication requirements that vary over time, and propose heuristic methods for constructing different flavors of reconfiguration policies. Only bandwidth constraints are taken into account. In [22], Zhu and Ammar consider virtual network assignment problems with and without reconfiguration; the authors propose subdividing heuristics and adaptive optimization strategies to reduce node and link stress. Ricci et al. [19] also take a heuristic approach and describe a simulated annealing solver for Netbed. To deal with the computational hardness, Yu et al. [16] advocate to rethink the design of the substrate network to simplify the embedding, e.g., by allowing to split a virtual link over multiple paths and perform periodic path migrations. The focus of the work by Butt et al. [4] is on re-optimization mechanisms that ameliorate the performance of the previous VN embedding algorithms in terms of acceptance ratio and load balancing; their algorithm is able to prioritize resources and is evaluated by simulations. Finally, VNet embeddings have also been studied from a distributed computing point-of-view: Houidi et al. [12] present a distributed algorithm where agent-based nodes communicate to map and load-balance requests on the substrate.

An interesting aspect of the VNet embedding problem is that a mapping algorithm has the flexibility to simultaneously assign both nodes and links to substrate elements, unless the VNet request specifies explicit locations. Many approaches in the literature pursue an approach to first map the nodes and then solve the link mapping problem separately. In their VISA’09 paper, Lischka and Karl [14] aim to map nodes and links in the same stage by employing subgraph isomorphism detection [8]. Their simulations provide evidence that their heuristic results in better mappings.

In contrast to the literature reviewed above, we take a more formal approach. We present a general formulation of the embedding problem as a mathematical program. We believe that this formulation gives interesting insights into the nature of the dynamic VNet embedding problem. Moreover, it allows to compute optimal embeddings for small network, and hence, e.g., features all the benefits of simultaneous node and link embeddings described in [14]. While finding optimal solutions is time-intensive for larger networks, fast approximate or heuristic solutions can be computed with standard software tools such as CPLEX [9] and Ip_solve [15]. In particular, the MIP formulation has the advantage that there is no need to reinvent new pruning heuristics for each embedding problem variant; often such heuristics are also unlikely to be faster than the sophisticated algorithms incorporated into CPLEX or Ip_solve.

Resource allocation problems in networks have been formulated as mathematical programs before. For instance, Kumar et al. [13] describe an approach to solve a virtual private network (VPN) tree computation problem for bandwidth provisioning. The paper closest to ours is by Chowdhury et al. [5] who present a mixed integer linear embedding program through substrate network augmentation. The authors pursue a relaxation strategy, applying randomized and deterministic rounding to find approximate solutions. Our work is orthogonal to [5], as our focus is not on performance of the program evaluation, but rather on its generality. In particular, we extend the expressiveness of the mathematical program in several dimensions, without sacrificing linearity.

Finally, it remains to mention that migration in virtual networks has also been studied from a QoS/QoE point-of-view recently. In [3], we describe a competitive online algorithm that “embeds” a single server at locations which minimize the access cost of mobile clients while taking into account migration costs.

1.2 Our Contribution

The main contribution of this paper is a novel network embedding algorithm which is formalized as a linear Mixed Integer Program (MIP) that can be solved (or approximated) with the help of standard software tools. Our algorithm can be run using different objective functions, and allows to map virtual nodes and virtual links simultaneously onto a substrate network, which in general leads to more efficient embeddings [14]. Virtual links can be implemented as a multi-path flow in the substrate network, which allows for larger aggregated bandwidth and, under certain circumstances, faster embeddings [16]. The main focus of this paper however is on the flexibility or generality of the embedding program, while enforcing linearity. In particular, we extend existing mathematical programs in several dimensions and support:

- Cost efficient reconfigurations and migrations;
- Embeddings of both symmetric and asymmetric full-duplex as well as half-duplex virtual links on both symmetric and asymmetric full-duplex and half-duplex substrate links;
- Embeddings of shared virtual links with several endpoints (e.g., hubs and bridges) and onto shared substrate links with several endpoints;
- Embeddings across resources, e.g., splitting a request for memory among SW AM and RAM;
- Provider-side placement policies as well as resource prioritization (e.g., bandwidth over latency).

As the type and arrival time of VNet requests is hard to predict, the possibility of reconfigurations and migration is crucial.

This paper also reports on some of our experiments evaluating the algorithm’s performance. We believe that the runtimes are reasonable and can still be improved. Thus, we have fully implemented this algorithm in our prototype. Finally, we also discuss a resource description language that can be used to communicate virtual network requests as well as embeddings in a formal manner.

1.3 A Motivating Example

In order to acquaint ourselves with the VNet embedding problem, Figure 1 shows a concrete example. The
substrate network consists of four nodes $S_1, ..., S_4$. Here, nodes \{${S_1, S_3, S_4}$\} are connected via a shared communication channel (with three endpoints). Nodes \{${S_1, S_2}$\} are connected by two separate unidirectional links (henceforth called a full-duplex link), while nodes \{${S_2, S_3}$\} communicate via a shared link, henceforth called a half-duplex link—where the capacity is shared between the two directions.

