A Distributed, Parallel, and Generic Virtual Network Embedding Framework

Michael Till Beck*‡, Andreas Fischer*, Hermann de Meer*
*University of Passau, Passau, Germany
{michael.beck,andreas.fischer,demeer}@uni-passau.de
‡Stadtwerke Passau GmbH, Passau, Germany

Juan Felipe Botero, Xavier Hesselbach
Universitat Politécnica de Catalunya, Barcelona, Spain
{jfbotero,xavierh}@entel.upc.edu

***Abstract*—One of the main challenges of network virtualization is the mapping of virtual network demands to physical network resources, commonly known as the virtual network embedding (VNE) problem. This paper introduces DPVNE, a *distributed*, *parallel* and *generic* VNE framework. DPVNE can be used 1) to run various cost-reducing embedding algorithms 2) in a distributed way. Thereby, computational load for embedding multiple virtual networks is spread across the substrate network reducing workload of individual nodes and 3) enabling the embedding of multiple virtual networks in parallel. DPVNE, in contrast to existing distributed algorithms, 4) achieves lower message overhead and, despite of being distributed, 5) keeps embedding costs comparable to those of centralized approaches.**

I. Introduction

To overcome the inflexibility of the current Internet, network virtualization (NV) has been recognized as a key technology for the Future Internet [2]. One main problem in NV lies in the mapping of virtual network resources to the substrate network (SN). Each virtual resource has to be assigned to one or several substrate resources. These, however, only have limited capacity, thereby restraining the number of virtual resources that can be hosted by a certain substrate resource. The optimal mapping of virtual network requests (VNRs) on top of the SN has been called Virtual Network Embedding (VNE) [3]. An efficient embedding is not only important in network virtualization; the EU funded project All4Green (www.all4green-project.eu) aims for efficiently allocating demands in data centers to minimize energy consumption accomplishing a fixed (static SLA) or elastic (green SLA) QoS guaranteed to the ICT end users.

VNE is $\mathcal{NP}$-hard [3]. As such, one has to refer to heuristic solutions. However, even heuristics do not scale well for large networks. Except for the distributed VNE approach presented in [5], that incurs in a huge message overhead, current proposals are centralized, resulting in scalability problems.

In this paper, we introduce DPVNE: a generic, fully Distributed and Parallel VNE approach. Contrary to centralized approaches where a single node computes the entire embedding, DPVNE has the advantage of making better use of available computing resources, letting multiple nodes (*embedder nodes*) calculate embeddings in parallel. Multiple substrate nodes, belonging to the SN, perform the actual embedding, instead of having a central instance that gets overwhelmed by receiving a large amount of requests. We introduce a hierarchical partitioning that allows to perform the VNE in smaller, cooperating parts of the SN. Some substrate nodes are defined to be the entry points of the embedding framework (*delegation nodes*), managing SN partitions. The VNRs are initially randomly distributed to these special nodes by a load balancer. Delegation nodes then embed the VNRs they receive, either by delegating it to other parts of the network, or by performing the embedding themselves.

More precisely, the main contributions of DPVNE are:

- **Distributed Embedding**: The VNE is not performed by a centralized *embedder node*. Instead, the network is partitioned into several parts and computational overhead is spread among them. In contrast to other distributed algorithms, message overhead is noticeably reduced.
- **Parallel Embedding**: Parts of the network can operate independently, so the SN is able to embed, in parallel, a set of VNRs, increasing the computation performance.
- **Generic Embedding Framework**: DPVNE is a general framework that can integrate various existing cost-optimizing VNE algorithms. Embedder nodes use one of the existing VNE algorithms to actually embed the VNR.

Since DPVNE is distributed and embeds multiple VNRs in parallel, it scales with respect to computational and messaging overhead – even for large network topologies.

II. Related Work

Due to the $\mathcal{NP}$-hardness of the VNE problem, current solutions are based on heuristics [9], [7], [1]. All those previous approaches share the following characteristics: they rely on a *centralized embedder node* to perform the VNE while they serve *just one VNR at a time*. Being centralized, they can benefit from full knowledge of the substrate network to compute a near-optimal VNE result without any communication overhead. However, they present a single point of failure.

