Virtual Network Embedding with Collocation: Benefits and Limitations of Pre-Clustering

Carlo Fuerst, Stefan Schmid, Anja Feldmann
Telekom Innovation Laboratories (T-Labs) & TU Berlin, Germany
{carlo,stefan,anja}@net.t-labs.tu-berlin.de

***Abstract*—Given that mechanisms for resource isolation are in place, the collocation of virtual network (VNet) nodes is attractive as it reduces the inter-machine communication and hence improves the VNet embedding. However, existing VNet embedding algorithms either do not support the collocation of virtual nodes of the *same* VNet, or only support it implicitly by referring to the possibility to pre-cluster the VNet topology: this pre-clustered network forms the new VNet request and is embedded accordingly.**

This paper presents a pre-clustering algorithm OPTCUT that is optimal in the sense that it minimizes the amount of link resources needed for the embedding. It is based on a smart linear program formulation that ensures fast solutions. OPTCUT can be used together with any existing VNet embedding algorithms, and we show that it can greatly improve the state-of-the-art embedding algorithm SecondNet [16]. The paper also describes a simple algorithm LOCO that *directly* supports collocation. This algorithm is part of a novel and generic VNet embedding framework METATREE which may be of independent interest.

We compare the performance of the pre-clustering approaches with the direct VNet embeddings by LOCO, and find that pre-clustering also has its limitations. In particular, the information gap between the pre-clustering and the actual algorithm, as well as an inaccurate estimation of the distribution of remaining substrate resources, may lead to a low network utilization.

I. INTRODUCTION

Network virtualization [8] is a new networking paradigm which combines node virtualization (e.g., Xen) with link virtualization (e.g., OpenFlow or, in the wide-area, MPLS), to provide the abstraction of a *virtual network (VNet)*. A VNet comes with explicit resource and performance guarantees, also on the networking part, and is logically isolated from other VNets sharing the physical resources of the so-called *substrate network*.

Making networking a first-class citizen has the advantage that the *access* to cloud resources becomes more deterministic, and hence may reduce the variance and duration (price) of an execution. [10], [22] Accordingly, in a virtualized environment with resource guarantees [2], bandwidth reservations can be reduced by collocating virtual nodes: if the nodes of a VNet are mapped to the same physical machines, communication can stay local (e.g., goes via the ring buffer). [21]

Interestingly, however, most of the state-of-the-art VNet embedding algorithms described in the literature do not explicitly support the collocation of virtual nodes of the same VNet, but only collocate nodes of *different* VNets. Moreover, while several papers acknowledge the benefits of collocation, they refer to the possibility to *pre-cluster* VNets in advance, i.e., to transform the original VNet request to a VNet where a single virtual node describes multiple virtual nodes; however, no explicit algorithm is given on how to compute such a *pre-clustering (PC)*.

We, in this paper, argue that the collocation of nodes of the same VNet has many advantages over the collocation of nodes of different VNets. First, while collocating nodes of different VNets indeed allows to share the physical resources of a single machine better, there are no gains in terms of bandwidth reservations. Moreover, collocation of different VNets (and hence maybe tenants) may introduce security threats, e.g., regarding the shared hardware registers [24], or render performance less predictable [26], [27].

This paper also argues that pre-clustering itself may not be sufficient to reap the full benefits of collocation. In fact, we show that pre-clustering comes at an inherent resource price.

Our Contributions. This paper makes the following contributions. We first address the obvious open question of how existing VNet embedding algorithms can be extended to support collocation. Concretely, we describe an algorithm OPTCUT which computes an optimal VNet pre-clustering in the sense that, given an estimation of the available resources in the physical network, the amount of network resources required for the VNet embedding is minimized. While this problem is computationally hard in general, our OPTCUT is based on a smart linear program formulation which facilitates efficient solutions, even for large VNets.

In order to compare the performance of existing VNet embedding algorithms (enhanced with PC) to a direct embedding approach, we present a simple collocation algorithm LOCO ("location correlation"). LOCO seeks to exploit collocation opportunities by embedding multiple nodes on the same physical node, by also taking into account potential network capacity constraints. LOCO is based on a graph-growing scheme, which quickly results in compact VNet embeddings.

To further improve the execution time in large substrate networks, LOCO uses a hierarchically clustered representation of the substrate network, which allows to guide the embedding process. We believe that this hierarchical clustering approach may be of independent interest, as it can serve as an algorithmic framework (called METATREE) for various VNet embeddings tasks. For example, if resources cannot be perfectly isolated, METATREE can be used to only collocate VNets with low interference.

We report on our extensive simulation study where we

compare the benefits and limitations of pre-clustering, and investigate the performance of LoCo in various settings. We show that pre-clustering can be very beneficial to increase the VNet acceptance ratios of existing embedding algorithms without collocation support (such as SecondNet [16]). Pre-clustering turns out to be particularly useful in scenarios where VNets are large and highly connected. However, we also show that relatively to direct embeddings (e.g., using LoCo), pre-clustering suffers relatively more in scenarios with almost oversubscribed substrate networks where the remaining resources are fragmented.

Background on VNet Embedding. We study a setting where the VNet request comes in the form of an (undirected) graph $G = (V_G, E_G)$ (the "guest graph"), where virtual nodes V_G and virtual links E_G are annotated with resource requirements (e.g., memory or bandwidth). [8] The embedding problem asks for a mapping of the VNet nodes and VNet edges onto an (undirected) physical network $H = (V_H, E_H)$ (also called the "host graph" or *substrate network*) with certain capacity constraints on nodes and links. Concretely, each virtual node $u \in V_G$ needs to be mapped to *any* (and exactly one) physical node node $v \in V_H$ with sufficient capacity, henceforth referred to by $v = \mu(u) \in V_H$; but a physical node may host multiple virtual nodes (subject to capacity constraints).