Consider the VNets, VNet 1 and VNet 2 where VNet 1 is already embedded onto the substrate network, namely on the substrate node $S_1$. In order to embed VNet 2, our algorithm takes the already embedded VNet into account; only in case a much more efficient embedding can be obtained by migrating (parts of) VNet 1—i.e., if the migration cost can be amortized against a substantially better embedding of VNet 2—, our algorithm will do so by reconfigurations.

![Figure 1: Example for the embedding problem.](image)

1.4 Paper Organization

The remainder of this paper is organized as follows. In Section 2 we provide some relevant background information on the network virtualization architecture, its roles and the prototype we are building. The main ideas and concepts behind our embedding algorithm are presented in Section 3. The mathematical embedding program together with a sample objective function is then described in full detail in Section 4. Section 5 reports on our experiments. We round off our contribution by discussing a formal resource description language that can be used to communicate embeddings, as well as with some remarks on implementation aspects (Section 6). In Section 7 the paper concludes.

2. VIRTUALIZATION ARCHITECTURE

This section briefly reviews the virtualization architecture we are currently implementing and summarizes the four roles we envision. We constrain ourselves to the main concepts which we believe are relevant to understand the network embedding context. For more details, the reader is referred to [21].

Our network architecture is motivated from both business and technical perspectives. Basically, we identify several roles: The (Physical) Infrastructure Provider (PIP), which owns and manages an underlying physical infrastructure (called “substrate”); the Virtual Network Provider (VNP), which provides a holistic virtual network by assembling resources from different PIPS, Virtual Network Operator (VNO), administrating and operating a virtual network in support of a service provided by a Service Provider (SP) on top of the VNet. Every one of these roles is assumed to be interested in different abstraction levels of a virtual network and following its own optimization goals constrained by the requests of its customers and offers of its providers.

These assumptions have two major implications:

- There is a need for a Resource Description Language (RDL) to describe arbitrary networks at arbitrary levels of detail, as well as the mapping relations between the different descriptions. Moreover, as the level of specificity is related to a provider’s flexibility w.r.t. optimization and can therefore be considered an economic factor, incomplete VNet descriptions are likely to occur; thus descriptions are refined as they are handed through the role hierarchy.

- Network mapping will happen in several steps on different levels of the role hierarchy rather than in one single process: namely a PIP role needs to be able to operate its substrate (e.g., a cluster of machines with shared NFS volume to facilitate live migration), and a VNP role needs to map VNets on the set of cooperating PIPs (e.g., national sites if the role is taken by a national ISP, or worldwide contractors in the case of a globally operating entity).

The main focus of this paper is on the computation of such mappings. An appealing property of our algorithm is that it can be used for a variety of notions of cost-optimality, and it can be computed by standard and optimized software tools. Moreover, we will also sketch a formal language that can be used to describe (or specify) both service and virtual network requests, as well as the corresponding embeddings.

3. KEY CONCEPTS

The main objective of any embedding algorithm is to satisfy most VNet requests that arrive over time by mapping them onto a given substrate network. On the one hand each VNet node imposes certain requirements and constraints with regards to its embedding: for instance, a virtual node may require a 100 MHz CPU and may only be mapped onto Linux nodes in the US. Similarly, VNet links may need at a minimum of 10 MBit/s. On the other hand substrate nodes and links offer certain resources that can be shared among the virtual networks.

We postulate the following properties for our embedding algorithm and then present the main ideas of our VNet embedding program that meets these goals.

- All VNet requirements and constraints have to be fulfilled via a feasible resource allocation governed by the substrate network constraints;
- The substrate resource usage for VNet embedding is minimized while respecting the previous constraints;
The arrival of a new VNet should cause minimal changes to the existing embedding of prior V Nets (i.e., since VNet migration cost is non-zero there is a trade-off between migration cost and the benefit of a superior embedding).

In the following, we introduce the main ideas of our embedding algorithm. In doing so, we will also define the main constants and variables of the mathematical program. Please note that this section serves to get the reader familiar with our approach, and that the complete formal program description will follow later in the paper.

### 3.1 Graph Representation

Our embedding algorithm must be able to deal with (both virtual and substrate) links representing shared communication channels, i.e., links with several end points. To describe virtual and substrate networks as classic graphs \( G = (V, E) \) consisting of vertices \( V \) that are connected pairwise by edges \( E \), we introduce the concept of network elements \( (\text{NE}s) \): network elements represent both nodes \((\text{NE}s)\) and links \((\text{NE}Ls)\). Network elements are connected by interfaces, which form the edges of the graph.