The first distributed VNE approach that aims to guarantee load-balancing among all substrate nodes during the VN mapping was proposed in [5]. The proposal maps each VNR by subdividing it into a set of hub-and-spoke clusters where each cluster is handled by a root node in the SN. Although this proposal solves some of the weaknesses of centralized approaches, it suffers from introducing high message overhead.

A partly distributed approach is presented in [4]: the SN is split into different, hierarchic partitions. However, in contrast to DPVNE, this approach is 1) not fully distributed (it relies on a general manager and a set of sub-managers dealing with each partition level), and 2) it is not generic.

III. Distributed VNE Framework

In this section we present a distributed, parallel, generic VNE framework, based on hierarchical network partitioning.

A. Partition-based Distributed Embedding

The DPVNE framework uses a hierarchical partitioning scheme. By partitioning the network, the partitions can be used to perform embeddings independently. Therefore, the network partition has to be performed in a preparatory phase,

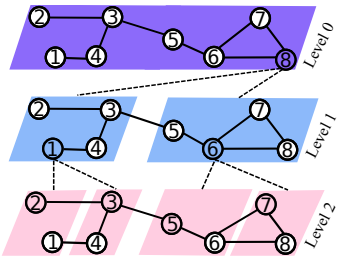


Fig. 1: Hierarchical Partitioning of a Network

before the first VNR arrives. DPVNE is, thus, divided in two stages: the *initialization of the substrate network phase* and the *embedding phase*.

1) Substrate Network Initialization Phase: This phase consists of three steps: a) Partitioning the SN, b) assigning delegation nodes, and c) setting up distributed lock trees.

a) Partitioning the SN: DPVNE looks for a hierarchical partitioning that provides, in each layer, a set of non-overlapping partitions. Partitions should be highly connected for increasing the likelihood that VNRs are assigned to well-connected substrate resources. Fig. 1 depicts an example.

We choose the *Multilevel Recursive Bisection* partitioning algorithm proposed by Karypis et al. in [6]. This algorithm identifies a specified number of isolated and non-overlapping partitions inside the substrate network's graph that have nearly the same amount of CPU and bandwidth (BW) resources. Additionally, it also aims at minimizing the number of links between partitions. This means that the algorithm tends to group nodes that are highly interconnected. Later, these partitions of nodes are used to embed virtual networks.

For each partition, the node with the highest available CPU resources is selected. We call this node a *clusterhead*. It runs the partitioning algorithm that builds isolated, non-overlapping partitions, and it will also run the actual embedding algorithm that the DPVNE implementation was assembled with. The procedure is recursively repeated: Within each smaller partition, another *clusterhead* (that has not been chosen as a clusterhead before) is selected. Each new clusterhead then divides the partition it is located in (cf. Fig. 1). For the sake of simplicity, partitioning is described as a centralized process here, however, it can also be done in a fully distributed way.

This way, a hierarchical partitioning structure of the substrate network is built. Network partitions are organized hierarchically allowing an easy distribution of the VNRs through the SN. Partitions located at the same hierarchy level do not intersect, and each partition encloses all nodes of its children. On each level, DPVNE's partitioning approach tries to find partitions having a similar amount of resources. It stops when the size of the partitions is 1 node. At the end each clusterhead is aware of the topology and available resources inside its partition and, as consequence, is able to run an embedding algorithm to compute the actual VNE.

b) Assigning Delegation Nodes: The network operator sets up a load balancer that randomly distributes VNRs. However, these requests are not sent to any arbitrary substrate node, but only to special ones: Some clusterheads inside the hierarchy will be in charge of receiving the VNRs and *delegating* them to the partition that fits better to perform the embedding. Those nodes are called *delegation nodes*. To increase the number of virtual networks that can be embedded in parallel, we define more than just one single node inside the partition hierarchy to be a delegation node.

If delegation nodes are situated at different levels in the hierarchy, problems can occur with overlapping areas of responsibility. E.g., if one delegation node is a child of another delegation node, both nodes will forward VNRs to clusterheads below both of them, leading to inconsistent information with regard to available resources. To avoid this kind of problem, it is assumed here that delegation nodes are all located on the same level in the hierarchy, taking care of mutually independent parts of the network (cf. Fig. 2).