Accordingly, each virtual link $e = (u, v) \in E_G$ is mapped to a (possibly empty) path between $\mu(u)$ and $\mu(v)$. We will focus on unsplittable paths, i.e., a virtual edge e is mapped onto a single path. However, many of our results are also applicable in settings where graphs are directed or flows splittable.

II. Optimal Pre-Clustering

Many existing algorithms refer to the possibility to pre-cluster VNets in order to support collocation. *How* to pre-cluster a graph however is often not described. We present an *optimal* pre-clustering algorithm, with respect to a given (potentially exact) estimation of the resource availability in the substrate network and with respect to the specified objective function (in our case: overall link resources between clusters). This algorithm (short: OptCut) is based on a *Mixed Integer Program (MIP)* formulation. A MIP consists of a linear objective function and a set of linear constraints. Once a problem can be cast in this form, standard software solvers such as *CPLEX* or *Gurobi* can be applied to it. While the MIP is computationally hard to solve in general (and in particular, the pre-clustering problem can be shown to be NP-hard [6]), it turns out that for reasonably sized VNets (up to 30 nodes), optimal solutions can be computed within seconds. However, note that the performance of a MIP critically depends on how the MIP is formulated (see, e.g., [5] for an introduction, or Figure 1 described later in more detail). In the following, we will present a smart MIP formulation which avoids many symmetries and yields good solution times. Moreover, the fact that OptCut computes optimal solutions also gives us the required baseline for studying the limitations of any pre-clustering algorithm.

OptCut works as follows. It has some knowledge or *estimation* of the available node (MAX_V) and link (MAX_E) resources in the substrate. As an input, the algorithm takes a VNet topology $G = (V, E, W)$ where W represents weights, i.e., resource requirements (the constants of the MIP). The output is a partitioning $\mathcal{P} = \{C_1, C_2, \ldots, C_m\}$ of the virtual nodes V (the variables), such that all nodes in V are mapped to exactly one cluster C_i, and capacity constraints on the physical network are respected. Obviously, the cluster will represent the set of collocated nodes. Our objective is to reduce the amount of link resources needed, and hence we seek to minimize the number of inter-cluster links. The links between clusters form the *cut* K. Note that the number of clusters m in the partition is subject to optimization as well: the best number of clusters is chosen such that the cut is minimized.

Constants:

$$\text{Set of nodes:} \quad V \tag{1}$$
$$\text{Set of edges:} \quad E \subset V \times V \tag{2}$$
$$\text{Weights:} \quad W : V \cup E \to \mathbb{R}^{\geq 0} \tag{3}$$
$$\text{Maximal node resources:} \quad MAX_V \tag{4}$$
$$\text{Maximal link resources:} \quad MAX_E \tag{5}$$
$$\text{Larger nodes:} \quad \rho : V \to 2^V \tag{6}$$

Variables:

$$\text{Node mapping:} \quad \text{alloc}_V : V \times V \to \{0, 1\} \tag{7}$$
$$\text{Auxiliary variable:} \quad x : E \times V \to \mathbb{R}^{\geq 0} \tag{8}$$

Constraints:

$$\forall v \in V : \sum_{v' \in V} \text{alloc}_V(v, v') = 1 \tag{9}$$
$$\forall v \in V : \sum_{v' \in \rho(v)} \text{alloc}_V(v, v') \leq 0 \tag{10}$$
$$\forall v \in V : \sum_{v' \in V} \text{alloc}_V(v', v) \cdot W(v') \leq \text{alloc}_V(v, v) \cdot MAX_V \tag{11}$$
$$\forall (v_1, v_2) \in E, v \in V : \text{alloc}_V(v_1, v) + \text{alloc}_V(v_2, v) \leq c((v_1, v_2), v) \tag{12}$$
$$\forall (v_1, v_2) \in E, v \in V : \text{alloc}_V(v_1, v) \geq c((v_1, v_2), v) \tag{13}$$
$$\forall (v_1, v_2) \in E, v \in V : \text{alloc}_V(v_2, v) \geq c((v_1, v_2), v) \tag{14}$$
$$\forall v \in V_V : \left[\sum_{(v_1, v_2) \in E_V} \text{alloc}_V(v_1, v) + \text{alloc}_V(v_2, v) - 2c((v_1, v_2), v) \right] \cdot W((v_1, v_2)) \leq MAX_E \tag{15}$$

Minimize:

$$\sum_{(v_1, v_2) \in E} \left[1 - \sum_{v \in V_V} c((v_1, v_2), v) \right] \cdot W(v_1, v_2) \tag{16}$$

Constants (1), (2) and (3) represent the weighted VNet $G = (V, E, W)$. The maximal link and node resources are given by Constants (5) and (4).

To describe the cluster to which a virtual node belongs, we use the allocation variable (Variable (7)) that maps a node to a cluster; that is, the variable will take the value 1 iff. the node is mapped to the cluster, and 0 otherwise. These sets and variables are enough to describe the node mapping constraints.

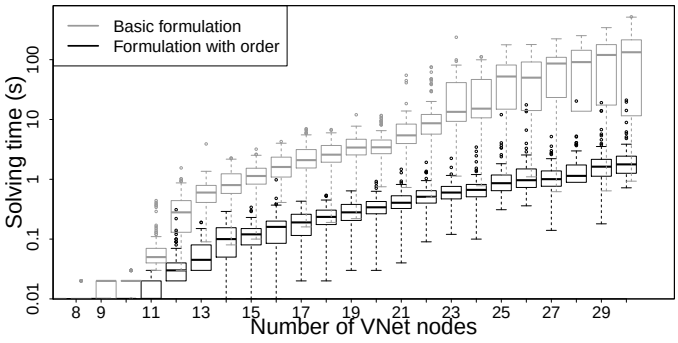

Fig. 1. Execution times to compute optimal pre-clustering solution, as function of VNet size and depending on the MIP formulation. For this experiment, we generated 100 random topologies for each VNet size. (Setup: Gurobi 5.5 MIP solver on Intel Xeon L5420, 8 × 2.50GHz, 16 GB RAM.)