We distinguish between virtual network elements of the VNet \((\text{NE}V) \) of virtual nodes and virtual links) and substrate network elements of the substrate network \((\text{NE}S) \) of virtual nodes and substrate elements. In principle, any virtual node can be mapped onto any substrate node; moreover, depending on the requirements, a virtual link can be embedded onto a substrate link, or onto a set of paths in the substrate network (resulting in a multi-flow embedding).

In our example of Figure 1 \( \text{NE}V \) = \{\( S_1, S_2, S_3, S_4 \)\} and \( \text{NE}S \) = \{\{\( S_1, S_2 \), \( S_1, S_3 \), \{\( S_1, S_3, S_4 \)\}\}\}. The main purpose of the embedding algorithm is to find a mapping of the virtual networks and their elements to the network elements of the substrate. To handle our diverse classes of links, including multiple parallel links and links with several endpoints, we replace each link with a vertex and add appropriate graph edges. Figure 2 shows the transformation for our example.

![Figure 2: Transformation of network to graph representation.](image)

### 3.2 Where and How to Map?

For the MIP, we use the binary matrix new\((u,v)\) to denote whether a virtual network element \( u \) in \( \text{NE}V \), either a node or a link, is mapped to a substrate network element \( v \) in \( \text{NE}S \) (new\((u,v) = 1\)) or not (new\((u,v) = 0\)).

A substrate element allocates resources for all virtual elements it hosts. To describe these allocations, we introduce the variables alloc\((u,v)\) which capture the resources used by a virtual node while the requested resources are represented by the constant matrices req\((u,v)\). To ensure that the sum of the allocated resources never exceeds the substrate element’s capacities we use a constant capacity matrix cap\((v)\) which specifies \( v \)’s capacity.

It is not always possible to map a virtual network element to any arbitrary substrate element. For example, a virtual node requiring CPU resources cannot be mapped onto a node that provides only storage, or a virtual node may only be mapped onto substrate elements within the US. The constant binary matrix suit\((u,v)\) specifies whether \( v \) is suitable to host network element \( u \) (suit\((u,v) = 1\)) or not.

Our mathematical program considers placement restrictions: A provider may want to bias or fix a mapping for a specific VNet according to internal placement policies. We thus use a constant weight matrix w e i g h t\((u,v)\) to introduce a cost of for each node placement; these weights can also be used to prioritize certain resources over others in the objective function (e.g., maximizing bandwidth rather than minimizing latency).

### 3.3 Link Types

Next we discuss how we handle the different link types: If the bandwidth in both directions is the same we call a link symmetric, otherwise it is called asymmetric. A full-duplex link supports traffic in both directions independently. A full-duplex link can be regarded as two independent unidirectional links. A shared (non-switched, hub-like) channel is referred as half-duplex link.

Note, half-duplex links are symmetric by nature. To handle the different link types, we explicitly distinguish between two classes of resource types \( \text{RT} \), namely virtual resource types \( t_V \) and substrate resource types \( t_S \). For example,

\[
\begin{align*}
  t_V &= '/\text{link/symmetric/bandwidth}' \\
  t_S &= '/\text{link/upstream/bandwidth}'
\end{align*}
\]

denotes a symmetric bandwidth-type link, while

\[
\begin{align*}
  t_S &= '/\text{link/upstream/bandwidth}'
\end{align*}
\]

corresponds to an asymmetric resource. In our embedding program, we assume a proportional relationship between \( t_V \) and \( t_S \), that is, we consider a proportional factor prop\((t_V,t_S)\).

Interestingly, the fact that \( t_V \) and \( t_S \) differ in the VNet specification and the corresponding implementation in the substrate, e.g., in terms of direction, is not only useful for handling different link types but also for mapping nodes: Differentiating between \( t_V \) and \( t_S \) enables us, to, e.g., map RAM requests to SWAP memory, and vice versa, or to map memory requests to both RAM and SWAP memory.

An interesting property of our algorithm is that it allows to map shared communication channels. To handle shared
3.4 The Flow Problem

While a virtual node can only be mapped onto a single node on the substrate network, virtual links can be realized either as a single path or multiple paths within the substrate network. The aggregated resources along the paths must then satisfy the requirements of the virtual link while not exceeding the capacity limits of the substrate elements; for instance, the sum of the bandwidths over the different paths must equal the link’s bandwidth demand. This constitutes a flow problem. However, since we tackle placement and embedding at the same time this corresponds to a multi-commodity flow problem with a twist: The embedding algorithm is allowed to simultaneously choose the link’s endpoints.

Thus, our mathematical program ensures that the allocated flows are connected, consistent with the requirements and capacities, and also with the source and sink constraints. We enforce a flow preservation invariant, that is, we guarantee that the amount of flow arriving at a node equals the amount of flow leaving the node. However, we must exempt the source and the sink of the flow from this invariant: We ensure that the traffic leaving the source equals the demand of the virtual link; the link’s sink simply consumes the incoming flows. This is implemented via selector variables that render the constraint trivially true for endpoints (a tautology).

