Optimizing Long-Lived CloudNets with Migrations

Gregor Schaffrath, Stefan Schmid, Anja Feldmann
Telekom Innovation Labs & TU Berlin, Berlin, Germany

***Abstract*—This paper attends to the problem of embedding flexibly specified virtual networks connecting cloud resources (e.g, storage or computation) on a given substrate (e.g., a data center, an ISP backbone, a router site, or a virtual provider network). We study a scenario where a substrate provider (or a potential intermediate broker or reseller) wants to optimize the embedding of these so-called CloudNets by migrating them to more suitable locations. For instance, such re-embeddings can be useful if the CloudNets were requested at short notice and initially placed heuristically. Subsequent optimizations can, e.g., reduce the peak resource loads in the network by spreading CloudNets across the infrastructure or save energy by moving CloudNets together and switching off unused components.**

We present the generic mathematical programming algorithm used in our CloudNet prototype to compute optimal embeddings. For example, this algorithm supports different objective functions (such as load minimization or energy conservation), arbitrary resource combinations and the mapping of multiple virtual nodes of a CloudNet to a single substrate node, cost-aware migrations, and it can deal with all link types that arise in practice (e.g., full-duplex or even wireless or wired broadcast links with multiple endpoints). Of course, such rigorous CloudNet optimizations are time consuming, and we report on the time complexities obtained from our experiments with our network virtualization prototype architecture. It turns out that optimizing CloudNets over moderate sized infrastructures is feasible, even for scenarios with high flexibility and without tuning the solver software to speed up computations further.

I. Introduction

Virtualization is arguably the most important design principle to facilitate innovation in the Internet. While node virtualization enabled a very flexible resource allocation in data centers and clouds, the virtualization trend now seems to spill over also to Internet Service Providers (ISPs). The decoupling of services from the constraints of the underlying infrastructure promises new forms of resource sharing in the ISP network. Moreover, the possibility to deploy (and migrate) services closer to the eyeballs of the users can significantly increase productivity and revenues. These new forms of networking do not only rely on node virtualization, but also on *link virtualization*. For example, Google recently announced to have adopted a *Software-Defined Networking (SDN)* approach to manage its massive internal network using OpenFlow.

The technological advances allows us to realize the vision of CloudNets [9] which provide an abstraction of *both nodes and links*, connecting (and providing access to) virtual cloud resources with virtual networking. Decoupling virtual networks from the physical constraints of the underlying infrastructure (the *substrate*), CloudNets can offer opportunities for customized network environments and can be flexibly *embedded* (or *mapped*) at optimal (e.g., economical) locations and even migrated.

The main challenge for the realization of CloudNets is the question of how to optimally exploit the flexibility of the specification in order to embed the virtual CloudNet in the best location in the substrate network (while *guaranteeing* all resource / performance requirements stated in the specification), where "best" depends on the objective: in some scenarios, a CloudNet should be embedded in such a manner that the maximal load or congestion is minimized; in other scenarios, the CloudNets should be embedded in a compact manner in order to be able to shut down the other parts of the physical network, e.g., to save energy.

In addition to the static embedding, (parts of) a CloudNet can be migrated. As CloudNet requests arrive over time and may be hard to predict, certain embeddings computed online in the past may become suboptimal, and re-embeddings and migrations are necessary. Another reason for migration is network or server maintenance: in order to upgrade the infrastructure, it can be useful to temporarily migrate (parts of) the network or the servers to an alternative location. While today's network virtualization technology facilitates seamless migration (without session interruption), migration has a cost (e.g., in terms of computation, bandwidth, or even roaming fees in case of cross-provider migrations). Whether and where to migrate is hence a non-trivial problem.

Our Contribution. This paper addresses the problem of CloudNet embedding optimization by migration. We present the very generic embedding algorithm used in our CloudNet prototype architecture [18] which *jointly optimizes node placements and link embeddings* and exploits this flexibility to compute *optimal embeddings* ("realizations") of the CloudNets, while *guaranteeing* the allocation and provisioning of the requested combination of resources. Our algorithm integrates *cost-aware migrations* [1], and we believe that as the type and arrival time of CloudNet requests is hard to predict, the possibility of reconfigurations and migration is crucial. Moreover, our algorithm does not rely on any particular substrate type [22].

In addition, in contrast to other virtual network embedding algorithms, our approach provides a high flexibility: it supports all link types that occur in practice, such as half-duplex, full-duplex, or even broadcast links with multiple endpoints (as they appear in wireless networks but also in wired contexts with hubs), in the sense that any of these links can be mapped on any other; it supports embeddings across resources and resource types and exact solutions for the mapping of partial networks to single substrate nodes; it further supports provider-side placement policies as well as resource prioritization (e.g., prioritizing lucrative resource allocations); it also allows us to take into account node-based loads, e.g., as a function of the packet rate (shorter packets increase the computational load at the forwarding engine) etc.; among many more.

Finally, our algorithm can be used to study the migration cost-benefit tradeoff: By computing the embedding that would result from migration together with the migration cost, it is left to the (potentially automated) administrator to decide whether the changes are worthwhile. For example, our algorithm allows to answer questions of the form: Can we migrate CloudNets to a more compact form such that 20% of the currently used resources are freed up, and what would be the corresponding migration cost?

Interestingly, despite this flexibility, our algorithm is a (linear) Mixed Integer Program (MIP) and can hence be solved by standard and optimized tools such as `CPLEX`. Another advantage of the mathematical programming approach is that it enables us to propose different objective functions which can be easily exchanged. For example, at some point a provider may choose to place the virtual networks on the "edge" of the physical network in order to avoid blocking bottleneck links and hence to maximize the likelihood that future CloudNet requests can be accepted. At another point in time it wants to spread the CloudNet embeddings as much as possible in order to minimize the load or congestion, or to collocate the CloudNets as much as possible in order to be able to switch off other parts of the network in order to save energy or for maintenance work.

In contrast to various existing embedding heuristics for virtual networks (e.g., [8], [12], [13]), the focus of this paper is on an *integrated* approach which emphasizes the *quality* of the CloudNet embeddings. In particular, we focus on the *optimal solutions*, and investigate the feasibility of this approach to improve initially heuristically placed CloudNets; this is interesting for *long-lived* CloudNets where the resource investments for computing the optimizations may pay off in the future. For example, we envision a scenario where a provider reserves a fraction of the resources of the components for such initial, fast placements. If the durations of the CloudNets are heavy-tailed, our algorithm should optimize the oldest CloudNets first.