Constraint (9) ensures that each node is mapped to exactly one cluster while Constraint (11) guarantees that no cluster exceeds the maximally available resources.

In order to compute the cut, an auxiliary variable (Variable (8)) can be used which indicates whether both endpoints of a link are mapped to the same cluster. Note that that the variable does not need to be binary, which is useful for the execution time. For a proper realization, we use Constraints (12), (13) and (14). The variable also allows us to describe the communication capacity constraint of the clusters in Constraint (15).The objective function (Objective (16)) minimizes resources used for links between clusters.

While the variables and constraints described so far are sufficient to solve the pre-clustering problem and find an optimal solution, we can improve the program formulation to speed-up the computations. To this end, we introduce a total order ρ on the nodes V: ρ is a mapping of a virtual node v, to the set of virtual nodes which are larger than v. This is useful to reduce the symmetry—often the main reason for long execution times. Concretely, we can leverage the total order ρ in combination with Constraint (10). The constraint states that each node may only be mapped to a partition which is represented by a node of order smaller or equal than itself. For example, Nodes 1 and 2 may be mapped to the partition represented by Node 1, and Nodes 3 and 4 are mapped to the partition represented by Node 3.

Figure 1 illustrates the impact of the MIP formulation: without the order ρ technique, the solving times are almost two orders of magnitude larger. For VNet sizes smaller than 7 nodes, the execution time is below 0.1 seconds.

III. Direct Collocation

Before presenting our direct collocation algorithm LoCo, we propose a more general virtual network embedding framework, henceforth simply referred to by MetaTree. MetaTree can be used together with many different embedding algorithms. For example, in a wide-area setting, our framework could be used together with *ViNE* [7] to incorporate geographic constraints; for fast embeddings in data centers, it could be used together with *SecondNet* [16]; finally, for collocation-aware embeddings, it could be used together with

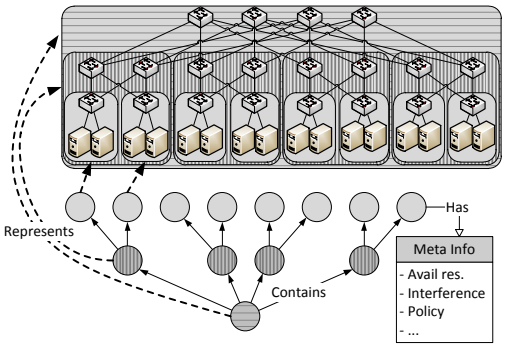

Fig. 2. MetaTree partitions a given substrate hierarchically.

LoCo (described in more detail later).

A. Algorithmic Framework MetaTree

MetaTree is based on hierarchical partitions of the substrate network. The partition hierarchy can be seen as a *(logical) tree* T; the root of T is a single-cluster partition that represents the entire physical network, and its descendant clusters partition the network into smaller contiguous (i.e., connected) component. Generally, the lower in the tree, the smaller the corresponding clusters (see Figure 2).

At each node $v \in T$ of the tree, meta-data is stored that can guide the embedding process. For example, the meta-data may include information on the total amount of available resources in the corresponding subtree or cluster; alternatively, it may provide additional information, such as topological information (e.g., mincut) or the type of workloads running in the corresponding cluster. The latter information may be useful, e.g., to learn about potentially interfering workloads [28] (e.g., a disk-intensive VNet may not be embedded together with another disk-intensive VNet, or two services which should not be executed on the same machine); however, we will consider scenarios where resources can be isolated well from each other, and will concentrate on the available resources.

The embedding algorithm starts at the root of T, and iteratively proceeds down the tree layers towards the leaves. At each internal node, the algorithm decides on which cluster of the corresponding partition to embed the VNet. For example, if the meta-data stored at the node $v \in T$ indicates that there are still sufficient resources in the sub-cluster represented by a single descendant of v, the algorithm recursively checks the corresponding sub-trees; on the other hand, if the VNet is so large that it cannot be embedded in a sub-cluster, it is embedded (using any of the algorithms described below) in the cluster represented by v.

While the choice of cluster is flexible within the MetaTree framework, we in this paper will focus on the *acceptance ratio* objective (measured in terms of physical node utilization): we want to accept and embed as many VNets as possible. Accordingly, we want to map virtual nodes close together and maximize the residual capacities. Thus, for the embedding, the cluster with minimal available resources should be chosen such that the VNet still fits. Of course, for different objectives, e.g., minimizing the max load, the opposite strategy is better.

We do not provide any explicit algorithm to compute the hierarchical partition, but refer the reader to the various existing algorithms, see e.g., the surveys [6], [13] or the recent result by Krauthgamer et al. [18]. Of course, for specific substrate topologies, tailored and optimized algorithms should be used. We will later present the algorithm used in our simulation to cluster the considered data center topologies (e.g., the FatTree). In general, we will focus on hierarchical partitions where on the next higher layer, a given cluster is partitioned into two clusters; clusters of the same layer are roughly equally sized.

B. Collocation with LoCo

Our collocation algorithm LoCo jointly places virtual nodes and links. We will keep the description of LoCo simple and general, and avoid discussions of several (obvious) optimizations. In particular, note that LoCo can essentially be applied to *any* substrate topology. We believe that in this form, LoCo is best suited to manifest our claims.

In order to embed a virtual network G (the "guest graph"), first a peripheral start node $s \in V(G)$ (a node which maximizes the distance to all other VNet nodes) is mapped to an arbitrary substrate node with sufficient capacity. Subsequently, LoCo maintains the following data structures: a set M of already *mapped virtual nodes* (initially, $M = \{s\}$), and an array P of *pending virtual nodes* which still need to be mapped (initially, $P = (\Gamma(s))$ where $\Gamma(s)$ denotes the neighbors of s). After mapping a new virtual node $u \in P$, LoCo moves u to M and maps all virtual links $\{u, v\}$ which connect this node u to all already mapped virtual nodes $v \in M$. Nodes neighboring to u which are not part of M or P yet are put into the pending node array P. This is repeated until all nodes are *mapped*.