3.5 Migration Support

Next we add migration support by enabling re-configurations. Virtual network requests typically arrive over time and the provider faces the problem of how to embed a new VNet given the existing allocations of other requests. Clearly, a complete re-embedding of all requests is out-of-question, as this potentially comes at a high cost and with long outage times. However, a small local reconfigurations may reduce the overall resource overhead and improve the overall embedding substantially or even enable the embedding at all. We use the constant binary matrix \( old(u, v) \) to describe existing embeddings which is similar to \( new(u, v) \) and specifies whether a virtual network element \( u \) is currently mapped to a substrate element \( v \) \((old(u, v) = 1)\) or not \((old(u, v) = 0)\).

To account for the cost of migration, we introduce a constant penalty function \( penalty(u) \) that states a penalty for moving \( u \in NEV \). In addition, we use the binary variable \( mig(u) \) to denote whether \( u \) is migrated or not for this embedding or not. Note that node migration is typically more expensive relative to link migration, as link do not involve state or bulk data transfers but are simply re-instantiated; hence, in the remainder of this paper we assume \( penalty(u) = \epsilon \) for \( \forall u \in NEV \) and an arbitrarily small \( \epsilon > 0 \).

4. THE EMBEDDING PROGRAM

Based on the above ideas we next describe the MIP in more details, see Figure[3] for a summary.

While we introduced most sets, variables, and constants above we need a few extensions to account, e.g., for different resource types. For instance the capacity matrix explicitly depends on the resource type \( t_S \), and we define \( cap_{t_S}(v) \) to be \( v \)'s capacity w.r.t. \( t_S \) and \( cap_{t_S}(v, w) \) is defined analogously for edges \((v, w)\).

In the following, we refer to the set of value types as \( VT \); examples include maximum, minimum, or constant.

The requested resources \( req(u, t_V, s) \) depend on the type \( t_V \) and on the value type \( s \in VT \) of the request. The allocation variables \( alloc_{t_s}(u, v, t_V) \) represent the allocated resources of type \( t_S \) to host resources of type \( t_V \) on \( u \); similarly, \( alloc_{t_V}(u, v) \) denotes the hosted resources of type \( t_V \) on \( v \).

Similarly to \( RT \), we define the resource type set \( RT_f \) as the subset of \( RT \) that corresponds to flow \( f \) (e.g., (symmetric) bandwidth, up-/downstream-bandwidth, etc.). For allocation, we introduce \( flow_{t_f}(f, v, w; t_V) \) to denote the allocated resources of type \( t_S \) on substrate edge \((v, w)\) to host resources of type \( t_V \) for flow \( f \); \( flow_{t_V}(f, v, w) \) is the hosted resources of type \( t_V \) of flow \( f \) on substrate edge \((v, w)\).

4.1 Objective Function

When is a resource usage optimal? The answer depends on the goals of the mapping entity, and also relies crucially on the predictability of future resource requests. Even with good predictions, an optimal solution found at time \( t \) may be suboptimal upon the arrival of the next request at \( t' > t \).

This implies that the possibility of reconfigurations and migrations is vital in order to accommodate additional requests and to avoid inefficient use of resources. However, as reconfigurations, and especially migrations in the sense of virtual machine migrations incur costs in terms of outage, bandwidth consumption, administrative overhead (etc.), it is desirable to be able to weight migration benefits against incurred costs.

The study and discussion of optimal objective functions is
a topic on its own and it is beyond the scope of this paper. One advantage of our mathematical programming approach is the ease in exchanging different objective functions with only limited implementation effort. In the following, we hence concentrate on a simple objective function taking into account currently used resources as well as migration costs in an additive manner.

Concretely, in our prototype implementation (and in our experiments), we seek to minimize:

$$
\left[ \sum_{u \in NEV} \sum_{v \in NE} \left( \sum_{t \in RT} \text{weight}(u, v) \text{alloc}_{u}(u, v, t) \right) \right] + \sum_{u \in NEV} \text{penalty}(u) \text{mig}(u)
$$

### 4.2 Constraints

The embedding must fulfill various type, capacity, and other consistency constraints, see Figure 6 for a complete and formal constraint list.

**Nodes:** This constraints category is used to ensure that each virtual node is mapped to an appropriate substrate node. Recall, in contrast to links, nodes cannot be mapped to multiple substrate elements, and hence Constraint map_node is necessary to guarantee a unique mapping location. At the location where the node is mapped (and only there!), resource requirements must be fulfilled (Constraint set_new). Depending on the substrate resource type (minimum, maximum, or constant), the resource constraints are imposed in a different manner (Constraints req_min, req_max, req_const).

**Mapping:** The mapping constraints ensure that the substrate element has sufficient capacity (Constraint capacity) allocated (Constraint relate_V); moreover, it must be of the correct type (Constraint allowed).