We have three use-cases in mind: (1) a *VPN-like scenario* (**VPN**) where the virtual node locations are given, (2) a *data center like scenario* (**DC**) where the virtual node placements are fully flexible, and (3) an *out-sourcing / (spillover to) cloud scenario* (**OC**) where some virtual nodes have a fixed location (location and access network of a company) and others do not (out-sourced services, e.g., for cloud bursting). Clearly, the data center scenario is the most challenging for optimization, while the VPN scenario, without the possibility to optimize terminal mappings, boils down to a classic flow problem which can even be solved optimally in polynomial time.

In our evaluation, we will focus on the out-sourcing scenario, which—depending on the flexibility parameter used—can emulate the other two scenarios. (We refer the reader to [16] for more scenarios.) Our results confirm that even without tuning the algorithm or solver parameters to speed up computations, optimal CloudNet embeddings are feasible on moderate size substrates, e.g., a router site, or a small physical or virtual provider network.

II. Basic Concepts

The main objective of our algorithm is to (re-)embed CloudNet requests, consisting of virtual links connecting virtual nodes, by mapping them onto the given substrate network resources in such a way that the specification is fulfilled (i.e., all specified node and link resources are allocated to the CloudNet); or to reject the request otherwise. (See, e.g., [5] for an introduction to the general virtual network embedding problem.) For instance, a virtual node may require a 1GHz CPU and may only be mapped onto Linux nodes in the US; however, it does not come with any more specific constraints on which physical node it is embedded. Similarly, CloudNet links may need a minimum of 10 MBit/s, and may be realized as a set of $k \geq 1$ flows connecting the locations where the the endpoints of the virtual link are mapped (if the endpoints are mapped to a single physical node, the virtual link may be realized at almost no cost). The resources offered by the substrate nodes and links can be shared among the virtual networks.

The algorithm should be flexible in terms of the objective function to be optimized for the CloudNet placement, and the arrival of a new (or re-embedding of an old) CloudNet should cause minimal changes to the existing embedding of prior CloudNets (i.e., since CloudNet migration cost is non-zero there is a tradeoff between migration cost and a superior embedding).

In the following, we will introduce the main ideas of our embedding algorithm. We pursue a mathematical programming approach and present a (linear) *Mixed Integer Program (MIP)* which has the advantage that standard software tools such as `CPLEX` or `lpsolve` can be used to perform the computations. (Although we focus on optimal embeddings, these tools also offer different heuristics for faster but approximate solutions; all these heuristics are directly applicable to our program as well.)

A MIP consists of a (linear) *objective function* expressed using a set of variables, plus a set of (linear) constraints on these variables that ensure "valid" solutions. If a problem can be specified in this form (what we do in this paper for the CloudNet embedding problem), state-of-the-art optimized algorithms can be used for evaluation. This section serves to introduce the reader to our approach and the different variables and constants used in our program. The complete formal program description appears in Figure 1.

Graph Representation. *Shared communication channels*, i.e., links with several end points (both in the virtual and the substrate network) constitute a first challenge for such a generic embedding algorithm. 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 notion of *network elements* (NEs): network elements represent both *nodes* (set NE_{N}) and *links* (set NE_{L}). Network elements are connected by interfaces, which form the edges of the graph.

We distinguish between virtual network elements of the CloudNet (set $NE_{\mathrm{V}} = NE_{\mathrm{VN}} \cup NE_{\mathrm{VL}}$ of virtual nodes and virtual links) and substrate network elements of the substrate network (set $NE_{\mathrm{S}} = NE_{\mathrm{SN}} \cup NE_{\mathrm{SL}}$). In principle, any virtual node can be mapped onto any substrate node, depending on the requirements. A virtual link can be embedded onto a substrate node, a substrate link, or onto a set of paths in the substrate network (resulting in a multi-flow embedding).

The 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 links with several endpoints, we replace each link with a vertex and add graph edges accordingly.

Placement Policies and Suitability. We use the binary matrix $new(u, v)$ to denote whether a virtual CloudNet element $u \in NE_{\mathrm{V}}$ is mapped to a substrate network element $v \in NE_{\mathrm{S}}$ ($new(u, v) = 1$) 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 $alloc_{r_{\mathrm{V}}}(u, v)$ which captures the amount of virtual resource r_{V} of u hosted on v and $alloc_{r_{\mathrm{S}}}(u, v, r_{\mathrm{V}})$ describing the substrate resources r_{S} used to allocate it. The resources r_{V} requested for u are represented by the constant matrix $req(u, r_{\mathrm{V}}, s)$, where $s \in VT$ refers to the value type of request (e.g., minimum, maximum, ...). To ensure that the sum of the allocated resources never exceeds substrate capacities of substrate we use the constant capacity matrices $cap_{r_{\mathrm{S}}}(v)$, $cap_{r_{\mathrm{S}}}(v, w)$, and $cap(r_{\mathrm{S}})$. The first two hold individual capacities of substrate components v and substrate interfaces interconnecting v and w with respect to r_{S}. The last represents the capacity of a resource r_{S} itself. All three are required to correctly model various possible shared resources assignments in the substrate.

It is not always possible to map a virtual network element to any arbitrary substrate element. For example, a virtual cloud node may be restricted to 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 CloudNet according to internal placement policies or cost factors. We thus use a constant weight matrix $weight(u, v)$ to introduce a cost for each node placement. These weights can also be used as policy support to prioritize certain resource allocations over others in the objective function.

Link Types and Resources. 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 (wireless, or non-switched, hub-like) channel is referred as *half-duplex* link. Note that half-duplex links are symmetric by nature.

We explicitly distinguish between two classes of resources $R = R_{\mathrm{V}} \cup R_{\mathrm{S}}$, namely virtual resources $r_{\mathrm{V}} \in R_{\mathrm{V}}$ and substrate resources $r_{\mathrm{S}} \in R_{\mathrm{S}}$. To handle the different link types, virtual half-duplex links are associated to an r_{V} of attribute `'/link/symmetric/bandwidth'` whereas substrate full-duplex links receive two r_{S} with `'/link/upstream/bandwidth'` and `'/link/downstream/bandwidth'` respectively. In our embedding program, we assume a proportional relationship between r_{V} and r_{S}, that is, we consider a proportional factor $prop(r_{\mathrm{V}}, r_{\mathrm{S}})$. As different relation functions are possible (e.g., involving constant instantiation overhead), the respective constraints should be considered exemplary.

Interestingly, differentiating between r_{V} and r_{S} in both CloudNet specification and MIP is not only useful for handling different link types but also for mapping nodes: It enables us to map and even split resources of arbitrary resource types onto arbitrary other resource types.