Concretely, LoCo sorts the pending nodes in decreasing order of *maximal* capacity of an incident virtual link connecting the pending node so any mapped node. Ties are broken by preferring nodes with minimal virtual node capacities. The intuition behind this approach is that if links with capacities are mapped first, the cost benefits are potentially higher and dead ends can be detected faster. Similarly, mapping nodes with lower demand first has the advantage of being able to collocate more virtual nodes.

In case of embedding failure, LoCo backtracks over the alternative substrate nodes on which s could be embedded. Although it may yield earlier and/or better solutions, we do not backtrack over other nodes. (In addition, in our simulations, we set a hard limit of 1s per VNet embedding.)

Let us now elaborate more on the node and link mapping functions. Both mappings are essentially breadth-first-search based. When mapping a *pending* virtual node u with the corresponding virtual link $\{u, v\}$ connecting it to a *mapped* virtual node v, we first check whether the substrate node on which v is hosted still has sufficient capacity to host also u. If this is fulfilled, we additionally perform the following *forward checking*: We verify whether there are substrate links left on that substrate node which have sufficient resources to embed the links incident to v (or, if there is capacity to even host neighbors of v, the corresponding links). Thus, forward checking avoids unnecessary backtracking by verifying that all adjacent virtual links can be embedded, *before* a virtual node is mapped to a substrate node.

Algorithm 1 The LoCo Algorithm

Require: VNet $G = (V, E)$, $M = \{s\}$ for some $s \in V(G)$, $P = (\Gamma(s))$
  while $|P| > 0$ **do**
    sort P (* decreasing link capacities *)
    choose $u = P[0]$ (* next node to map *)
    map u (* forward checking *)
    map $\{u, v\} \quad \forall \quad v \in M$**, where** $\{u, v\} \in E(G)$
    $M = M \cup \{u\}$ **and** $P = P \setminus \{u\}$
  end while
  if (embedding failed), **backtrack** on s

While LoCo performs well as a stand-alone algorithm already, in the remainder of this paper, we will use it within the MetaTree framework: LoCo will start at the root of the partition hierarchy and going down the levels, greedily determines the smallest cluster which contains sufficient capacity to embed the entire VNet.

IV. Experiments

Given our pre-clustering and direct collocation algorithms OptCut and LoCo, we can study the number of embeddable VNets in a given substrate network in different scenarios. We have developed a simulation framework which includes several additional algorithms besides OptCut and LoCo. Figure 3 gives an overview: in a nutshell, we distinguish between VNet requests which first go through a pre-clustering step and only then are embedded by some existing embedding algorithm, and VNets which are embedded directly. The input to our algorithms is a sequence of VNets arriving over time.

For direct embeddings, besides LoCo, we also consider the *SecondNet (SNet)* algorithm from [16]. As pre-clustering algorithms, besides OptCut, we use the Farhat clustering heuristic, as well as LoCo itself, but now only to collocate nodes. These pre-clustered VNets (the "modified requests") are then embedded using the SecondNet algorithm. To avoid ambiguous terminology we will use LoCo*, OptCut* and Farhat* to denote the combination of SecondNet and the pre-clustering algorithms.

In general, as a main performance criterion, we will focus on the number of simultaneously embeddable VNets. More specifically, since different VNets have different sizes, we propose to study the *utilization* of the physical substrate network as a yardstick for our evaluation: the utilization is defined as the ratio of the node resources actually used by the VNets in the physical network, divided by the overall amount of physical node resources (the overall "capacity").

Pre-Clustering Heuristics. In addition to (and for comparison with) our pre-clustering algorithm OptCut, we also studied the performance of pre-clustering heuristics (without qual-

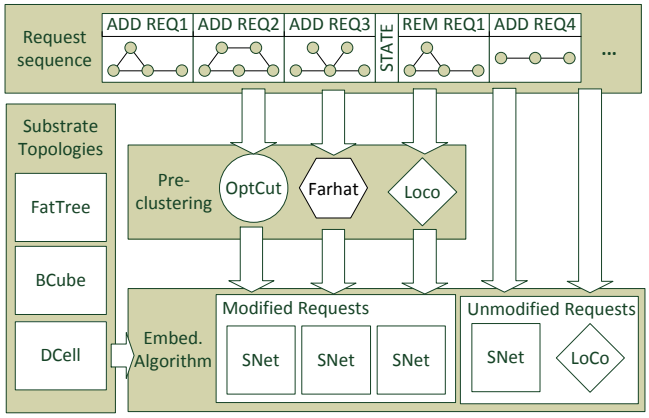

Fig. 3. Overview of experiment pipeline.

ity guarantees). In particular, we use *Farhat's algorithm* [13] which is based on graph growing. Like most graph growing algorithms, Farhat's algorithm is very fast. To make the initial algorithm suitable for our problem, we modified the version for graphs presented by Elsner [13]. Instead of aiming for a specific number of clusters like the original algorithm does, we group nodes together until either: (1) the next node which should be added to the current cluster is too big to fit into the cluster; in this case we generate a new cluster according to the definition of the algorithm. Or: (2) all nodes are mapped to a cluster. In this case the execution finished.

Alternatively, we can also use LoCo itself to compute a heuristic pre-clustering: we try to embed the VNet request on a substrate with the estimated capacity using LoCo. Since the resulting placement will be collocation-aware, we interpret the sets of virtual nodes mapped to the same substrate node as a cluster. To pre-cluster the virtual network with LoCo we have to generate a simulated substrate. This substrate consists of nodes with MAX_V resources, which are connected to a central node with no resources. All links have capacity MAX_E.