**Resource-Variable Relation:** This set of constraints deals with the relation between the resource types $t_s$ that host resources of type $t_v$ on substrate element $v$ for virtual element $u$, $\text{alloc}_{u}(u, v, t_v)$, and the resources of type $t_v$ of $u$ on $v$, $\text{alloc}_{u}(u, v)$. In our mathematical program, we assume a linear relation, which is given by the constant factor $\text{prop}(t_v, t_s)$ (Constraint resource).

**Links:** Mapping links is similar to mapping nodes, and hence, several constraints apply also for links. However, there are also some crucial differences. For instance, a single virtual link can be implemented by multiple flows in the substrate network and both vir-

---

**Table 1: Sets**

| $NE_V$ | Virtual Network Elements |
| $NE_{VN}$ | Virtual Nodes |
| $NE_{VL}$ | Virtual Links |
| $NE_S$ | Substrate Network Elements |
| $NE_{SN}$ | Substrate Nodes |
| $NE_{SL}$ | Substrate Links |
| RT | Resource Types |
| RT$_f$: RT$_f$ < RT | RT Applicable to $f$ |
| VT | Value Types |
| $F_l(u)$ | Flows ((source,sink)-Tuples) |

**Table 2: Constants**

| weight($u$, $v$) | Resource Weight |
| penalty($u$) | Cost of Migration for $u \in NE_{VN}$ |
| old($u$, $v$) | Old Mapping |
| suit($u$, $v$) | Suitable Mapping |
| cap$_S$(v) | Capacity Matrix w.r.t. $t_S$ |
| cap$_S$(v, $w$) | Connection Capacity |
| req($u$, $v$, $t$) | Resource Request (Type $t_v$) |
| prop($t_v$, $t_s$) | Scaling Factor |

**Table 3: Variables**

| alloc$_S$(u, v, $t_v$) | Allocated Resources |
| alloc$_V$(u, v) | Hosted Resources |
| new(u, v) | Mapping Matrix for Elements |
| mig(u) | Migration Selector |
| flows$_S$(f, v, w, $t_v$) | Allocated Resources for Flow |
| flows$_V$(f, v, w) | Hosted Resources for Flow |

**Table 4: Constant Ranges**

| weight($u$, $v$), $\forall u \in NE_V$, $v \in NE_S$ | $[0, 1]$ |
| penalty($u$), $\forall u \in NE_{VN}$ | $\geq 0$ |
| old($u$, $v$), suit($u$, $v$), $\forall u \in NE_V$, $v \in NE_S$ | $\{0, 1\}$ |
| cap$_S$(v), $\forall v \in NE_S$, $t_S \in RT$ | $\geq 0$ |
| cap$_S$(v, $w$), $\forall (v, w) \in NE_S^2$, $t_S \in RT$ | $\geq 0$ |
| req($u$, $v$, $t$), $\forall u \in NE_V$, $v \in NE_S$, $t \in RT$, $s \in VT$ | $\geq 0$ |
| prop($t_v$, $t_s$), $\forall t_v, t_s \in RT$ | $\geq 0$ |

**Table 5: Variable Ranges**

| new($u$, $v$), $\forall u \in NE_V$, $v \in NE_S$ | $\{0, 1\}$ |
| alloc$_V$(u, v), $\forall u \in NE_V$, $v \in NE_S$, $t_V$, $t_S \in RT$ | $\geq 0$ |
| alloc$_V$(u, v), $\forall u \in NE_V$, $v \in NE_S$, $t_V \in RT$ | $\geq 0$ |
| mig($u$), $\forall u \in NE_V$ | $\{0, 1\}$ |
| flows$_S$(f, v, w, $t_v$), $\forall f \in F_l(u)$, $(u, v) \in NE_S^2$, $t_v \in RT$, $\forall u \in NE_{VL}$ | $\geq 0$ |
| flows$_V$(f, v, w), $\forall f \in F_l(u)$, $(u, v) \in NE_S^2$, $t_v \in RT$, $\forall u \in NE_{VL}$ | $\geq 0$ |

**Figure 4:** Overview: Sets, Constants, and Variables.
Our results are summarized in Figure 7 (for the first step) and Figure 8 (for the second step). Clearly, the runtime of the first step dominates the total execution time. Figure 7 shows that virtual networks up to size 100 can be embedded in a few seconds only. For larger virtual networks, the runtime goes up to a couple of minutes. The variance in the runtime is large; this is mainly due to the different number of migrations.

Our results are summarized in Figure 7 (for the first step) and Figure 8 (for the second step). Clearly, the runtime of the first step dominates the total execution time. Figure 7 shows that virtual networks up to size 100 can be embedded in a few seconds only. For larger virtual networks, the runtime goes up to a couple of minutes. The variance in the runtime is large; this is mainly due to the different number of migrations.
links and connectivity existing in the generated Rocketfuel topologies of different node sizes, and to some extent due to the solver dealing with the different problem structures. Repeated runs confirmed this observation.