To handle a shared communication channel we decompose its multiple endpoints into a *set of flows*. In particular, for each link u, we introduce a set $Fl(u)$ that describes the set of possible source-sink pairs for u.

Each flow $f \in Fl(u)$ inherits the requirements of u. Analogously to the $alloc$ matrices, $flow_{r_{\mathrm{V}}}(f, v, w)$ and $flow_{r_{\mathrm{S}}}(f, v, w, r_{\mathrm{V}})$ reflect tentative resource allocations on substrate interfaces, and $new(f, v)$ denotes corresponding tentative flow mappings. Resources of these flows $f \in Fl(u)$ form the set $R_f \subset R_{\mathrm{V}}$.

The Flow Problem. While we consider virtual nodes atomic in the context of our MIP, virtual links can be realized either as single path or multiple paths within the substrate network. The aggregated resources of the paths must satisfy the requirements of the virtual link while not exceeding the capacity limits of the substrate elements. For instance, the sum of the bandwidths of 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 endpoints are not fixed, but candidate locations overlap.

Our mathematical program ensures that the allocated flows are connected, consistent with the requirements and capacities. 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).

Migration Support. Support for network migration is one of the key features of our algorithm. For example, migrations can be useful to re-embed a CloudNet due to a new demand (e.g., due to user mobility), because other CloudNets expire, because physical resources may change prices over time or maintenance is needed, or just to optimize long-lived CloudNets which were initially quickly placed to satisfy the user request fast.

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
R_{V}	Set of Virtual Resource
R_{S}	Set of Substrate Resource
$R_f : R_f \subset R_{\mathrm{V}}$	Set of Virtual Flow Resources
VT	Value Types
$Fl(u)$	Flows ((source,sink)-Tuples)

TABLE I
SET DEFINITIONS

Migration costs depend on many factors. For instance, classic server migrations may entail service interruption costs that depend on the available bandwidth along the migration path, while live migration technology may provide a seamless service and only come at a cost of bulk data transfers (or even at no cost if it is constrained to a fully switched rack). There can also be costs of removing resources, and a migration can entail a management overhead C_{mgmt}. If, e.g., a virtual network provider triggers a cross-provider migration, the termination of provisioning contracts may entail penalties C_{contract}. Temporary redundant allocations of resources and reconfiguration based service outages during migration entail opportunistic costs C_{reconfig}. An example cause for the latter would be outages triggered by switchovers to and from transitional provisioning solutions. The resource transferral cost may depend on the migration destination and relate to the actual transfer and possible property changes. Bulk data transfers may entail both real and opportunistic transit costs C_{transit}. E.g., transfer of host state may require additional bandwidth to be leased from transit providers. Furthermore, adaptations may impose overhead if crucial properties change (e.g., migrating a virtual host from Xen to KVM). We denote this cost factor by $C_{\mathrm{adaptation}}$. And so on.

It is not always necessary to migrate entire networks, and sometimes, already small local reconfigurations may reduce the overall resource overhead and improve the embedding substantially.

To this end, we introduce matrices and constraints that allow the specification of reconfiguration costs and enable the solver to weight them against the respective benefits. Analogue to $new(u, v)$, we use the constant binary matrix $old(u, v)$ to describe existing mappings, and specify whether a virtual network element u is currently mapped to a substrate element v ($old(u, v) = 1$) or not ($old(u, v) = 0$).

We account for the cost of migration in two respects: destination independent cost factors are reflected in the constant penalty matrix $penalty(u) = C_{\mathrm{contract}}(u) + C_{\mathrm{mgmt}}(u) + C_{\mathrm{reconfig}}(u)$. Destination dependent C_{transit} and $C_{\mathrm{adaptation}}$ cost factors to migrate virtual network element u from its current position to substrate element v are summed up in the constant matrix $transit(u, v)$.

Node migration is typically more expensive relative to link migration ($penalty(u) = \epsilon$ for $\forall u \in NE_{\mathrm{VL}}$ and an arbitrarily small $\epsilon > 0$), as links do not involve state or bulk data transfers but are rather re-instantiated. We will reflect costs by the link's $transit(u, v)$ variables.

III. THE EMBEDDING ALGORITHM

Based on the above ideas we next describe the MIP in details. While we introduced most sets, variables, and constants above we will describe the remaining ones in support of specific objective functions, and proceed with an explanation of the constraints.

How is an optimal embedding of a CloudNet on a set of resources in the substrate network characterized? 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_0 may be suboptimal upon the arrival of the next request at some time $t > t_0$.

We hence do not propose any specific objective functions here (recall that one advantage of our mathematical programming approach is the ease of exchanging different objective functions with only limited implementation effort) but just consider a canonical example: minimizing resource usage to ensure localizing embeddings. The minimization of the amount of substrate resources used for CloudNet allocations is a natural objective that maximizes the chances to be able to embed also future requests, to save energy by switching off unused hardware, or to perform maintenance work. The objective function also used for our experiments hence balances resource usage and migration cost: $\sum_{u \in NE_{\mathrm{V}}} \sum_{v \in NE_{\mathrm{S}}} \sum_{r_{\mathrm{S}} \in R_{\mathrm{S}}} weight(u, v) \cdot alloc_{r_{\mathrm{S}}}(u, v, r_{\mathrm{V}}) + \sum_{u \in NE_{\mathrm{V}}} (penalty(u) \cdot mig(u) + \sum_{v \in NE_{\mathrm{S}}} transit(u, v) \cdot new(u, v))$.

Alternatively, in our prototype, we also employ an objective function that seeks to distribute the load equally among all network elements to minimize peak loads and congestion. Please refer to the technical report for more details.

The embedding must fulfill various type, capacity, and other consistency constraints, see Figure 1 for a complete and formal constraint list.

Nodes: This constraints category is used to ensure that each CloudNet node is mapped to an appropriate substrate node. In contrast to links, we do not map nodes 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_con`).