Request Model. In general, we model VNet topologies as random graphs of size between 2 to 10 nodes (chosen uniformly at random). Two types of random graphs are considered: (1) *General random graphs* where any two nodes are connected with probability $p = 0.15$; in case a VNet is not connected, another random topology is generated. (2) *Random binary trees*.

A VNet request must either be immediately accepted and the VNet embedded, or the request is rejected. We do not consider access control aspects, and all algorithms in our simulation will accept a VNet request whenever it can be embedded. In order to be able to compare different executions, the same generated VNet arrival sequence is fed to all algorithms.

The duration of a VNet follows an exponentially distributed random variable with mean $\lambda = 10$. If not stated differently, we consider a high-load scenario where initially and before starting the simulation, a maximal number of VNets is embedded. Subsequently, whenever the duration of a VNet expires, we immediately schedule a new random VNet request. Depending on this VNet requests size, it may or may not be embeddable; if the former is the case, we repeat generating VNet requests until a request cannot be embedded anymore. At this point we measure the utilization. Independent of the elapsed time between two measurment points, we treat all measurement points as equal.

If not stated otherwise, each experiment is repeated until the experiment time exceeds 500λ . The substrate topology is a *FatTree* [1] with 432 hosts, connected by a set of 12-port switches. Each physical resource (node and link) has a capacity of 4 units, while VNet elements (nodes and links) have a demand of one unit. As shown in [14], this combination of node and link resources yields the most interesting tradeoff, where collocation can impact the performance; of course relatively lower link resources only improve collocation benefits further. VNets are general graphs and the substrate is a FatTree with 432 nodes. However, for comparison, we also implemented the data center topologies *BCube* [15] and *DCell* [17].

MetaTree Framework. In order to tailor our embedding framework (Section III) to the specific substrate topology, we use the following approach. For all topologies considered in our simulations, the FatTree, the BCube, and the DCell topology, to compute the next partition with larger clusters, we exploit their recursive definition to repeatedly and greedily merge the two smallest clusters from the previous partition. As the approach is reminiscent of Huffman coding procedure, we will henceforth refer to the algorithm by Huf.

We leverage the symmetries in the FatTree topology by using the hop-count as a metric for the partition, which theoretically generates a perfect hierarchical decomposition. However, in order to be able to tune the cluster size at a finer granularity in larger FatTree topologies, Huf generates additional clusters and adds them to the hierarchical decomposition. These clusterings are generated in a greedy manner: For each partition on layer ≥ 1, we consider the cluster sizes at the next lower layer. We then repeatedly merge the two smallest clusters in the list (in a "Huffman-tree manner"), until only one cluster is left. Note, that for large instances, where maintaining the state of all this partitions might hit resource limitations, we can simply adjust the number of partitions which are merged together, in order to reduce the total number of partitions.

On BCubes, a similar Huf procedure can be used. A BCube is defined recursively: an (n, k)-BCube (where n is the number of ports per switch and k is the maximum level of hierarchy) consists of multiple $(n, k-1)$-BCubes. Repeatedly using all smaller BCubes as clusters will result in an perfect hierarchical decomposition. Using this as a basis we can again generate clusters in the greedy manner as described above. Also DCell topologies can be clustered analogously.

A. The Benefit of Collocation

We first investigate how the utilization achieved by the different embedding algorithms depends on the possibility to collocate virtual nodes from the same VNet. Figure 4 compares the performance of SecondNet with our different pre-clustering algorithms LoCo, OptCut and Farhat; we

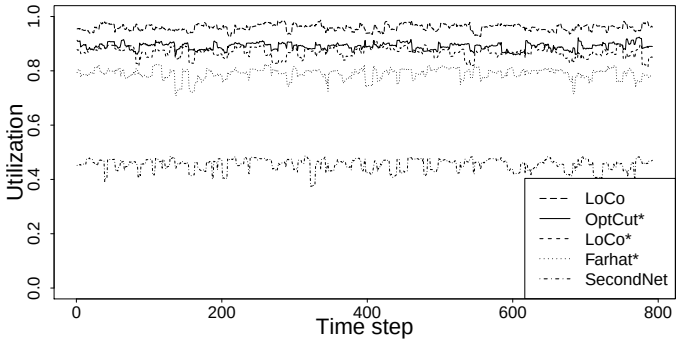

Fig. 4. Effective resource utilization of LoCo and SecondNet under different pre-clustering algorithms and without collocation. (Order of algorithms in legend corresponds to order of curves.)

also plot the performance of SecondNet and LoCo without pre-clustering. (The order of the algorithms in the legend corresponds to the order of the curves.)

We can see that *any* pre-clustering algorithm yields significantly higher utilizations compared to scenarios where SecondNet does not modify the input. However, a direct collocation algorithm such as LoCo outperforms any other algorithm: compared to SecondNet without PC, the resource utilization is improved by almost 100% (LoCo achieves an absolute utilization of 100% while the utilization of SecondNet is around 50%); however, LoCo also outperforms any PC algorithm by between 10%-20%.

Among the pre-clustering algorithms, OptCut performs the best, but the utilization of LoCo is almost as high. In fact, the figure shows that sometimes LoCo pre-clustering is even slightly better. This can be explained by the fact that: (1) OptCut is only optimal w.r.t. the amount of resources on the edge cut; if bandwidth is not the bottleneck, this does not affect the embedding quality. And (2) OptCut is optimized for a single VNet embedding, but not for multiple requests arriving over time. Thus, there exists a price of being online and near-sighted in time.

The quality of the pre-clustering does depend on the specific algorithm. Since the experiments in Figure 4 were performed on VNet requests which are easy to pre-cluster (random VNet size with average 6), we also conducted a series of experiments to investigate the performance of the proposed algorithms in larger networks. (Note that by keeping the connection probability p, this also increases the expected number of links per node.) Figure 5 shows the results for Figure 4 but now for VNets of a given (fixed) size.