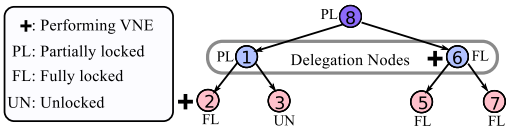


Fig. 2: Locking of network areas

c) Setting up Distributed Lock Trees: To perform parallel VNEs, a locking mechanism is implemented to prevent inconsistencies of clusterheads sharing parts of the same substrate topology. For example, in the scenario presented in Fig. 1, we may not run a mapping algorithm on the left partition of the level 1, while at the same time, another embedding is going on at the leftmost partition of the level 2. In contrast, we can continue at the right partition of the level 1 in parallel, since this area is independent. For this reason, each clusterhead maintains a data structure (locktree) where information about the state of partitions that are currently involved in a running embedding process is managed. Each local locktree only covers the set of clusterheads that the clusterhead itself communicates to (i.e., the subtree below the partition and the parent clusterhead). Each partition is in one of the following three states (cf. Fig. 2): fully locked (FL), partially locked (PL) or unlocked (UN). FL means that the clusterhead responsible for the partition is currently embedding a VNR. A PL clusterhead itself is not currently performing a VNE, however, one or more of its children are. Consequently, the partition can not embed a VNR at the moment, because it did not receive the latest, up-to-date resource information yet. Finally, UN clusterheads are not currently performing VNEs and their children neither.

2) Distributed and Parallel Embedding: After initializing the SN, the embedding process can take place. To reduce the number of exchanged messages, we confine the distribution of the VNR to the hierarchical subtree starting at the delegation node that has received the request.

a) Embedding within the scope of the delegation nodes: Each delegation node controls an independent part of the network. To avoid interferences, a delegation node may only delegate VN requests to clusterheads within its assigned partition scope (or hand it off to its parent node). So, when a VN request is received, the delegation node tries to find another underlying clusterhead within its partition scope that might be able to embed the VN request. The main requirements of a partition to perform a VNE are to have, at least, the same number of nodes than the VNR and to count on sufficient available resources. To check this, a simple heuristic calculating the potential π of a partition p (composed of a set of nodes $n \in N$ and links $l \in L$) to embed the VNR v^i (composed of a set of virtual nodes $n^i \in N^i$ and links $l^i \in L^i$) is used:

$$\pi = \frac{\sum_{n \in N} res_{cpu}(n)}{\sum_{n^i \in N^i} dem_{cpu}(n^i)} \cdot \frac{\sum_{l \in L} res_{bw}(l)}{\omega \cdot \sum_{l^i \in L^i} dem_{bw}(l^i)}$$

π is calculated by the delegation nodes for every underlying partition they know. It is a combination of CPU and BW potential. CPU potential is calculated by dividing available CPU resources ($res_{cpu}(n)$) by CPU demands ($dem_{cpu}(n^i)$).

Likewise, BW potential is calculated by dividing available BW ($res_{bw}(l)$) by demanded BW ($dem_{bw}(l^i)$). Since virtual links can span several physical links, the demanded BW is weighted with a parameter ω, which estimates how much physical BW will be used by a virtual link. A higher ω leads to a more pessimistic heuristic, expecting to spend more substrate resources to fulfill the demands. Note that this definition can easily be extended to take different constraints into account.

The potential tries to estimate whether a specific VNR can be embedded into a substrate partition. If the partition's number of nodes is sufficient and its potential is greater than 1, it is considered as a candidate to embed the VNR. There might be more than just one single partition able to embed the request. The *delegation nodes* start delegating the embedding from the partition in the bottom hierarchy level having the smallest complying π. The embedding is then performed by the clusterhead of that partition. This behaviour ensures both minimization of the embedding cost in the SN and maximization of the number of network areas that can be used for parallel VNRs: only the smallest possible network partitions are busy, while other parts of the SN can be used in parallel. In the meantime, when a clusterhead of an underlying partition is still busy calculating the VNE, the delegation node that forwarded the VNR can continue receiving further VNRs and delegate them to other idle clusterheads.