Mapping: The mapping constraints ensure that the substrate element has sufficient capacity (Constraint `ne_capacity`) allocated. If resources are shared amongst substrate elements, we need to check against the capacity of the resource itself (Constraint `capacity`). In order to limit link splitups, we set a minimal resource allocation unit (Constraint `relate_V`). Moreover, virtual elements hosted must be of the correct type (Constraint `allowed`). Constraint `load` and Constraint `max_load` define the load of a resource (i.e. the fraction of

Constants		Range	
$weight(u,v)$	Resource Weight	$\forall u \in NE_{\mathrm{V}}, v \in NE_{\mathrm{S}}$	$\in [0,1]$
$penalty(u)$	Migration Cost	$\forall u \in NE_{\mathrm{V}}$	> 0
$transit(u,v)$	Costs transferring u resources to v	$transit(u,v), \forall u \in NE_{\mathrm{V}}, v \in NE_{\mathrm{S}}$	≥ 0
$old(u,v)$	Old Mapping	$\forall u \in NE_{\mathrm{V}}, v \in NE_{\mathrm{S}}$	$\in \{0,1\}$
$suit(u,v)$	Suitable Mapping	$\forall u \in NE_{\mathrm{V}}, v \in NE_{\mathrm{S}}$	$\in \{0,1\}$
$cap_{r_{\mathrm{S}}}(v)$	Capacity of v w.r.t. r_{S}	$\forall v \in NE_S, r_S \in R_S$	≥ 0
$cap_{r_{\mathrm{S}}}(v,w)$	Connection Capacity	$\forall (v,w) \in NE_{\mathrm{S}}^2, r_{\mathrm{S}} \in R_{\mathrm{S}}$	≥ 0
$cap(r_{\mathrm{S}})$	Resource r_{S} Capacity	$\forall r_{\mathrm{S}} \in R_{\mathrm{S}}$	≥ 0
$req(u, r_{\mathrm{V}}, s)$	Resource Request	$\forall u \in NE_{\mathrm{V}}, r_{\mathrm{V}} \in R_{\mathrm{V}}, s \in VT$	≥ 0
$prop(r_{\mathrm{V}}, r_{\mathrm{S}})$	Scaling Factor	$\forall r_{\mathrm{V}} \in R_{\mathrm{V}}, r_{\mathrm{S}} \in R_{\mathrm{S}}$	≥ 0
$weight_{r_{\mathrm{S}}}$	Load Weight Factor	$\forall r_{\mathrm{S}} \in R_{\mathrm{S}}$	$\in [0,1]$
c	`sum`,`max` Load Priority Factor		$\geq \sum_{r_{\mathrm{S}} \in R_{\mathrm{S}}} weight_{r_{\mathrm{S}}}$
$min\_alloc_{r_{\mathrm{V}}}$	Min. r_{V} allocation unit	$\forall r_{\mathrm{V}} \in R_{\mathrm{V}}$	≥ 0

Variables		Range	
$alloc_{r_{\mathrm{S}}}(u,v,r_{\mathrm{V}})$	Allocated Resources	$\forall u \in NE_{\mathrm{V}}, v \in NE_{\mathrm{S}}, \forall r_{\mathrm{V}} \in R_v, r_{\mathrm{S}} \in R_{\mathrm{S}}$	≥ 0
$alloc_{r_{\mathrm{V}}}(u,v)$	Hosted Resources	$\forall u \in NE_{\mathrm{V}}, v \in NE_{\mathrm{S}}, \forall r_{\mathrm{V}} \in R_{\mathrm{V}}$	≥ 0
$new(u,v)$	Mapping Matrix for Elements	$\forall u \in NE_{\mathrm{V}}, v \in NE_{\mathrm{S}}$	$\in \{0,1\}$
$new(f,v)$	Mapping Matrix for Flows	$\forall f \in Fl(u), v \in NE_{\mathrm{S}}, \forall u \in NE_{\mathrm{VL}}$	$\in \{0,1\}$
$mig(u)$	Migration Selector	$\forall u \in NE_{\mathrm{V}}$	$\in \{0,1\}$
$flow_{r_{\mathrm{S}}}(f,v,w,r_{\mathrm{V}})$	Allocated Resources for Flow	$\forall (v,w) \in NE_{\mathrm{S}}^2,$ $\forall f \in Fl(u), r_{\mathrm{V}} \in R_{\mathrm{V}}, r_{\mathrm{S}} \in R_{\mathrm{S}}, \forall u \in NE_{\mathrm{VL}}$	≥ 0
$flow_{r_{\mathrm{V}}}(f,v,w)$	Hosted Resources for Flow	$\forall (v,w) \in NE_{\mathrm{S}}^2,$ $\forall f \in Fl(u), r_{\mathrm{V}} \in R_{\mathrm{V}}, u \in NE_{\mathrm{VL}}$	≥ 0
$load(r_{\mathrm{S}})$	Load on Resource r_{S}	$\forall r_{\mathrm{S}} \in R_{\mathrm{S}}$	≥ 0
max_load	Max Load over All r_{S}		≥ 0

TABLE II
CONSTANTS AND VARIABLES

Nodes:

$$\begin{array}{lll}
\texttt{map\_node:} & \sum_{v \in NE_{\mathrm{S}}} new(u,v) = 1 & \forall u \in NE_{\mathrm{VN}} \\
\texttt{set\_new:} & alloc_{r_{\mathrm{S}}}(u,v,r_{\mathrm{V}}) \leq cap_{r_{\mathrm{S}}}(v) new(u,v) & \forall u \in NE_{\mathrm{VN}}, v \in NE_{\mathrm{S}}, r_{\mathrm{V}} \in R_{\mathrm{V}}, r_{\mathrm{S}} \in R_{\mathrm{S}} \\
\texttt{req\_min:} & alloc_{r_{\mathrm{V}}}(u,v) \geq new(u,v) req(u,r_{\mathrm{V}},s) & \forall u \in NE_{\mathrm{VN}}, r_{\mathrm{V}} \in R_{\mathrm{V}}, r_{\mathrm{S}} \in R_{\mathrm{S}}, s = \texttt{minimum} \\
\texttt{req\_max:} & alloc_{r_{\mathrm{V}}}(u,v) \leq new(u,v) req(u,r_{\mathrm{V}},s) & \forall u \in NE_{\mathrm{VN}}, r_{\mathrm{V}} \in R_{\mathrm{V}}, r_{\mathrm{S}} \in R_{\mathrm{S}}, s = \texttt{maximum} \\
\texttt{req\_con:} & alloc_{r_{\mathrm{V}}}(u,v) = new(u,v) req(u,r_{\mathrm{V}},s) & \forall u \in NE_{\mathrm{VN}}, r_{\mathrm{V}} \in R_{\mathrm{V}}, r_{\mathrm{S}} \in R_{\mathrm{S}}, s = \texttt{constant}
\end{array}$$