As expected, the larger the VNet in terms of nodes and links, the lower the utilization in general. However, the loss differs from algorithm to algorithm. While the utilization of LoCo decreases from 0.86 to 0.37, SecondNet with OptCut drops from 0.77 to 0.63. (These number represent median and average values at the same time: they all differ by less then 0.005.) For VNets with eleven or more nodes, a good pre-clustering (at least with OptCut) becomes relatively more useful, as good edge cuts are still found in the VNet; the bad performance of Farhat is due to its link-bandwidth agnostic collocation which leads to clusters with too high in-

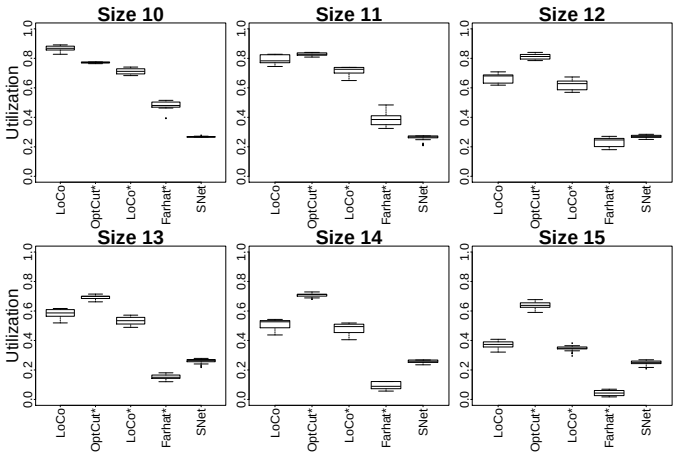

Fig. 5. Utilization under LoCo , different pre-clustering algorithms (OptCut*, LoCo* and Farhat*), as well as a "stand-alone" non-collocating SecondNet (SNet).

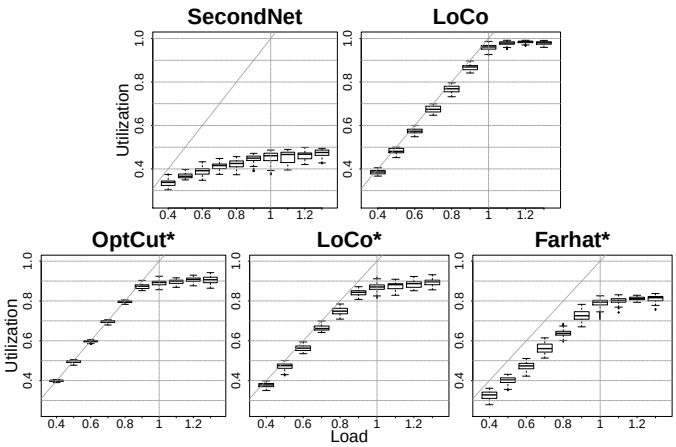

Fig. 6. Utilization of embedding algorithms under different loads. (Algorithms may perform slightly better than the diagonal due to minor load variations.)

coming/out-going bandwidth.

In conclusion, we find that if done properly (i.e., if good clusters are computed, e.g., with OptCut), pre-clustering can be a very attractive solution, especially under highly connected and large VNets. This is particularly interesting as OptCut can be used in combination with any other embedding algorithm.

Figure 6 shows the embedding performance of all suggested algorithms as a function of network load. In order to generate a load of $x\%$, we embed as many VNets as needed. Ideally, the utilization under load $x\%$ is $x\%$, see the diagonal. However, The vertical line at $x = 1$ represents the point at which all other experiments in this paper are executed. We see that between a load of 40% and 80%, SecondNet with OptCut outperforms all other algorithms, confirming the benefit of pre-clustering. For higher loads, LoCo performs best. This is a hint that LoCo is able to better exploit the residual node resources than SecondNet with OptCut, as the pre-clustering algorithm is agnostic to the topological fragmentation of the remaining available resources.

B. Limitations of Pre-Clustering

Our experiments so far confirm that pre-clustering can be very attractive for improving the performance of existing embedding algorithms. In particular, Figure 5 suggests that our pre-clustering algorithm performs very well for complex and large VNet with many virtual links. However, as we will discuss in this section, pre-clustering also has its limitations. In particular, while pre-clustering handles complex VNets well, it suffers from inaccurate information on the (possibly fragmented) available resources in the substrate network.

Recall that compared to an algorithm like LoCo which can collocate virtual nodes *directly*, pre-clustering divides the embedding process into two stages: the pre-clustering algorithm (henceforth called Alg1, e.g., OptCut) and the subsequent embedding algorithm (Alg2, e.g., SecondNet). This may lead to a situation where Alg1 had a specific and compact embedding in mind (i.e., in some part of the substrate network), but a rather different embedding is eventually chosen by Alg2. Moreover, in case Alg2 was programmed by someone else, or comes as a blackbox (e.g., is binary), Alg1 may not have an accurate and up-to-date view of the current network state. Rather, there exists an information gap.

From a different perspective, the pre-clustering discussion may also play a role in a scenario where Alg1 and Alg2 are executed by different economic entities. For example, when specifying the request, a customer may pro-actively pre-cluster its VNet request or determine the size of each virtual node, in order to increase the likelihood of collocation; the customer however faces the problem that while VNets with too large virtual nodes may not be embeddable by the cloud provider (running Alg2), too small virtual nodes may lead to a non-local VNet which may increase the execution times.

In order to investigate the impact of the two-stage embedding where Alg1 may have an inexact view of the physical network, we run different pre-clustering algorithms with different estimations of the available resources in the physical network.

In the following, we will focus on VNets describing random binary trees of size two to twenty nodes (uniformly at random). Due to the low degree of the virtual nodes, it is ensured that theoretically, these VNets can be embedded in the substrate. Figure 7 (*left*) shows the embedding performance of the different proposed algorithms.

