On the Benefit of Collocation in Virtual Network Embeddings

Carlo Fuerst, Stefan Schmid, Anja Feldmann
Telekom Innovation Laboratories & TU Berlin, Germany

***Abstract*—This paper attends to the problem of optimizing the resource allocation in data centers or cloud networks. Our work is motivated by the observation that existing virtual network (VNet) embedding algorithms typically waste link resources by embedding communicating virtual nodes to remote locations in the substrate. A simple alternative VNet embedding algorithm LoCo is proposed which seeks to automatically collocate communicating virtual nodes. We compare the performance of LoCo to the *SecondNet* [4] algorithm in different scenarios, and find that LoCo can often improve the resource efficiency.**

I. Introduction

Network virtualization is a new networking paradigm which combines node virtualization (e.g., VMWare, Xen, KVM) with link virtualization (e.g., OpenFlow, VLANs, MPLS). Essentially, a *virtual network* (VNet) specifies a graph consisting of virtual nodes (or *virtual machines (VMs)*) connected by virtual links. Both the virtual nodes and links come with resource requirements (e.g., memory, CPU, bandwidth). A major challenge of this paradigm regards the resource allocation: Given the VNet specifications, where to efficiently *embed* VNets on the substrate network (e.g., a physical infrastructure like a datacenter or an ISP network, or again a VNet)?

This paper attends to the VNet embedding problem and argues that prohibiting virtual node *collocations* upfront, as done by most current algorithms in the literature, can be suboptimal. Rather, the possibility to specify which elements of a virtual network can be *collocated* is an effective way to increase the resource efficiency: *collocation-enabled* algorithms can often embed more VNets. To confirm our claim, we propose a very simple embedding algorithm called LoCo which *automatically* groups virtual nodes to be mapped to the same substrate node. We show that given appropriate isolation mechanisms are in place (e.g., by mapping VM computations to different cores), LoCo yields an improved link resource allocation and achieves a higher number of embeddable VNets compared to the state-of-the-art *SecondNet* [4] embedding algorithm.

II. Virtual Network Embedding Model

We attend to the standard virtual network embedding problem in the literature (e.g., [2], [3], [6], [7]) where we are given (as input) a VNet topology and a substrate graph, and we should compute an allocation of the VNet on the substrate.

Formally, a *virtual network*, or short *VNet*, is essentially a graph $G = (V, E)$ (the *guest graph*) where V represents the set of virtual nodes or virtual machines (VMs) and E represents communication links. Both the nodes $v \in V(G)$ and the links $e \in E(G)$ may come with certain resource requirements. For example, a node $v \in V(G)$ may request a certain amount of RAM or CPU, and a link $e \in E(G)$ may request a certain amount of bandwidth. We will denote the resource requirements of a node or link by $r(v)$ and $r(e)$ respectively. A VNet G must be realized, i.e. *embedded* on a given substrate network which can again be modeled as a graph $H = (V, E)$ (the *host graph*). The nodes and links of the substrate network have certain capacities, which we will denote by $c(v)$ and $c(e)$, respectively.

There are different ways to embed a VNet G on a substrate H. While virtual nodes $v \in V(G)$ are typically mapped to exactly one substrate node $v' \in V(H)$, a virtual link $e \in E(G)$ can be mapped to one or even a linear combination of paths in H. We say that an embedding of a set of VNets is *valid* if the the allocations on all substrate nodes and links does not exceed the corresponding capacities.

Of course, the more substrate edges involved in the realization of a virtual link, the higher the embedding costs. We define the embedding cost as follows:

Definition 2.1 (Embedding Costs): Let $\Pi(e) = \{\pi_1, \pi_2, \ldots\}$ for some $e \in E(G)$ denote the set of substrate paths over which e is realized (i.e., embedded). Let $f(\pi)$ for some $\pi \in \Pi(e)$ denote the fraction of flow over path π, and let $\lambda(e)$ denote the length of e in terms of number of hops. The cost of embedding a VNet $G = (V, E)$ on a substrate H is defined as

$$\text{Cost} = \sum_{e \in E(G)} \sum_{\pi \in \Pi(e)} f(\pi) \cdot \lambda(e)$$

In other words, the allocation cost is simply the weighted distance of the different paths used by the virtual edges.

We, in this paper, will focus on embeddings where virtual links can only be mapped to a single path. Although it is easy to generalize our results to embeddings with parallel paths, we believe that a single path model has several advantages, especially in QoS environments where packet delays and orders should be predictable well.