b) Embedding outside the scope of the delegation nodes: If the heuristic cannot find any suitable partition in the scope of the delegation node, the request is forwarded to the parent clusterhead. Clusterheads above the level of delegation nodes in general do not have up-to-date information (since some nodes within their scope are working in parallel and therefore the availability of resources might have changed in the meantime) and therefore do not use any heuristic to estimate to which of their children they forward the request to. They simply forward the request to all of their children (except the one that sent the request), one by one, and stop as soon as one of them can embed the VNE. The original child that failed to embed the request stops embedding further VNEs until its parent node sends a notification to continue (this is done as soon as the parent node receives the final embedding result). If none of the children is able to embed the VNE, the parent clusterhead tries to embed the VNE itself. In case of failure, it forwards the request to its parent clusterhead and waits for its answer. This parent clusterhead again operates in the same way. If there is no parent clusterhead to forward the request to, the highest level of the partition hierarchy has been reached and the VNE failed. In this case, the VNR is rejected.

c) Distributed VNE Protocol: To deal with the communication among the clusterheads of the hierarchy, a VNE protocol is proposed. To facilitate the comprehension of DPVNE, its behavior is illustrated in Fig. 3. Here, delegation nodes are chosen at level 1 of the partitions' hierarchy. The VNRs are processed for each time (t_x) as follows:

t₀: VNR arrives at delegation node B. Let's assume the heuristic tells the delegation node that there are not enough resources available inside none of its subpartitions. In this case, the delegation node tries to run the embedding algorithm inside its own partition. However, it is not able to map the VNR.

t₁: Since VNR can not be mapped inside the partitions of delegation node B, it delegates the request to its parent clusterhead A. There might be unapplied resource updates generated by previous embeddings that only B and some underlying clusterheads are aware of, so B can also include

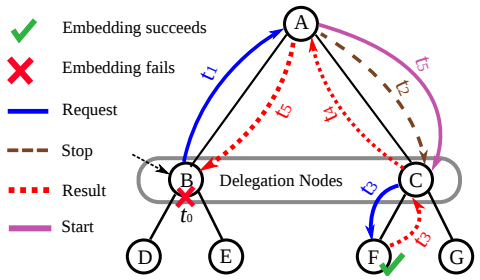


Fig. 3: Distributed Algorithm Behavior

update information if necessary. Clusterhead B fully locks all of its own partition scope locally, including its parent clusterhead, so it queues any other VNRs that arrive in the mean time until the embedding of the current VNR is done.

t₂: Upon receiving a delegated request, being a clusterhead located above the delegation nodes level, clusterhead A knows that B is unable to embed the VNR within its scope. So it fully locks the corresponding partition scope locally and randomly chooses one of its other, locally unlocked children and sends to it a STOP message asking for the embedding of the VNR. (Since we have a binary tree in this example, the only possibility here is C). When receiving a STOP message, newly incoming VNRs are queued by the delegation node (until a START message is received), the node finishes ongoing embeddings, and, finally, tries to embed the VNR that is part of the STOP message in the partition scope.

t₃: Delegation node C receives STOP from its parent. C stops receiving new requests, waits until all the VNRs being embedded inside its partition finish and, based on the potential, chooses the clusterhead F to delegate the VNR and perform the VNE. C fully locks node F locally; it also locks itself partially. In the next step, VNR is mapped in clusterhead F, which, in turn, communicates the result to clusterhead C.

t₄: After the VNE finishes, C locally unlocks the clusterheads again and notifies parent node A that the VNR was successfully embedded and which resource allocations changed. A receives the message, unlocks the locally locked clusterheads and applies the corresponding updates.

t₅: Node A forwards the result message to delegation node B, indicating that the embedding was performed successfully. In turn, it sends a START message to node C that was previously stopped. This message tells that C can continue processing VNRs. B unlocks the resources that were locked locally.

In summary, the DPVNE framework performs embeddings in parallel, distributing the load among different nodes in the SN. It first partitions the SN to create a hierarchy of SN partitions within which VNRs can be embedded. Each partition is coordinated by a clusterhead. Each clusterhead maintains a local lock tree, containing information about parts of the SN currently busy with an embedding. Some clusterheads are assigned as Delegation Nodes, receiving VNRs from the outside world and distributing them within the hierarchy. VNRs can be escalated to higher levels in the hierarchy, if a clusterhead does not have sufficient resources to satisfy the VNR. In the following section, efficiency of this approach will be evaluated and compared to other VNE algorithms.

IV. Evaluation

We evaluate various characteristics of DPVNE with regard to its distributed nature. Therefore, we compare it to the DAVNM algorithm, which is, to the best of our knowledge, the only other fully distributed VNE proposal up to now. We also analyze DPVNE concerning the quality of the embedding, in

particular embedding cost. To this end, we compare DPVNE to a set of well known centralized algorithms. In contrast to a distributed approach, the single embedder node of centralized approaches benefits from full knowledge of substrate topology.