Similarly, we also studied the inverse problem, where a virtual network is fixed (essentially a tree network with 8 nodes and 7 links) and mapped onto a substrate of different size (Figures 9 and 10). Again, the second step can almost be neglected and lies within seconds even for large networks. The runtime of the first step exhibits a high variance, as discussed above, and is slightly lower on average compared to a fixed substrate scenario.

As already mentioned, being able to reconfigure and migrate VNets can be beneficial, especially if the algorithm (or the objective function) does not know the arriving requests in advance. One experiment motivating the use of migration is given in Figure 11 together with the solution computed by our algorithm (see caption for explanations): A VNet 1 is already embedded in the substrate, when a VNet 2 request arrives. In order to embed VNet 2 with its shared link \{a, b, c, d\}, node B of VNet 1 needs to be migrated (see Figure 11 for the solution). Without migration, the VNet 2 request would have to be rejected.

We conducted several experiments to study the benefit of migration. Clearly, the additional number of VNets that can be embedded in scenarios with migration compared to scenarios where reconfigurations are not possible depends on the topology as well as the underlying resource distribution. Moreover, our simulations prove the intuition right that while typically in the beginning there is a set of VNets that can be embedded without migration, once reconfigurations are necessary, future VNet requests are likely to be satisfiable only with migration as well.

Table 1 summarizes the results of an experiment conducted with VNets of size eight nodes that are mapped onto a substrate with 37 nodes and 53 links (Rocketfuel topology 4755.r1.pop.cch). The table shows that in all settings, although the overall benefits of migration depend on the scenario, our algorithm with both node and link reconfigurations always yielded an optimal number of embedded VNets.

6. REMARKS AND DISCUSSION

For virtual network management, in order to describe or communicate resources as well as mappings in a standardized manner, a Resource Description Language (RDL) is used in our prototype. In order to complement the algorithmic part of this paper, in the following, we present some of the main ideas of our RDL. Subsequently, some implementation issues are discussed.
6.1 Resource Description Language

We identify two main scenarios where a formal resource specification can make sense: (1) communication of VNet requests, and (2) description of VNet state information. The first is used to specify VNet requirements, while the second could be used, e.g., for debugging purposes or to inform about the actual reserved range of resources (if differing from the requested values). Of course, in any RDL, flexibility with respect to the specificity of exchanged information is of prime importance: requirements and state information details must be optional and allow for formulation of ranges or heuristics. Furthermore the desired flexibility should also include the possibility of resource aggregation and pre-allocation for all participating entities. Hence, mappings should allow for arbitrary splitting or aggregation of resources. The RDL must be sufficiently flexible to formulate arbitrary networks and be extensible for future compatibility.

The RDL we use in our prototype implementation pursues an Attribute-Value-Pairs (AVPs) approach, where property objects consisting of attributes and values are used, allowing for extensions of a structured attribute hierarchy, and implicit wildcards by property omission. As already mentioned in the algorithm section, modeling both nodes and links as Network Element (NE) objects featuring pairwise connected Interfaces facilitates formalization of topologies into regular graphs for algorithmic handling: network elements turn into vertices, connections into edges.

The RDL sketched in Figure 12 contains two main topology objects: Network Elements (NEs) and Network Interfaces (NIs). The Connections (i.e., edges in the above mentioned graph) are not modeled as separate objects, as they do not carry additional information by themselves and may be modeled on an object level as NI attributes.

Both NEs and NIs may be attributed with property Feature and Resource objects (Resources may be associated to several NEs/NIs if they are shared). These represent the property AVPs. Resource objects furthermore sport a value type in order to discriminate minimum/maximum/average etc. requirements or guarantees.

Table 1: Use of migration: The first column gives the factor by which the resources in the substrate network elements exceed the requested virtual resources; the second column states the number of VNets that could be embedded without migration; the third column gives the number with link migrations only, the fourth column gives the number for both link and node reconfigurations; and finally, the last column gives the theoretical upper bound on the number of VNets that can be embedded in this setting.

<table>
<thead>
<tr>
<th>Factor</th>
<th>w/o</th>
<th>w/ Link</th>
<th>w/ Link&amp;Node</th>
<th>Opt</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>5</td>
<td>9</td>
<td>9</td>
</tr>
<tr>
<td>3</td>
<td>7</td>
<td>8</td>
<td>13</td>
<td>13</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>1</td>
<td>17</td>
<td>17</td>
</tr>
<tr>
<td>5</td>
<td>17</td>
<td>22</td>
<td>24</td>
<td>24</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>2</td>
<td>27</td>
<td>27</td>
</tr>
<tr>
<td>7</td>
<td>31</td>
<td>32</td>
<td>32</td>
<td>32</td>
</tr>
<tr>
<td>8</td>
<td>37</td>
<td>37</td>
<td>37</td>
<td>37</td>
</tr>
</tbody>
</table>

Figure 11: Top: Substrate network, VNet 1, and VNet 2. Numbers beside nodes and links denote capacities (for substrate elements) and requirements (for virtual elements), respectively. Bottom left: VNet 1 (red) is already embedded, when VNet 2 (yellow) arrives. Bottom right: Situation after embedding VNet 2, which required migrating part of VNet 1.