III. The LoCo Algorithm

We now introduce our simple algorithm LoCo that leverages the potential of node collocation. Unlike many existing algorithms, LoCo does not separate the node and link mapping into two stages, but rather alternates between mapping

single virtual nodes and virtual links. That is, LOCO computes locations for virtual nodes one after another and in a *breadth-first* and "graph-isomorphism" [5] aware manner. In doing so, LOCO does not only take into account substrate node capacities, but also substrate link capacities. This avoids collocating subgraphs whose edge cut exceeds bandwidth constraints. We will refer to this strategy as *forward checking*.

In order to embed a virtual network G (the "guest graph"), first an arbitrary (in our simulations: random) start node $s \in V(G)$ is mapped to an arbitrary (or random) substrate node (i.e., on the "host graph") with sufficient capacity. (Note that there may be smarter strategies to select the first location, but we stick to this simple scheme as it already yields good results.) 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 performs a simple heuristic and backtracks over all 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 to avoid high runtimes.

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 $\{u, v\}$ $\forall$ $v \in M$**, where** $\{u, v\} \in E(G)$
    $M = M \cup \{u\}$ **and** $P = P \setminus \{u\}$
  end while
  if (embedding failed), **backtrack** on s

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).

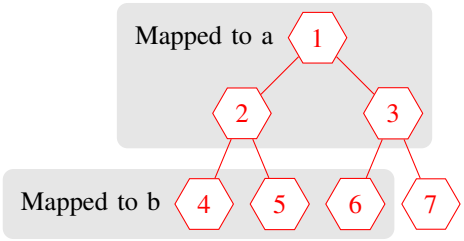

Fig. 1. Illustration of forward checking.

Figure 1 illustrates the need for forward checking: Here, a virtual tree network should be embedded on a fat-tree substrate topology, where all substrate nodes can host three virtual nodes, and all substrate link can host three virtual links. Say node s is Node 1 which is mapped to a Node a. Obviously, Node 2 and Node 3 can also be mapped to Node a since this does not violate resource constraints. Subsequently, three leaf nodes are mapped to a substrate node, and the fourth leaf node cannot be mapped, since Node a and the link which connects Node a are already fully utilized. Therefore, a naive algorithm would run into backtracking and face a similar problem for a different substrate node. *Forward checking* avoids this by verifying that all adjacent virtual links can be embedded before we map a virtual node to a substrate node. This allows the mapping of Node 1 and Node 2 to the same substrate node, but interdicts the mapping of Node 3 to the same substrate node: This would require mapping four virtual links to the substrate link connecting the substrate node to a switch. Node 4 could however be collocated with Node 1 and Node 2, since they only have two virtual links to virtual nodes which are mapped to different locations.

IV. EVALUATION

This section reports on our simulation results in different scenarios. We implemented a discrete event simulator (in Ruby) for LOCO. For comparison, we used the SecondNet[1] implementation which is publicly available.

Setup We will focus on a *fat-tree substrate topology* [1] with 128 nodes and 80 switches, but similar results can be expected in other symmetric or container-based data center topologies such as *BCube*, *DCell* or *MDCube* where LOCO computes local allocations. To model virtual network requests, as a simplification, we use jobs from the *Google cluster data set*.[2] Interestingly, this data set also contains information on which virtual nodes should not be collocated: the `different-machine constraint` flag indicates that a

[1]See `http://www.stanford.edu/ shyang/Site/vdcalloc.zip`.
[2]See `http://code.google.com/p/googleclusterdata/`. Unfortunately, the data contains only node resource information.

task must be scheduled to execute on a different machine than any other currently running tasks in the job. We observe that over 90% of all nodes do allow for node collocation. In the following, in order to focus on the collocation benefits more generally, we will simply assume all VNet nodes can be collocated. Moreover, we observe that about 80% of the jobs consist of 21 nodes or less. Since our substrate is smaller then the cell on which the data set was generated, we decided to reduce the limit the size of our requests to $[3, 10]$. This a similar to the the setting in other VNet embedding papers. [2]

Since the data set does not provide task connectivity information, two canonic VNet topologies are considered: the *master-slave (or star) network* and the *clique ("full-mesh") network*. To investigate also the performance of the algorithms on topologies with a low maximum degree, for comparison, we also include random binary trees in the input sequence. Moreover, VNet elements request resources uniformly at random from $\{1, 2, 4\}$. The substrate has homogenous resources on all substrate links and substrate nodes.