In order to evaluate our approach, we used the ALEVIN simulation tool [1][2]. This tool facilitates the development, comparison, and analysis of VNE algorithms and comes with various implemented VNE algorithms. Since our framework focuses on cost optimization, we selected a set of appropriate VNE algorithms sharing the same goal: DAVNM [5], Baseline [9], ASID [7], RW-MM-SP [1], and RW-BFS [1]. As already stated, our approach is generic. For the following evaluation, DPVNE embedder nodes used ASID to perform VNE.

We partition the networks into two parts at each level, creating a binary tree hierarchy. The set of evaluation scenarios were created over random network topologies. This is in line with evaluations done in related work [9], [7]. To generate the random topologies, a Waxman generator [8], parametrized with edge probability 0.5 and long edge ratio 0.5 was employed. Substrate networks were generated with 100 nodes. In order to evaluate the effect of increased stress on the substrate network, we generated requests of different sizes (5, 10, 15, or 20 nodes per VNR). As in most related work, CPU power in nodes and bandwidth in links were chosen as constraints, but other kinds of constraints can also be used without constriction. Substrate nodes/links received a random CPU/bandwidth capacity between $1-100$, and virtual nodes/links a random CPU/bandwidth demand between $1-50$, which is similar to parameters presented in related work. To ensure stability of the results, 30 scenarios were generated.

We evaluated and compared various algorithms towards embedding costs. As in previous approaches, cost is defined as the sum of the SN resources used to allocate the VNR. However, costs heavily depend on the topology and the demands of VNRs. Since our topologies are random, costs vary and thus, direct comparison can be unfair. A better comparison strategy could be to additionally take the demands of VNRs into account by determining the revenue (defined as the sum of requested bandwidth and CPU of the VNRs). So, to provide an appropriate comparison, the cost/revenue (C/R) ratio is defined as the division of cost and revenue.

As discussed in Section I, our goal was to present a *distributed*, *scalable* and *parallel* VNE framework. From the obtained results, we summarize the following key observations (confidence level of all figures: 95%).

A. Reduction of message overhead by DPVNE

In general, there are two primary factors for the success of a *scalable* algorithm that operates in a *distributed* manner – the distribution of load (set of VNRs) and the message overhead generated by it: On the one hand it is preferable that load is spread across as many nodes as possible (parallel VNE). On the other hand, communication overhead should be low. Otherwise, the approach will not scale well. This is a problem of the DAVNM algorithm: DAVNM has a noticeable communication overhead in our simulations in order to complete the embeddings. Table Ia compares the message overhead of DAVNM to DPVNE with $\omega = 10$, using the root node as the only delegation node that distributes load. So, DPVNE comes with significantly lower message overhead.

[1]ALEVIN: http://sourceforge.net/projects/alevin/

[2]Full source code of DPVNE: http://net.fim.uni-passau.de/beck/DPVNE.zip

TABLE I

(a) coordination messages

#nodes per VNR	*DAVNM*	*DPVNE*
5	53554	18
10	61591	13
15	74918	13
20	93470	14

(b) Node stress

#nodes per VNR	*ASID*	*DPVNE*
5	0.4	0.3
10	1.9	0.5
15	4.0	0.8
20	6.9	1.0

B. Influence of pessimistic estimations

Various parameters influence the behaviour of DPVNE and also have an impact to the number of exchanged messages needed to find an appropriate embedding result. One important factor is a reasonable estimation of the available resources inside SN partitions. The potential π of a determined partition can be fine-tuned by its ω parameter: A higher value results in a more pessimistic evaluation of available substrate resources, a smaller value in a more optimistic one.