Naturally, in this scenario where there is much collocation potential, all collocation-aware algorithms perform well. The reason why SecondNet performs better with LoCo than with OptCut is the distribution of cluster sizes (Figure 7 *right*): LoCo greedily groups as many virtual nodes as possible, generating one large cluster and many small ones. In contrast, the cluster of OptCut differ in size as the algorithm only focuses on the edge cut size.

Pre-clustering algorithms are limited by inaccurate resource estimations in the substrate network (MAX_V and MAX_E). Figure 8 shows the embedding performance for SecondNet combined with the three pre-clustering algorithms, as a function of the estimated resources.

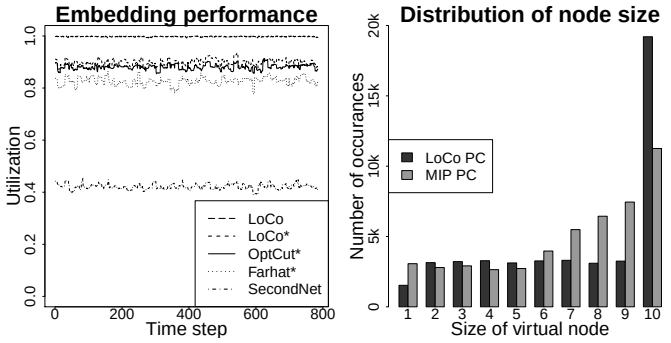

Fig. 7. The left figure shows the embedding performance for random graph VNets with 2 to 10 nodes and on a 432 host FatTree substrate. The right figure shows the distribution of the cluster sizes generated by OptCut and LoCo*.

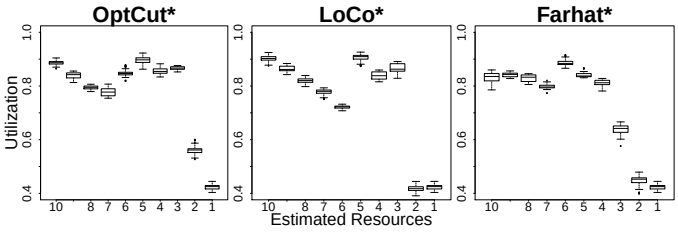

Fig. 8. Pre-clustering performance under different resource estimations. The x-axis shows how much of the actual node size (10) is used as the estimated node size.

Interestingly at first sight, the results suggest a particularly bad behavior of LoCo for an estimated resources capacity of 6. The reason for this is that every substrate node can only host one cluster containing 6 nodes, and LoCo will generate as many 6-node clusters as possible. However for $MAX_V = MAX_E = 5$ all algorithms generate good solutions. Since all discussed algorithms tend to generate clusters with the maximum possible size, we will end up with many 5 node clusters, which can be combined pairwise to fully utilize a node, which results in a high overall utilization. The good performance of LoCo could not be achieved with little information exchange between Alg1 and Alg2.

C. Other Substrate Topologies

Note that both our embedding framework in general as well as the collocation algorithm LoCo in particular do not rely on any specific substrate topology. All we need for the framework is a valid hierarchical partition of the substrate network. To complement the FatTree results, we have experimented with alternative substrate networks, namely with BCube and DCell topologies (using the partitions described before).

In general, we find that most of our insights so far are also valid for BCube and DCell topologies. To give one specific example, we compare, in Figure 9, a 12-port FatTree with a 8-port 2-layer BCube. To keep the resources at each node comparable, we initialized the FatTree with a resource capacity of 6 on each node and link, and the BCube with a capacity of 6 on each node and a capacity of 2 on each link, since hosts have three attached links for the given parameters. Clearly, the results indicate a very similar performance.

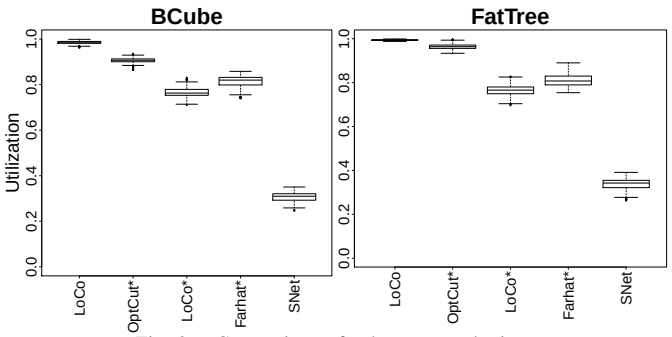

Fig. 9. Comparison of substrate topologies.

V. Related Work

The survey by Chowdhury et al. [9] gives a good introduction to network virtualization. In [4], Belbekkouche et al. recently collected and compared the state-of-the-art VNet embedding algorithms. We refer the reader to these papers for a more complete review of the literature in the field. Technologically, today, the question of how to realize VNets is fairly well-understood (see, e.g., [12] for a Software-Defined Networking perspective).

The VNet embedding problem has been studied from many different angles: there exist offline [20] and online [3] algorithms; while some algorithms rigorously compute optimal solutions, sometimes even supporting VNet migration and reconfiguration [25], due to the computational hardness of the problem, much literature focuses on heuristics [23].

We compare our work to the SecondNet algorithm presented in [16]. Our algorithm LoCo can be seen as an instance of a graph growing algorithm, and in this respect has similarities with the subgraph isomorphism detection algorithm by Lischka and Karl [19]. However, in contrast to [19], our algorithm short cuts unnecessary backtracking steps through its forward checking technique. Accordingly, the algorithm is faster, already without the MetaTree framework. While there already exist algorithms that directly support collocation, e.g., [25] or [11], we are not aware of any literature comparing the collocation benefits, also from the perspective of pre-clustering. Moreover, our algorithm LoCo is attractive for its low runtime, especially in combination with MetaTree. To the best of our knowledge, any existing embedding algorithm can be used in combination with our PC algorithm OptCut. We are not aware of any optimal pre-clustering algorithm yielding low runtimes (e.g., by symmetry breaking).