Figure 12: The resource description language RDL.
The Constraint Group (CG) object allows for the formulation of feature dependencies between NEs. As an example, consider a scenario where the requirements are not given w.r.t. whether servers need to be 32 or 64 bit architectures, but groups of servers need to be binary compatible. The CG object binds groups of prioritized Constraint objects to groups of NEs. The constraints themselves refer to feature attributes and specify applying relations (e.g., equal word-size, or architecture).

The Console Interface objects specify the service to contact in order to gain out-of-VNet access to a resource. It is not mandatory, as not all NEs will sport one (and it may not be required for all NEs that do support console access).

While a Graph Label object identifies any topology description, mapping between topology descriptions can be expressed in two ways: Either by a list of hosting, or hosted NEs carried by each NE (in the case of canonic resource mappings), or by the use of Resource Mapping objects relating resources of one graph to resources of another.

In order to provide the required extensibility, the attributes must follow a canonical extensible structure. We use structures in the nature of MIME or SNMP MIB, creating hierarchies, where increasing depth corresponds to increasing specificity. Feature attributes therefore consist of a path corresponding to the NE or NI type, and a leaf designating the described property (e.g., /node/host/.../wordsize). The resource attribute hierarchy is structured along a classification of resources (e.g. /link/symmetric/bandwidth). In order to allow for arbitrary levels of specificity, every level of the path hierarchy requires a .../generic/ subtree, featuring the union of all leafs of the respective subtrees as leafs.

### 6.2 Remark on Implementation

In our prototype, prior to running the solvers, two preprocessing steps are performed to solve constraint groups, and to refine the topological descriptions.

The constraint groups and constraints are interpreted as a logic formula to which valid assignments are found by substitution. The given VNet description is thereafter refined by modification of the associated feature object set, which will serve as one part of the basis for the aforementioned suitability matrix suit(u, v) upon MIP formulation.

On the PIP level, any missing components may be added to the VNet description. (E.g. in a VNet topology where several links of different properties are directly connected, an active component (e.g., a media adapter, or a traffic shaping device) is required. Since this adapter implicitly limits mapping solutions, the adapter should be included in the MIP. Therefore, an extended VNet request description (according to requirements and policies) is created, onto which the original description’s elements are mapped. This process may be repeated multiple times.

In principle, the same procedure may apply to the substrate description; for instance, several substrate hosts and an NFS server may be modeled as a set of substrate hosts with shared disk storage space, and the respective resources may be mapped to the actual substrate description. However, if the substrate changes only at a low interval, these intermediate descriptions may be candidates for caching in the database.

Subsequently, iterating through the topology descriptions stored in the database, the mapping component creates a mathematical program according to the grammar provided above. Only the most refined versions of both VNet and substrate topology descriptions are considered in this step. The mapping component begins by creating a data structure containing information about prior mappings, requirements, and available resources. A suitability matrix is created based on comparison of the virtual and substrate network elements.

### 7. CONCLUSION

This paper presented an algorithm in the form of a mathematical program that computes resource-efficient embeddings. It allows for different objective functions. The possibility to dynamically re-configure existing mappings is especially useful in scenarios where future VNet requests are not known in advance. We have shown that VNet embeddings are beyond simple multi-commodity flow problems and that many settings can be captured without sacrificing linearity—an important property for automated solutions by standard software tools. Indeed, we showed how to (a) incorporate migration cost, (b) map multiple virtual nodes to the same substrate node and distribute links across multiple path, (c) handle a multitude of different link types both in the substrate and the VNet, and (d) how to incorporate provider policies. We complemented our algorithmic analysis with a discussion of a resource description language that can be used to communicate requests and embeddings in a formal manner.

Both the algorithm as well as the language are fully incorporated in our prototype implementation. Our simulations indicate that the performance of a standard 2-step solution is in the order of seconds for network sizes of around hundred nodes, and we plan to investigate further mechanisms to speed-up the mapping process in the future. Moreover, we look forward to performing tests “in the wild.”

### Acknowledgments

We are grateful to Stefan Heinz from the Optimization Department at TU Berlin for his assistance with the SCIP solver \cite{scip}. Special thanks also go to Ernesto Abarca who was a great help during the prototype implementation. Part of this work was performed within the 4WARD project, which is funded by the European Union in the 7th Framework Programme (FP7), the Virtu project, funded by NTT DOCOMO Euro-Labs, and the Collaborative Networking project, funded by Deutsche Telekom AG. We would like to thank our colleagues in these projects for many fruitful discussions.
8. REFERENCES