We model a challenging situation where the VNet demand roughly matches resource supply. This situation allows us to study the influences of existing embeddings on the incoming requests that cannot completely fill the gaps of previously rejected VNets. To achieve this, we create a sequence of VNets $S = (G_1, G_2, \ldots, G_n)$, where each VNet is randomly chosen from the above described requests and resources. To initially fill the substrate, we initially embed a maximal set of random VNets. Whenever a VNet expires, we immediately schedule a next VNet request if the total amount of requests virtual node resources would be otherwise lower than the overall node resources in the substrate.

Resource Efficiency Figure 2 provides a general overview of the utilization results, under different substrate node and link capacities. We find that SecondNet typically yields relatively good embeddings only if the substrate link capacities are at least twice as high as the node capacities. Obviously, a similar phenomenon can be expected from *any* embedding algorithm which avoids collocation and hence incurs higher communication costs.

While LoCo and SecondNet exhibit a similar dependency on link resources, LoCo benefits relatively more from higher node resources—although these indirectly increase the amount of virtual networks.

Regarding the variance, we find that lower resources typically lead to higher fluctuations. This can be explained by the stress put on the network by our VNet arrival sequence which always seeks to keep the load high. The more resources are available, the smaller the probability that an unfortunately placed VNet blocks a significant share of the network.

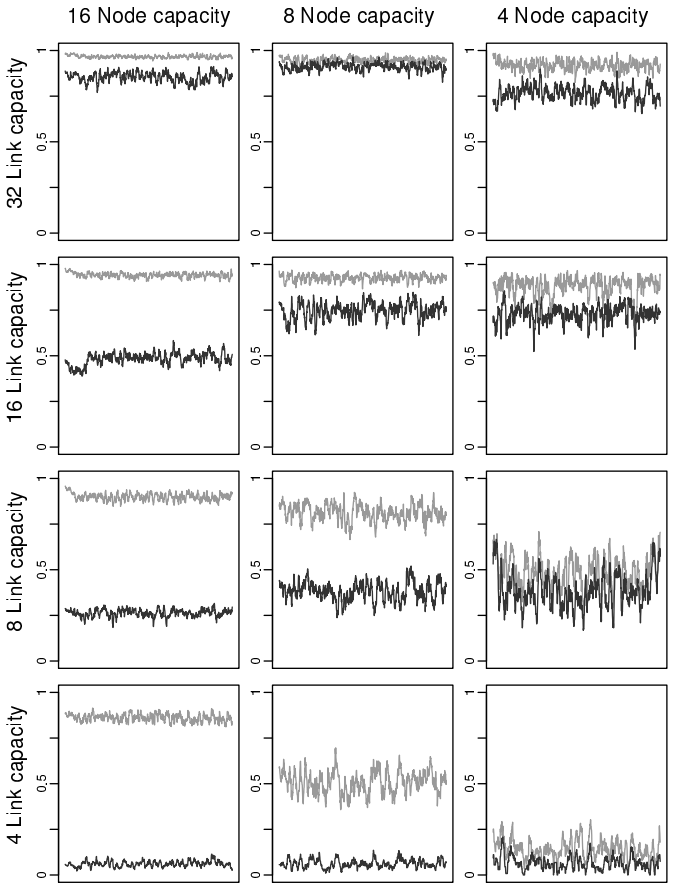

Fig. 2. Embedding efficiency (y - axis) of LoCo (*grey, higher*) vs SecondNet (*grey, higher*) under different node and link capacities. The x - axis shows discretised time

V. Conclusion

This paper has studied the potential benefit of collocating virtual nodes (with a focus on data center networks). We presented a simple algorithm LoCo that reveals potential benefits of node collocation. Obviously node collocation is not suitable for all use-cases. For example customers with stringent requirements on fault-tolerant embeddings will not allow that, e.g., all data mirrors are hosted on the same physical disk. Nevertheless, by allowing the customer to specify which of her virtual nodes may be collocated with other nodes, we can create a monetary advantage for the VNet provider (or even the flexible customers).

References

[1] M. Al-Fares, A. Loukissas, and A. Vahdat. A scalable, commodity data center network architecture. In *Proc. ACM SIGCOMM*, pages 63–74, 2008.

[2] K. Chowdhury, M. R. Rahman, and R. Boutaba. Virtual network embedding with coordinated node and link mapping. In *Proc. INFOCOM 2009 and IEEE/ACM Trans. Netw. (ToN) 2012*, 2012.

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

[4] 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.

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

[6] A. Ludwig, S. Schmid, and A. Feldmann. The price of specificity in the age of network virtualization (short paper). In *Proc. 5th IEEE/ACM UCC*, 2012.

[7] G. Schaffrath, S. Schmid, and A. Feldmann. Optimizing long-lived cloudnets with migrations. In *Proc. 5th IEEE/ACM UCC*, 2012.