Mapping:

$$\begin{array}{lll}
\texttt{relate\_V:} & alloc_{r_{\mathrm{V}}}(u,v) \geq min\_alloc_{r_{\mathrm{V}}} \cdot new(u,v) & \forall u \in NE_{\mathrm{V}}, v \in NE_{\mathrm{S}}, r_{\mathrm{V}} \in R_{\mathrm{V}} \\
\texttt{allowed:} & suit(u,v) \geq new(u,v) & \forall u \in NE_{\mathrm{V}}, v \in NE_{\mathrm{S}} \\
\texttt{ne\_capacity:} & \sum_{u \in NE_{\mathrm{V}}} \sum_{r_{\mathrm{V}} \in R_{\mathrm{V}}} alloc_{r_{\mathrm{S}}}(u,v,r_{\mathrm{V}}) \leq cap_{r_{\mathrm{S}}}(v) & \forall v \in NE_{\mathrm{S}}, r_{\mathrm{S}} \in R_{\mathrm{S}} \\
\texttt{capacity:} & \sum_{v \in NE_{\mathrm{S}}} \sum_{u \in NE_{\mathrm{V}}} \sum_{r_{\mathrm{V}} \in R_{\mathrm{V}}} alloc_{r_{\mathrm{S}}}(u,v,r_{\mathrm{V}}) \leq cap(r_{\mathrm{S}}) & \forall r_{\mathrm{S}} \in R_{\mathrm{S}} \\
\texttt{load:} & weight_{r_{\mathrm{S}}} / cap(r_{\mathrm{S}}) \cdot & \forall r_{\mathrm{S}} \in R_{\mathrm{S}} \\
 & \sum_{v \in NE_{\mathrm{S}}} \sum_{u \in NE_{\mathrm{V}}} \sum_{r_{\mathrm{V}} \in R_{\mathrm{V}}} alloc_{r_{\mathrm{S}}}(u,v,r_{\mathrm{V}}) \leq load(r_{\mathrm{S}}) & \\
\texttt{max\_load:} & load(r_{\mathrm{S}}) \leq max\_load & \forall r_{\mathrm{S}} \in R_{\mathrm{S}}
\end{array}$$

Resource-Variable Relation:

$$\begin{array}{lll}
\texttt{resource:} & \sum_{r_{\mathrm{S}} \in R_{\mathrm{S}}} prop(r_{\mathrm{V}}, r_{\mathrm{S}}) alloc_{r_{\mathrm{S}}}(u,v,r_{\mathrm{V}}) = alloc_{r_{\mathrm{V}}}(u,v) & \forall u \in NE_{\mathrm{V}}, v \in NE_{\mathrm{S}}, r_{\mathrm{V}} \in R_{\mathrm{V}} \\
\texttt{flow\_res:} & \sum_{r_{\mathrm{S}} \in R_{\mathrm{S}}} prop(r_{\mathrm{V}}, r_{\mathrm{S}}) flow_{r_{\mathrm{S}}}(f,v,w,r_{\mathrm{V}}) = flow_{r_{\mathrm{V}}}(f,v,w) & \forall f \in Fl(u), (v,w) \in NE_{\mathrm{S}}^2, r_{\mathrm{V}} \in R_f, \\
 & & \forall u \in NE_{\mathrm{VL}}
\end{array}$$

Links:

$$\begin{array}{lll}
\texttt{map\_link:} & \sum_{v \in NE_{\mathrm{S}}} new(u,v) \geq 1 & \forall u \in NE_{\mathrm{VL}} \\
\texttt{map\_src:} & new(u,v) \geq new(q_f, v) & \forall f \in Fl(u), v \in NE_{\mathrm{S}}, q_f \text{ source of } f; \forall u \in NE_{\mathrm{VL}} \\
\texttt{map\_sink:} & new(u,v) \geq new(d_f, v) & \forall f \in Fl(u), v \in NE_{\mathrm{S}}, d_f \text{ sink of } f; \forall u \in NE_{\mathrm{VL}}
\end{array}$$

$$\begin{array}{lll}
\texttt{req\_fmin:} & \sum_{w \in NE_{\mathrm{S}}} (flow_{r_{\mathrm{V}}}(f,v,w) - flow_{r_{\mathrm{V}}}(f,w,v)) & \geq new(q_f,v) req(u,r_{\mathrm{V}},s) - new(d_f,v)\infty \\
 & \forall f \in Fl(u), v \in NE_{\mathrm{S}}, r_{\mathrm{V}} \in R_f; \forall u \in NE_{\mathrm{VL}}, s = \texttt{minimum} & \\
\texttt{req\_fmax:} & \sum_{w \in NE_{\mathrm{S}}} (flow_{r_{\mathrm{V}}}(f,v,w) - flow_{r_{\mathrm{V}}}(f,w,v)) & \leq new(q_f,v) req(u,r_{\mathrm{V}},s) + new(d_f,v)\infty \\
 & \forall f \in Fl(u), v \in NE_{\mathrm{S}}, r_{\mathrm{V}} \in R_f; \forall u \in NE_{\mathrm{VL}}, s = \texttt{maximum} & \\
\texttt{req\_fconst:} & \sum_{w \in NE_{\mathrm{S}}} (flow_{r_{\mathrm{V}}}(f,v,w) - flow_{r_{\mathrm{V}}}(f,w,v)) & = new(q_f,v) req(u,r_{\mathrm{V}},s) - new(d_f,v) req(u,r_{\mathrm{V}},s) \\
 & \forall f \in Fl(u), v \in NE_{\mathrm{S}}, r_{\mathrm{V}} \in R_f; \forall u \in NE_{\mathrm{VL}}, s = \texttt{constant} &
\end{array}$$

Link Allocation:

$$\begin{array}{lll}
\texttt{exp\_out:} & \sum_{w \in NE_{\mathrm{S}}} flow_{r_{\mathrm{S}}}(f,v,w,r_{\mathrm{V}}) \leq alloc_{r_{\mathrm{S}}}(u,v,r_{\mathrm{V}}) & \forall f \in Fl(u), v \in NE_{\mathrm{S}}, r_{\mathrm{V}} \in R_f, \\
 & & r_{\mathrm{S}} \in R_{\mathrm{S}}, \forall u \in NE_{\mathrm{VL}} \\
\texttt{exp\_in:} & \sum_{w \in NE_{\mathrm{S}}} flow_{r_{\mathrm{S}}}(f,w,v,r_{\mathrm{V}}) \leq alloc_{r_{\mathrm{S}}}(u,v,r_{\mathrm{V}}) & \forall f \in Fl(u), v \in NE_{\mathrm{S}}, r_{\mathrm{V}} \in R_f, \\
 & & r_{\mathrm{S}} \in R_{\mathrm{S}}, \forall u \in NE_{\mathrm{VL}} \\
\texttt{direction:} & flow_{r_{\mathrm{S}}}(f,v,w,r_{\mathrm{V}}) \leq new(u,v) cap_{r_{\mathrm{S}}}(v,w) & \forall f \in Fl(u), (v,w) \in NE_{\mathrm{S}}^2, \\
 & & r_{\mathrm{V}} \in R_f, r_{\mathrm{S}} \in R_{\mathrm{S}}, \forall u \in NE_{\mathrm{VL}} \\
\texttt{relate\_f:} & \sum_{w \in NE_{\mathrm{S}}} flow_{r_{\mathrm{S}}}(f,v,w,r_{\mathrm{V}}) + flow_{r_{\mathrm{S}}}(f,w,v,r_{\mathrm{V}}) \geq new(u,v) & \forall f \in Fl(u), \forall u \in NE_{\mathrm{VL}}, \\
 & & v \in NE_{\mathrm{S}}, r_{\mathrm{V}} \in R_f, r_{\mathrm{S}} \in R_{\mathrm{S}}
\end{array}$$