Nodes:
\[
\text{map node: } \sum_{v \in N_{E_S}} \text{new}(u, v) = 1 \\
\text{set new: } \text{alloc}_{t_S}(u, v, t_V) \leq \text{cap}_{t_S}(v) \text{new}(u, v) \\
\text{req min: } \text{alloc}_{t_S}(u, v) \leq \text{new}(u, v) \text{req}(u, t_V, s) \\
\text{req max: } \text{alloc}_{t_S}(u, v) \geq \text{new}(u, v) \text{req}(u, t_V, s) \\
\text{req con: } \text{alloc}_{t_S}(u, v) = \text{new}(u, v) \text{req}(u, t_V, s) \\
\text{set new: } \text{alloct}_S(u, v, t_V) \leq \text{capt}_S(v) \text{new}(u, v) \\
\forall u \in N_{E_V}, v \in N_{E_S}, t_V, t_S \in RT, s = \text{minimum} \\
\forall u \in N_{E_V}, v \in N_{E_S}, t_V, t_S \in RT, s = \text{maximum} \\
\forall u \in N_{E_V}, v \in N_{E_S}, t_V, t_S \in RT, s = \text{constant} \\
\]

Mapping:
\[
\text{relate V: } \text{alloc}_{t_v}(u, v) \geq \text{new}(u, v) \\
\forall u \in N_{E_V}, v \in N_{E_S}, t_V \in RT \\
\text{allowed: } \text{alloc}(u, v) \geq \text{new}(u, v) \\
\forall u \in N_{E_V}, v \in N_{E_S} \\
\text{capacity: } \sum_{u \in N_{E_V}, v \in N_{E_S}, t \in RT} \text{alloc}_{t_S}(u, v, t_V) \leq \text{cap}_{t_S}(v) \\
\forall v \in N_{E_S}, t_S \in RT \\
\]

Resource-Variable Relation:

Links:
\[
\text{map link: } \sum_{v \in N_{E_S}} \text{new}(u, v) \geq 1 \\
\forall u \in N_{E_V} \\
\text{map src: } \text{new}(u, v) \geq \text{new}(q_f, v) \\
\forall f \in F(u), v \in N_{E_S}, q_f \text{ sink of } f; \forall u \in N_{E_V} \\
\text{map sink: } \text{new}(u, v) \geq \text{new}(d_f, v) \\
\forall f \in F(u), v \in N_{E_S}, d_f \text{ sink of } f; \forall u \in N_{E_V} \\
\text{req min: } \sum_{v \in N_{E_S}} (\text{flow}_{t_v}(f, v, w) - \text{flow}_{t_v}(f, w, v)) \\
\geq \text{new}(q_f, v) \text{req}(u, t_V, s) - \text{new}(d_f, v) \infty \\
\forall f \in F(u), v \in N_{E_S}, t_V \in RT; \forall u \in N_{E_V}, s = \text{minimum} \\
\text{req max: } \sum_{v \in N_{E_S}} (\text{flow}_{t_v}(f, v, w) - \text{flow}_{t_v}(f, w, v)) \\
\leq \text{new}(q_f, v) \text{req}(u, t_V, s) + \text{new}(d_f, v) \infty \\
\forall f \in F(u), v \in N_{E_S}, t_V \in RT; \forall u \in N_{E_V}, s = \text{maximum} \\
\text{req const: } \sum_{v \in N_{E_S}} (\text{flow}_{t_v}(f, v, w) - \text{flow}_{t_v}(f, w, v)) \\
= \text{new}(q_f, v) \text{req}(u, t_V, s) - \text{new}(d_f, v) \text{req}(u, t_V, s) \\
\forall f \in F(u), v \in N_{E_S}, t_V \in RT; \forall u \in N_{E_V}, s = \text{constant} \\
\]

Link Allocation:
\[
\text{exp out: } \sum_{w \in N_{E_S}} \text{flow}_{t_v}(f, v, w, t_V) \leq \text{alloc}_{t_S}(u, v, t_V) \\
\forall f \in F(u), v \in N_{E_S}, t_V \in RT; t_S \in RT; \forall u \in N_{E_V} \\
\text{exp in: } \sum_{w \in N_{E_S}} \text{flow}_{t_v}(f, v, w, t_V) \leq \text{alloc}_{t_S}(u, v, t_V) \\
\forall f \in F(u), v \in N_{E_S}, t_V \in RT; t_S \in RT; \forall u \in N_{E_V} \\
\text{direction: } \text{flow}_{t_v}(f, v, w, t_V) \leq \text{new}(u, v) \text{cap}_{t_S}(v, w) \\
\forall f \in F(u), (v, w) \in N_{E_S}^2, t_V \in RT; t_S \in RT; \forall u \in N_{E_V} \\
\]

Migration:
\[
\text{new: } \sum_{v \in N_{E_S}} \text{old}(u, v) \geq \text{mig}(u) \\
\forall u \in N_{E_V} \\
\text{migrated: } \text{old}(u, v) - \text{new}(u, v) \leq \text{mig}(u) \\
\forall u \in N_{E_V}, v \in N_{E_S} \\
\]

Figure 6: Embedding constraints for linear Mixed Integer Program. Explanations are given in the text.