VI. Conclusion

The main lessons from this paper are that (1) using our symmetry breaking MIP formulation optimal pre-clusterings can be computed within a second even for large VNets with up to 30 nodes, (2) pre-clustering increases the number of VNets that can be hosted simultaneously on a given substrate network; however (3) unknown fragmentation of the residual resources constitutes a problem and already a simple algorithm such as LoCo can outperform a rigorous pre-clustering in combination with, e.g., SecondNet. Moreover, (4) the gains depend on the size and connectivity of the VNet. Finally, we have introduced (5) a VNet embedding framework which cannot only speed up the embedding runtimes by reducing the size of the to be considered substrate, but also improve the embedding itself by providing additional metadata about the network state.

Acknowledgements. The authors would like to thank Matthias Rost. Research is supported by the EU projects BigFoot and OFELIA, and a German Software Campus grant.

References

[1] M. Al-Fares, A. Loukissas, and A. Vahdat. A scalable, commodity data center network architecture. In *Proc. ACM SIGCOMM*, 2008.
[2] B. Hindman et al. Mesos: a platform for fine-grained resource sharing in the data center. In *Proc. USENIX NSDI*, 2011.
[3] Nikhil Bansal, Kang-Won Lee, Viswanath Nagarajan, and Murtaza Zafer. Minimum congestion mapping in a cloud. In *Proc. ACM PODC*, 2011.
[4] A. Belbekkouche, M. Hasan, and A. Karmouch. Resource discovery and allocation in network virtualization. *IEEE Communications Surveys Tutorials*, (99):1–15, 2012.
[5] Dimitris Bertsimas and Robert Weismantel. *Optimization over integers*. Dynamic Ideas, 2005.
[6] Bradford L. Chamberlain. Graph partitioning algorithms for distributing workloads of parallel computations. Technical report, 1998.
[7] K. Chowdhury, Muntasir Raihan Rahman, and Raouf Boutaba. Virtual network embedding with coordinated node and link mapping. In *Proc. INFOCOM 2009 and IEEE/ACM Trans. Netw. (ToN) 2012*, 2012.
[8] Mosharaf Kabir Chowdhury and Raouf Boutaba. A survey of network virtualization. *Elsevier Computer Networks*, 54(5), 2010.
[9] NM Chowdhury and R. Boutaba. A survey of network virtualization. *Computer Networks*, 2009.
[10] P. Costa, M. Migliavacca, P. Pietzuch, and A. L. Wolf. Naas: Network-as-a-service in the cloud. In *Proc. USENIX Hot-ICE Workshop*, 2012.
[11] D. Dietrich, A. Rizk, and P. Papadimitriou. Multi-domain virtual network embedding with limited information disclosure. In *IFIP Networking 2013 Conference*.
[12] D. Drutskoy, E. Keller, and J. Rexford. Scalable network virtualization in software-defined networks. *Internet Computing, IEEE*, (99):1, 2012.
[13] Ulrich Elsner. Graph partitioning - a survey. *Survey*, 1997.
[14] C. Fuerst, S. Schmid, and A. Feldmann. On the benefit of collocation in virtual network embeddings (short paper). In *Proc. CLOUDNET*, 2012.
[15] C. Guo, G. Lu, D. Li, H. Wu, X. Zhang, Y. Shi, C. Tian, Y. Zhang, and S. Lu. BCube: A high performance, server-centric network architecture for modular data centers. In *Proc. ACM SIGCOMM*, 2009.
[16] C. Guo, G. Lu, H. J. Wang, S. Yang, C. Kong, P. Sun, W. Wu, and Y. Zhang. SecondNet: A data center network virtualization architecture with bandwidth guarantees. In *Proc. 6th CoNEXT*, 2010.
[17] Chuanxiong Guo, Haitao Wu, Kun Tan, Lei Shi, Yongguang Zhang, and Songwu Lu. DCell: A scalable and fault-tolerant network structure for data centers. In *Proc. ACM SIGCOMM*, pages 75–86, 2008.
[18] Robert Krauthgamer, Joseph (Seffi) Naor, and Roy Schwartz. Partitioning graphs into balanced components. In *Proc. 20th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 942–949, 2009.
[19] Jens Lischka and Holger Karl. A virtual network mapping algorithm based on subgraph isomorphism detection. In *Proc. VISA*, 2009.
[20] J. Lu and J. Turner. Efficient mapping of virtual networks onto a shared substrate. In *Technical Report, WUCSE-2006-35, Washington University*, 2006.
[21] Jeffrey C. Mogul and Lucian Popa. What we talk about when we talk about cloud network performance. *Computer Communication Review*, 42(5):44–48, 2012.
[22] Ripal Nathuji, Aman Kansal, and Alireza Ghaffarkhah. Q-clouds: Managing performance interference effects for qos-aware clouds. In *Proc. EuroSys*, 2010.
[23] Robert Ricci, Chris Alfeld, and Jay Lepreau. A solver for the network testbed mapping problem. *SIGCOMM CCR.*, 33(2), 2003.
[24] Thomas Ristenpart, Eran Tromer, Hovav Shacham, and Stefan Savage. Hey, you, get off of my cloud: exploring information leakage in third-party compute clouds. In *Proc. 16th ACM CCS*, pages 199–212, 2009.
[25] Gregor Schaffrath, Stefan Schmid, and Anja Feldmann. Optimizing long-lived cloudnets with migrations. In *Proc. IEEE/ACM UCC*, 2012.
[26] Guohui Wang and Eugene Ng. The impact of virtualization on network performance of amazon ec2 data center. In *Proc. INFOCOM*, 2010.
[27] Alan Williamson. Has amazon EC2 become over subscribed? http://alan.blog-city.com/has_amazon_ec2_become_over_subscribed.htm, 2013.
[28] Q. Zhu and T. Tung. A performance interference model for managing consolidated workloads in qos-aware clouds. In *CLOUD*, 2012.