Migration:

$$\begin{array}{lll}
\texttt{new:} & \sum_{v \in NE_{\mathrm{S}}} old(u,v) \geq mig(u) & \forall u \in NE_{\mathrm{V}} \\
\texttt{migrated:} & old(u,v) - new(u,v) \leq mig(u) & \forall u \in NE_{\mathrm{V}}, v \in NE_{\mathrm{S}}
\end{array}$$

Fig. 1. Embedding constraints for linear Mixed Integer Program. Explanations are given in the text.

its capacity used) and the maximum of all individual resourc loads, respectively.

Resource-Variable Relation: This set of constraints deals with the relation between the resource types r_S that host resources of type r_V. In our mathematical program, we assume a linear relation, which is given by the constant factor $prop(r_V, r_S)$ (Constraints `resource` and `flow_res`).

Links: Mapping links is similar to mapping nodes, and hence, several constraints apply also to links. However, in contrast to nodes, links may be mapped to more than one substrate element (as one or several paths). Shared communication channels need to allocate resources to satisfy their requirements with respect to every virtual node pair connected. In order to calculate allocations in this case, links are expanded into a set of flows, as described earlier. Clearly, each virtual link must be mapped to at least one substrate element (Constraint `map_link`). Sources and sinks of the expanded flows definitely are part of this mapping (Constraints `map_src` and `map_sink`). Note that this allows to find a valid mapping even for pure local links, i.e. if all virtual nodes are mapped to a single substrate node. As a simplification, we assume that pure local links require only nominal resources, considering only resource allocation corresponding to $min\_alloc_{r_V}$[1].

The multi-path propagation of each flow f must satisfy flow preservation, except for the source and sink element. The constraint depends on the value type (`minimum`, `maximum`, or `constant`): In case of a minimum type, the net flow of a given resource type must be a least the requested resources at the source and preserved otherwise. If the substrate element is the sink, the flow preservation invariant is suspended and the constraint becomes fulfilled trivially. To implement a corresponding selector, multiplication by a sufficiently large number (e.g., slightly larger than the maximal amount of involved resources, here simply represented by ∞) is used in the subtrahend. This yields the desired tautology (see Constraint `req_fmin`). The Constraints `req_fmax` and `req_fconst` are defined analogously. Note however that it is not possible to mathematically strictly ensure maximum or constant bandwidth in combination with half-duplex links.

Link Allocation: The r_S allocated for a virtual u on a substrate element v is the maximum of the r_S required for every single of u's flows. Constraints `exp_out` and `exp_in` ensure that these resources are allocated on sources and destinations of the respective flows. Constraint `direction` enforces direction specific capacity constraints on full-suplex substrate resources.

Migration: Our program allows us to migrate already embedded CloudNet elements to new locations, if the reconfiguration costs are amortized by the more efficient embedding. The migration constraints set the migration flag $mig(u)$ if[2] the mapped element is not new (Constraint `new`) and was previously embedded at a different location, where it was removed (Constraint `migrated`).

[1]This can be extended trivially by adding a variant of constraints req_* for links u, where $new(u, v)$ is replaced by $new(u', v)$ for all virtual nodes u' connected to u

[2]and only if, whenever migration costs are relevant - i.e., > 0, and minimized in the objective function

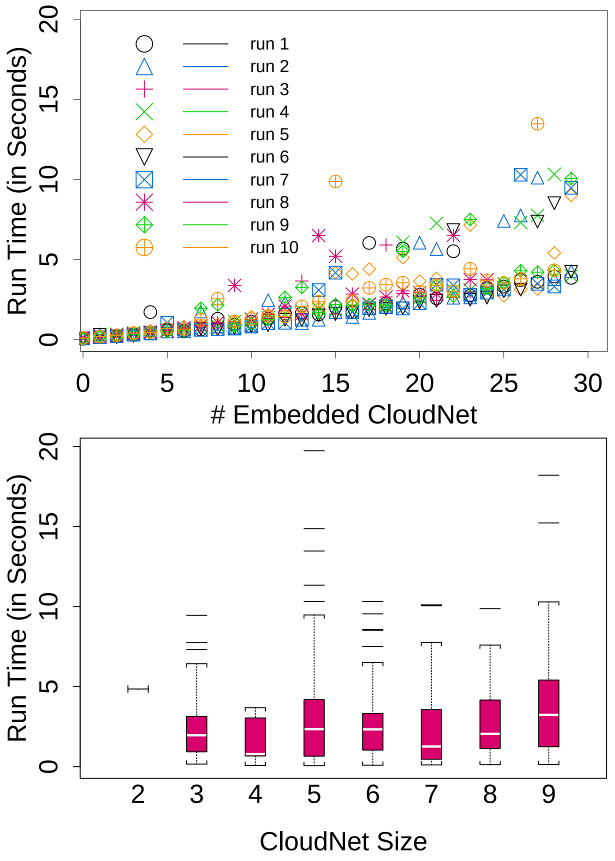
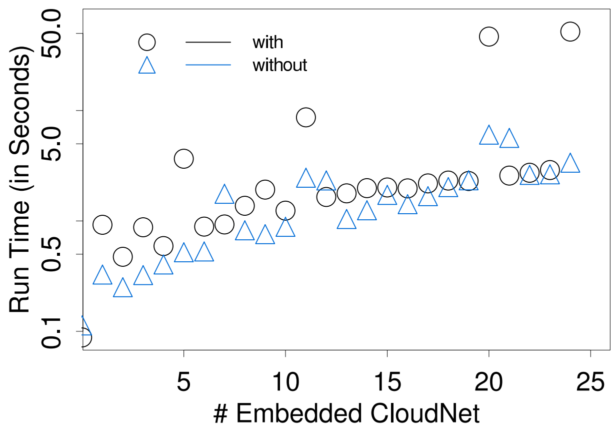


Fig. 2. Runtime per embedded CloudNet without migration for multiple runs (*top*: time series, *bottom*: boxplot).