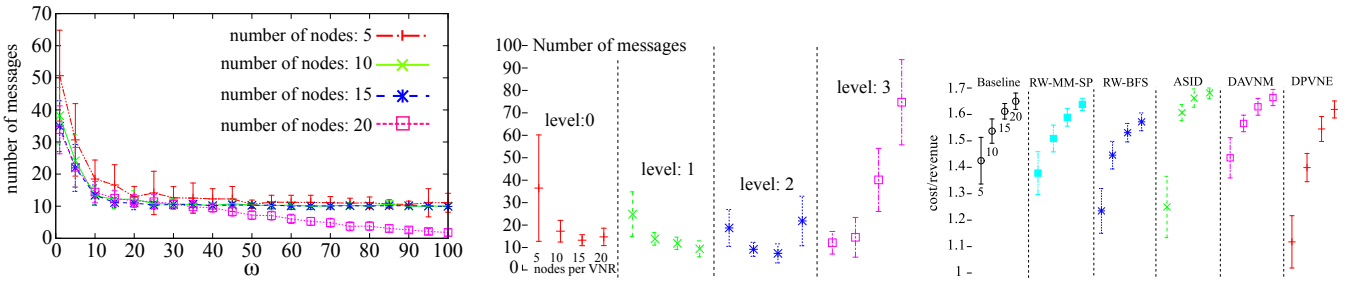
Fig. 4a shows the number of coordination messages that are exchanged between substrate nodes. This is evaluated for varying ω. Additionally, the size of the virtual networks was varied between 5 nodes and 20 nodes, and the root node was chosen as the only delegation node. Regardless of the size, there is a sharp drop up to $\omega = 10$ with the amount of messages staying at or below 20 messages for larger ω. This is because for lower ω, DPVNE acts in an optimistic way, trying to embed VNRs at levels too deep in the tree, leading to an increased messaging overhead, when oversized virtual networks have to be delegated to larger partitions. On the other hand, if ω is too large, eventually everything will be embedded by the root node, leading to lower message overhead (but increased centralization). In the figure, this is the case for 20 virtual nodes per network.

C. Influence of delegation node placement

For Fig. 4a, to have a clear presentation of results, we had chosen the root node (level 0) as the only delegation node. However, the number of messages also depends on where the delegation nodes are placed, because this influences which level of the hierarchy the VNRs are initially sent to and how they are delegated. Fig. 4b, shows the number of messages in dependence of the level where delegation nodes are placed. Each level is evaluated with varying load, having virtual networks with 5, 10, 15, or 20 nodes each. It can be seen that message overhead does not significantly change when delegation nodes were set to a lower level. The advantage of choosing multiple delegation nodes that delegate and spread computation to other clusterheads is the reduction of message passing overhead and the dependability on a single node. However, delegation nodes should not be placed too deep in the hierarchy. Otherwise, this has a negative effect to message overhead: As depicted in the figure, for level 3, message overhead starts to grow again, especially for bigger networks. The average number of nodes within one of the 2^3 partitions at level 3 is $100/2^3 \approx 13$. Since a partition of this size will, on average, not be able to embed a VNR that has 15 or 20 nodes, these VNRs have to be delegated to a higher hierarchy level – which results in additional message overhead.

D. Tradeoff - number of messages vs level of parallelism

Keeping the number of coordination messages at a low level is not the only aim of a VNE algorithm. Otherwise, an algorithm that only uses one single node to compute all the embeddings (like centralized algorithms do) would be ideal, already. For a distributed approach, however, VNRs should be appropriately distributed among substrate nodes, so that they



(a) # messages vs. ω (b) # messages vs delegation level (c) Cost/revenue (different algorithms)

Fig. 4: DPVNE Results

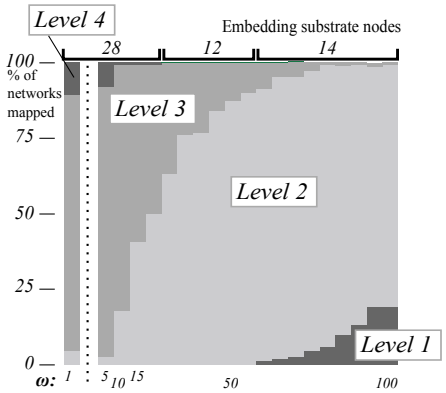


Fig. 5: Embedding involvement vs ω

can be embedded in parallel. To spread load equally, DPVNE organizes substrate nodes in a tree structure. As such, it is interesting to see which percentage of the requested virtual networks is mapped at which level in the tree. The deeper in the hierarchy, the more nodes will participate in the set of embeddings. E.g., if 10% of all networks are mapped at level 3, then $2^3 = 8$ nodes take 10% of the load. This is depicted in Fig. 5. In this figure, the load distribution between levels in the binary tree is shown in relation to ω, which is the main regulating factor in this approach. A higher ω means that estimates for embedding costs are more pessimistic, leading to networks being embedded at a higher level in the tree and, as a consequence, to a lower level of parallelism. The number of nodes participating in the embedding is shown at the top of the figure as "embedding substrate nodes". It can be seen that for higher ω there are less nodes participating in the embedding.