IV. Experiments

While the potential benefits of rigorous and generic re-optimizations may be attractive, the question remains on what is the time frame needed to perform them. In the following, we report on the simulations we have performed using our network virtualization prototype architecture [18]. Our prototype which is currently based on VLANs as a link virtualization technology (this however is irrelevant for our simulations here) contains a module to generate the MIP programs presented in this paper. As a solver, IBM's standard `CPLEX` software is used in deterministic mode with a limit of six concurrent threads on a 8-core Xeon server running at 2.5GHz. We will focus on the out-sourcing scenario, and refer the reader to [16] for the other scenarios.

In order to model the physical substrate network, we extracted *Rocketfuel topologies* [19]. Connected subsets of these graphs are also used to describe the topology of the CloudNet requests. For the OC use case, the virtual cloud nodes of the CloudNets are partitioned into freely allocatable cloud resources (CR) and fixed access points (AP) (e.g., connection points to corporate subnets of the requesting entity). Unless stated otherwise, substrate network elements feature capacity



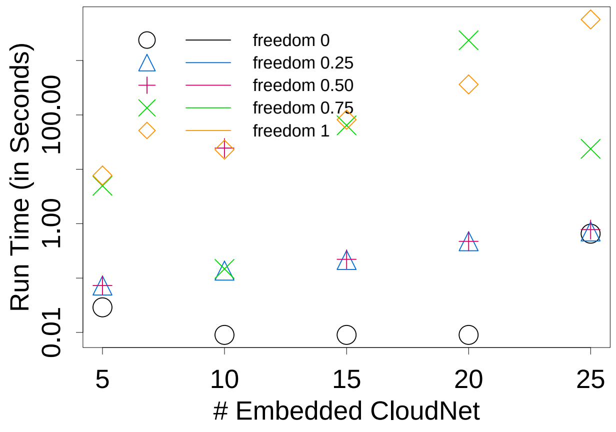


Fig. 3. *Top*: Runtime (log scale) per embedded CloudNet with and without migration (sample run). *Bottom*: Runtime (log scale) per embedded CloudNet for different degrees of freedom (sample runs). `CPLEX` is used in deterministic mode yielding constant runtimes.

for fifteen virtual network elements, no placement preferences are given (with destination independent migration penalties for all node pairs and with unit weights) and the embeddings are optimized with the load objective function.

Concretely, for each CloudNet we chose (uniformly at random) between one and three flexible cloud nodes and between one and seven fixed access nodes. We refer to the percentage of flexible nodes in the CloudNet by the variable $freedom \in [0, 1]$. For our experiments, we use a substrate network of twenty-five nodes, and we iteratively place incoming CloudNet requests. Evaluations are repeated ten times, and all CloudNet requests are accepted as long as resources are available. We study scenarios with and without migration.

Figure 2 shows the runtime (real time, in seconds) required to embed CloudNets iteratively (one after the other, sorted on the x-axis) in a scenario without migrations. There are several takeaways from these experiments: First, we observe that the embedding times are small (never exceeding 14 seconds). Moreover, depending on the load on the substrate network (the number of already embedded CloudNets), the runtime increases slightly. The data also exhibits a relatively high variance, which can be explained by the randomized nature of the to be embedded CloudNets (in terms of size and nature).

The runtimes generally increase if we enable the option to migrate, see Figure 3 (*top*), although there are instances where the results are comparable. This is to be expected as migration increases flexibility and therefore the complexity of the MIP. An interesting feature of integrating migration support is that we can at any time check if a subset of the resources, e.g., half of the network is sufficient to fulfill the demand. In the above cases such a run takes on average 2.73 seconds with a standard deviation of 0.42 seconds.

As the above experiment suggests, the main parameter that determines the time complexity of the embeddings is the freedom of the node placement. We conducted on a series of experiments where the CloudNet size and the proportion of CRs, i.e., the variable $freedom$, varies. The findings are summarized in Figure 3 (*bottom*) which confirms this dependency. Interestingly, despite the flexibility of the CloudNets and the existing load on the substrate, the runtimes are still in the range of several minutes.

We conclude that although the option to migrate and the placement flexibility effect the execution times, optimal solutions for relatively large problems are feasible and can be computed in reasonable time (e.g., within hours or over night). Moreover, as the runtimes without migration support are lower than their counterparts, hybrid designs, where incoming CloudNets are first placed ad-hoc, and persisting ones are optimized regularly (by an offline, background process) seem to be attractive.

V. Related Work

There has been a significant interest in virtual and cloud networks and over the last years. The reader is referred to the recent surveys [5] and [10]. The work described here is conducted in the context of our network virtualization project where we develop a CloudNet prototype (see [18] for an overview of the prototype and [17] for the used resource description language).

The virtual network embedding problem has already been studied intensively and there exist also interesting connections to resource allocation and migration problems in grid environments. [14], [20]. The survey by Belbekkouche [2] provides a nice overview of allocation and embedding algorithms. Most of the algorithms proposed in the literature today are heuristics, and come without any formal performance guarantees. For example, Fan and Ammar [8] study dynamic re-configurable topologies to accommodate communication requirements that vary over time, Zhu and Ammar [24] consider virtual network assignment problems with and without reconfiguration but only for bandwidth constraints, Ricci et al. [15] pursue a simulated annealing approach, and Lu and Turner [13] seek to find the best topology in a family of backbone-star topologies. Many approaches in the literature fail to exploit the flexibility to embed virtual nodes and links simultaneously and solve the two mappings sequentially (e.g., [12]), which entails a loss of efficiency [6]. To deal with the computational hardness, Yu et al. [22] 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. [3] is on re-optimization mechanisms that ameliorate the performance of the previous virtual network embedding algorithms in terms of acceptance ratio and load balancing; their algorithm is able to prioritize resources and is evaluated by simulations. Virtual network embeddings have also been studied for cross-provider settings [21] and from a distributed computing perspective [11].