Taking Fig. 5 into account, one can conclude that $\omega = 10$ provides a good tradeoff between message overhead and load distribution. In comparision to more than 60000 messages generated by *DAVNM*, it becomes clear that our approach is a significant improvement in terms of message overhead.

E. Economical resource usage

Fig. 4c shows the C/R ratio for several VNE algorithms. It can be seen that the C/R ratio of our approach is comparable to the ratio of other approaches. On average, results are better than for the distributed DAVNM algorithm. In some cases, DPVNE even outperforms centralized approaches, especially for smaller virtual networks, although embedder nodes do not have full knowledge of the overall network topology.

It is even slightly better than the pure ASID, because embeddings are confined to highly connected partitions with just enough resources. Therefore, the possibility to find appropriate embedding candidates within the immediate surroundings of a node is higher than when just choosing arbitrary, random areas that are possibly already crowded (as ASID does). This is indicated by the average number of nodes that gets mapped to a substrate node (see Table Ib).

Simulation results showed that DPVNE drastically reduces the number of exchanged messages when compared with the other distributed alternative, DAVNM. They also showed that making pessimistic estimations and placing delegation nodes up in the hierarchy reduces the level of parallelism. There is a tradeoff between the amount of exchanged messages and the number of VNRs that can be embedded in parallel. Embedding costs are comparable with the ones of centralized approaches. With VNRs of low sizes, DPVNE even achieves better C/R ratios, due to the restriction of the VNEs to smaller and strongly interconnected SN partitions.

V. Conclusion and Future Work

We introduced a generic, distributed and parallel framework for embedding virtual networks that can be used in combination with cost-reducing embedding algorithms. We showed that message overhead can be kept significantly smaller than other distributed VNE solutions. Despite its distributed nature, embedding costs of DPVNE remain comparable to those of centralized algorithms. Simulation results also indicate a tradeoff between message overhead and the level of parallelism that can be achieved with DPVNE.

Future work will be devoted to the evaluation of DPVNE with various VNE algorithms. More sophisticated heuristics for estimating the potential π will be investigated. Also, different partitioning methods can be evaluated, and timing and load balancing aspects should be discussed in more detail.

Acknowledgments This work has received support by the EU projects All4Green (FP7 STREP, grant no. 288674), EINS (FP7 NoE, grant no. 288021), and the Spanish Government project MICINN (grant no. TIN2010-20136-C03-01).

References

[1] X. Cheng, S. Su, Z. Zhang, H. Wang, F. Yang, Y. Luo, and J. Wang. Virtual network embedding through topology-aware node ranking. *SIGCOMM Comput. Commun. Rev.*, 41:38–47, April 2011.

[2] N. Chowdhury and R. Boutaba. A survey of network virtualization. *Computer Networks*, 54(5):862 – 876, 2010.

[3] A. Fischer, J. Botero, M. Beck, H. de Meer, and X. Hesselbach. Virtual network embedding: A survey. *Communications Surveys Tutorials, IEEE*, PP(99):1 –19, 2013.

[4] T. Ghazar and N. Samaan. Hierarchical approach for efficient virtual network embedding based on exact subgraph matching. In *GLOBECOM*, pages 1–6. IEEE, 2011.

[5] I. Houidi, W. Louati, and D. Zeghlache. A distributed virtual network mapping algorithm. In *Communications, 2008. ICC '08. IEEE International Conference on Communications*, pages 5634 –5640, may 2008.

[6] G. Karypis, V. Kumar, and V. Kumar. Multilevel k-way partitioning scheme for irregular graphs. *Journal of Parallel and Distributed Computing*, 48:96–129, 1998.

[7] J. Lischka and H. Karl. A virtual network mapping algorithm based on subgraph isomorphism detection. In *VISA*, pages 81–88, 2009.

[8] B. M. Waxman. Routing of multipoint connections. *IEEE Journal on selected areas in communications*, 6(9), 1988.

[9] 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):17–29, Apr. 2008.