There are other mathematical programming approaches for network embedding problems: Chowdhury et al. [4] present an integer embedding program and pursue a relaxation strategy, applying randomized and deterministic rounding to find approximate solutions. The presented graph extension approach supports exact solutions for placements where interconnected virtual nodes do not share candidate substrate nodes, as it would add bogus resources otherwise. Another recent interesting work is by Zhang et al. [23] who initiate the discussion of opportunistic resource sharing to improve the resource usage further, and also study two simple heuristics to include topology-awareness in the mapping process. However, both the algorithms in [4] and in [23] are much less generic than ours (e.g., in the combination of supported resources, migrations, and link types, among many more), and focus more on time-constrained and heuristic solutions. Finally, there is also work on the related problem of access control [7].

In contrast to the literature reviewed above, our work puts an emphasis on *generality* and *quality* of the embeddings, and attends to a scenario where initially quickly placed networks are subsequently optimized with less stringent time constraints. To the best of our knowledge, there is no algorithm to embed CloudNet like networks in a manner whose generality and flexibility is close to ours. In particular, none of the solutions above can handle all the heterogeneous links occurring in practice and map, e.g., a (wireless) broadcast link onto a set of asymmetric and full-duplex links; besides the virtual links, also the expressiveness of the node mapping is restricted, and we are not aware of any algorithm which e.g., allows to capture loads induced due to packet rates of the flows in a CloudNet; finally, we believe that the support of cost-aware migration is crucial, as is lies at the heart of network virtualization.

VI. Conclusion

This paper has argued that CloudNets embeddings do not always have to be computed within seconds in time. For example, rigorous optimizations of more long-lived (and maybe initially heuristically placed) CloudNets may pay off in the long run, especially as network virtualization can support seamless migrations. Another scenario where it is worthwhile to invest more time in the computation of CloudNets embeddings, are CloudNet *templates*: CloudNet requests which appear frequently and may be known in advance, e.g., because they are part of a standard catalogue offered by the infrastructure provider. We have described such a rigorous and general embedding approach which is also used in our prototype system. We find that joint optimal embeddings of long-lived CloudNets are feasible for moderate size networks, especially in a management hierarchy as we envision it in our federated prototype architecture and implementation [18].

References

[1] D. Arora, M. Bienkowski, A. Feldmann, G. Schaffrath, and S. Schmid. Online strategies for intra and inter provider service migration in virtual networks. In *Proc. IPTComm*, 2011.

[2] A. Belbekkouche, M. Hasan, and A. Karmouch. Resource discovery and allocation in network virtualization. *IEEE Communications Surveys Tutorials*, (99):1–15, 2012.

[3] N. F. Butt, M. Chowdhury, and R. Boutaba. Topology-awareness and reoptimization mechanism for virtual network embedding. In *Proc. IFIP/TC6 NETWORKING*, 2010.

[4] K. Chowdhury, M. R. Rahman, and R. Boutaba. Virtual network embedding with coordinated node and link mapping. In *Proc. IEEE INFOCOM*, 2009.

[5] M. K. Chowdhury and R. Boutaba. A survey of network virtualization. *Elsevier Computer Networks*, 54(5), 2010.

[6] L. P. Cordella, P. Foggia, C. Sansone, and M. Vento. An improved algorithm for matching large graphs. In *Proc. Workshop on Graph-based Representations in Pattern Recognition*, 2001.

[7] G. Even, M. Medina, G. Schaffrath, and S. Schmid. Competitive and deterministic embeddings of virtual networks. In *Proc. 13th ICDCN*, 2012.

[8] J. Fan and M. H. Ammar. Dynamic topology configuration in service overlay networks: A study of reconfiguration policies. In *Proc. IEEE INFOCOM*, 2006.

[9] A. Feldmann, G. Schaffrath, and S. Schmid. Cloudnets: Combining clouds with networking. *ERCIM News*, (88):5657, 2012.

[10] A. Haider, R. Potter, and A. Nakao. Challenges in resource allocation in network virtualization. In *Proc. ITC Specialist Seminar on Network Virtualization*, 2009.

[11] I. Houidi, W. Louati, and D. Zeghlache. A distributed virtual network mapping algorithm. In *Proc. IEEE ICC*, 2008.

[12] J. Lischka and H. Karl. A virtual network mapping algorithm based on subgraph isomorphism detection. In *Proc. ACM SIGCOMM VISA*, 2009.

[13] J. Lu and J. Turner. Efficient mapping of virtual networks onto a shared substrate. In *WUCSE-2006-35, Washington University*, 2006.

[14] K. E. Maghraoui, T. Desell, B. K. Szymanski, and C. A. Varela. Malleable iterative mpi applications. *Concurrency and Computation: Practice and Experience*, 21(3):393–413, 2009.

[15] R. Ricci, C. Alfeld, and J. Lepreau. A solver for the network testbed mapping problem. *ACM SIGCOMM CCR*, 33(2), 2003.

[16] G. Schaffrath, S. Schmid, and A. Feldmann. Generalized and resource-efficient vnet embeddings with migrations. In *ArXiv Technical Report 1012.4066*, 2010.

[17] G. Schaffrath, S. Schmid, I. Vaishnavi, A. Khan, and A. Feldmann. A resource description language with vagueness support for multi-provider cloud networks. In *Proc. ICCCN*, 2012.

[18] G. Schaffrath, C. Werle, P. Papadimitriou, A. Feldmann, R. Bless, A. Greenhalgh, A. Wundsam, M. Kind, O. Maennel, and L. Mathy. Network virtualization architecture: Proposal and initial prototype. In *Proc. ACM SIGCOMM VISA*, pages 63–72, 2009.

[19] N. Spring, R. Mahajan, D. Wetherall, and T. Anderson. Measuring isp topologies with rocketfuel. *IEEE/ACM Trans. Netw.*, 12(1):2–16, 2004.

[20] C. A. Varela, P. Ciancarini, and K. Taura. Worldwide computing: Adaptive middleware and programming technology for dynamic Grid environments. *Scientific Programming Journal*, 13(4):255–263, 2005.

[21] Y. Xin, I. Baldine, A. Mandal, C. Heermann, J. Chase, and A. Yumerefendi. Embedding virtual topologies in networked clouds. In *Proc. 6th International Conference on Future Internet Technologies (CFI)*, pages 26–29, 2011.

[22] M. Yu, Y. Yi, J. Rexford, and M. Chiang. Rethinking virtual network embedding: substrate support for path splitting and migration. *ACM SIGCOMM CCR*, 38(2), 2008.

[23] S. Zhang, Z. Qian, J. Wu, and S. Lu. An opportunistic resource sharing and topology-aware mapping framework for virtual networks. In *Proc. IEEE INFOCOM*, 2012.

[24] Y. Zhu and M. H. Ammar. Algorithms for assigning substrate network resources to virtual network components. In *Proc. IEEE INFOCOM*, 2006.